[redacted] Final Year Project
Linking Keystrokes to Writing Processes and Language Quality

BSc (Hons) Computer Science with a Year in Industry

Author: Cheng Lyn Yi

Supervisor: [redacted]

Student ID: [redacted]

[redacted]

Abstract

Despite the widespread use of essay writing in educational assessment, existing Automated Essay Scoring (AES) systems primarily evaluate the final written product, often overlooking how the text was composed. This product-based approach fails to capture essential aspects of the writing process—such as planning, fluency, revision, and cognitive effort—that are critical for understanding learner behavior and writing development.

This study leverages keystroke logging to analyze real-time writing behaviors from 2,471 sessions in the publicly available Linking Writing Processes to Writing Quality Kaggle dataset. Participants completed SAT-inspired persuasive writing tasks under timed conditions. A total of 383 keystroke features were extracted based on prior AES research, leaderboard solutions, and established writing analytics. In addition, 843 novel features were engineered to capture fine-grained, process-oriented behaviors, resulting in a combined set of 1,226 features.

LightGBM regression models were trained to predict essay scores, with feature contributions evaluated through ablation studies, SHAP values, and LightGBM gain. Results show that a subset of 14 novel keystroke features, when combined with the original baseline features, significantly outperforms the baseline model alone. Among these, three features were consistently identified as reliable and impactful across interpretability analyses, demonstrating the added predictive value of writing process data in AES systems.

This research highlights the potential of keystroke logging to enrich automated scoring by offering deeper insights into the cognitive and behavioral dimensions of writing, supporting the development of more transparent, learner-aware, and pedagogically meaningful assessment technologies.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Cheng Lyn Yi

[redacted]

Acknowledgements

I would like to thank my supervisor, [redacted] for her continued support, guidance, and encouragement throughout this project. Her expertise in machine learning and insightful feedback were invaluable in shaping both the direction and execution of this work.

Contents

Chapter 1

Introduction

1.1 Motivation

Essay writing plays a central role in education and is widely used in formal assessments to evaluate a student's ability to construct logical, well-reasoned arguments. Despite its significance, conventional assessment practices tend to focus solely on the final written product, often overlooking how the text was composed. This narrow focus limits our understanding of the writing process itself and the difficulties that learners may encounter during composition.

As digital writing environments become increasingly common, new opportunities have emerged for studying writing as a dynamic, unfolding process. Keystroke logging is one such method: it records each key press and release, along with user interactions such as cursor movements and editing actions, all timestamped in real time. This detailed data allows researchers to reconstruct the sequence and rhythm of writing, offering insight into a writer's fluency, planning, and revision strategies. These insights are particularly valuable for understanding writing development and learner behaviour.

There is growing interest in leveraging this type of data within Informatics and educational technology, particularly in the development of intelligent, learner-aware systems. Existing automated writing evaluation (AWE) tools often rely on surface-level textual features and lack sensitivity to writing effort or process. By incorporating keystroke-based features, such systems could become more interpretable, transparent, and responsive to learner needs.

This approach is especially relevant in the context of English language learning, where both native and non-native writers can benefit from feedback that reflects their writing process as well as their written output. Beyond academic settings, the ability to model writing behaviour at scale has promising applications in commercial educational platforms, language testing systems, and adaptive learning technologies, where personalised support and fair assessment are key priorities.

This project investigates how keystroke data can be used to identify novel and distinctive features of writing behaviour, and how these features relate to writing quality. It contributes to a broader movement in Informatics toward data-driven understandings of human behaviour, while addressing real-world educational challenges through interpretable and scalable AI modelling techniques.

1.2 Scope

This project is framed within the context of English language learning, encompassing both native and non-native writers engaged in academic writing development. It draws on the Kaggle "Linking Writing Processes to Writing Quality" [36] dataset, which includes keystroke logs and associated essay scores for responses to SAT-style [16] prompts. These prompts are designed to elicit argumentative writing in academic English under timed conditions, simulating the structure of standardised assessments commonly used in educational settings.

In this study, writing quality is defined as the score assigned to each essay, as provided within the dataset. These scores serve as the target variable for model training and evaluation, offering a practical and interpretable measure of writing proficiency for the purposes of this research.

The project focuses on two main areas:

- Engineering novel features from keystroke logs that reflect writing behaviours, guided by theories of writing and language production.
- Analysing the relationship between these features and writing quality, and exploring their potential to improve automated scoring models.

Although the feature design is informed by cognitive theories of writing, the project remains primarily applied in nature, with the goal of advancing automated assessment methods within an Informatics framework focused on educational technology. It does not aim to directly investigate broader theoretical questions about cognition or linguistic complexity, as explored in psycholinguistic research, but it may nonetheless offer insights that are relevant to future work in those domains.

1.3 Objectives

This project aims to:

1. Develop novel patterns in keystroke data as features for use in automated language assessment models.

2. Build and evaluate machine learning models that predict essay scores based on the engineered features.

3. Analyse the relationship between keystroke features and writing quality, and explore their contribution to improving automated scoring performance.

1.4 Report Structure

1.4.1 Report Structure

In the remainder of this report, the background and context of the project are introduced, along with a review of relevant literature on keystroke logging, writing process models, and the integration of process-based features in automated essay scoring. This provides the theoretical foundation for the work. The report then presents a structured feature engineering approach, introducing over 1,226 features derived from keystroke logs and organised into six categories.

Following this, the design of the project experiment is explained, outlining each stage of the machine learning pipeline. The implementation chapter describes the technical development of the system, with a focus on environment setup, modular code organisation, and reproducibility.

The results and evaluation chapter reports the outcomes of model training and evaluation, highlights key findings, discusses observed limitations, and proposes directions for future work. Legal, social, ethical, and professional considerations related to the use of keystroke data are then explored. Finally, the report concludes by summarizing the project's main contributions.

Supplementary materials, including extended results and code documentation, are provided in the appendices.

Chapter 2

Literature Review and Background

This chapter outlines the theoretical and methodological foundations of the project through a review of literature on cognitive writing models, keystroke logging, and the use of process features in automated essay scoring. It highlights key limitations in current approaches and motivates the project's structured feature design.

2.1 Measuring Writing Processes with Keystroke Data

Understanding writing as a process, rather than solely as a product, has long been central in cognitive writing research. Foundational models such as Hayes and Flower (1996) conceptualise writing as a recursive activity involving planning, translating, reviewing, and transcribing, all coordinated through working memory and influenced by the task environment [32]. These frameworks provide a theoretical basis for interpreting observable writing behaviours as reflections of underlying cognitive activity. [15, 40]

Keystroke logging has emerged as a practical method for studying this process in real time. It records a user's keyboard and mouse interactions with high temporal precision, typically down to the millisecond. This produces a detailed timeline of how the text was composed, offering insight into how writing unfolds over time. While this timeline can reveal valuable behavioural patterns, its interpretation is not always straightforward. As [28] point out, a single pause may reflect anything from planning high-level structure to hesitating over a word's spelling, while a burst of typing may indicate drafting, note-taking, or revision. This highlights the risk of assuming a direct one-to-one mapping between keystroke patterns and cognitive processes. For this reason, the features proposed in this project are not intended to model cognition directly. Instead, theoretical models are used to inform the design of features, with a focus on their predictive utility within automated scoring systems.

Compared to traditional observational techniques such as think-aloud protocols, eye-tracking, retrospective interviews, or video recordings, keystroke logging is unobtrusive, scalable, and objective. It allows writers to work naturally without disruption, which makes it particularly well-suited for large-scale or ecologically valid studies of writing. [40]

Within writing analytics, keystroke data has been applied in a range of contexts beyond assessment. Studies have used it to classify writing patterns [52], characterize writing processes [42], and cluster writers based on process strategies [49]. Features have also been explored in detecting engagement [11] and hesitation [28, 45]; identifying learning difficulties [10]; and supporting biometric applications such as authentication [1] and deception detection [8]. These diverse uses demonstrate the value of keystroke data for modeling human behavior. However, they also reveal inconsistencies in how keystroke features are defined and interpreted across domains.

Despite the growing interest in keystroke logging, a major challenge remains in the lack of standardization in feature definitions. Terms such as pause, burst, and inter-keystroke interval (IKI) are often used inconsistently, with thresholds and operational definitions varying between studies. This makes it difficult to compare findings, replicate analyses, or generalise results. To address this, the present project adopts a structured approach to feature engineering based on a consistent taxonomy, introduced in the following chapter 3. The goal is to bring clarity and interoperability to keystroke-derived features, particularly in relation to writing quality.

2.2 Keystroke Features and Writing Quality

Despite its potential, keystroke-derived data remains underutilized in automated essay scoring (AES). Most AES systems continue to rely on features extracted solely from the final written product, such as word count, lexical diversity, syntactic complexity, or coherence. These features offer insight into writing proficiency but do not reflect the process through which the text was produced.

A wide range of modeling approaches has been used to predict writing quality or essay scores. These include traditional statistical techniques such as linear regression [3, 48] and logistic regression [27], as well as machine learning algorithms like random forests [18], support vector machines and regressor [18], Naive Bayes [18], and XGBoost [42]. Deep learning architectures have also gained traction, including BiLSTM neural networks [21], long short-term memory (LSTM) networks [2, 33], and convolutional neural networks (CNNs) [33]. More recently, transformer-based models such as BERT [51] and Open-AI's GPT-3 [44] have also been introduced to leverage semantic and syntactic patterns in final text. However, most of these models do not incorporate behavioural signals from the writing process itself.

A small number of studies have begun exploring how keystroke-based features might contribute to AES [18, 23, 33, 48]. For example, submissions to the Kaggle competition included features based on anonymised keystroke logs, but these often focused on reconstructed product-level statistics [34]. Other studies have explored pause patterns, revision behaviors, and burst dynamics, though typically with limited feature sets and without consistent definitions [18, 23, 33, 48]. As prior work shows, even small changes in writing task design such as whether a prompt is persuasive or expository, or the complexity and structure of the instructions can alter the cognitive demands placed on writers [19]. These variations affect which keystroke features may be relevant and limit the generalisability of results across tasks.

While these studies show some promise, many rely on exploratory or ad hoc approaches. Feature definitions are often broad, and little distinction is made between types of pauses or revisions, or the locations in which they occur. This lack of structure makes it difficult to evaluate which behavioral signals are consistently predictive of writing quality.

To address this, the current project proposes a structured and interpretable set of keystroke-based features specifically designed for AES. While informed by prior research and theoretical models of writing, the majority of these features are novel and have not been previously applied in automated essay scoring contexts. They aim to capture meaningful aspects of the writing process in a consistent and replicable way, supporting the integration of process-based signals into scoring models.

As an outcome of this literature review, the next chapter A.2 details the full set of engineered features, organized into six behavioral categories.

Chapter 3

Feature Engineering

This chapter presents the full set of features engineered for this project. In total, **1,226 features** were extracted. Of these, **843** are novel features developed specifically for this study. These novel features address gaps in the literature and, to the best of my knowledge, have not yet been applied in automated essay scoring (AES) models. An additional **78** features are adapted from existing research in writing analytics, while **305** were replicated from top-performing submissions on the Kaggle leaderboard for the competition dataset.

All features are grouped into six categories:

1. **Pause Dynamics**: features capturing the timing and distribution of pauses during writing.

2. **Revision**: features related to revision behavior, including deletions, and editing actions.

3. **Verbosity**: features quantifying the volume of writing, measured through keystroke counts and distribution.

4. **Fluency**: features characterizing the rhythm and flow of writing, primarily through burst patterns.

5. **Other Events**: features capturing user interactions not tied to character input, such as paste actions, cursor jumps, or external model outputs.

6. **Product-related**: features derived from the final written text, reconstructed from the keystroke log.

The following sections provide a detailed breakdown of the features within each category and their respective sources.

3.1 Features Related to Pause Dynamics

This category includes 235 features related to the timing and distributions of pauses during writing. Features capture latencies across multiple unit levels, including within-word transitions, and word, sentence, and paragraph boundaries. To account for rapid typing behavior, additional features measure n-gram key sequences (digraphs and trigraphs) as well as overlapping keystrokes. Session-level timings are also included.

The terminology used to describe timing-based features is often inconsistent across studies. To ensure clarity, Figure 3.1 provides an overview of the time-based features included in this category. Additional visualizations are also included for features at sentence boundaries (Figure 3.2) and paragraph boundaries (Figure 3.3).

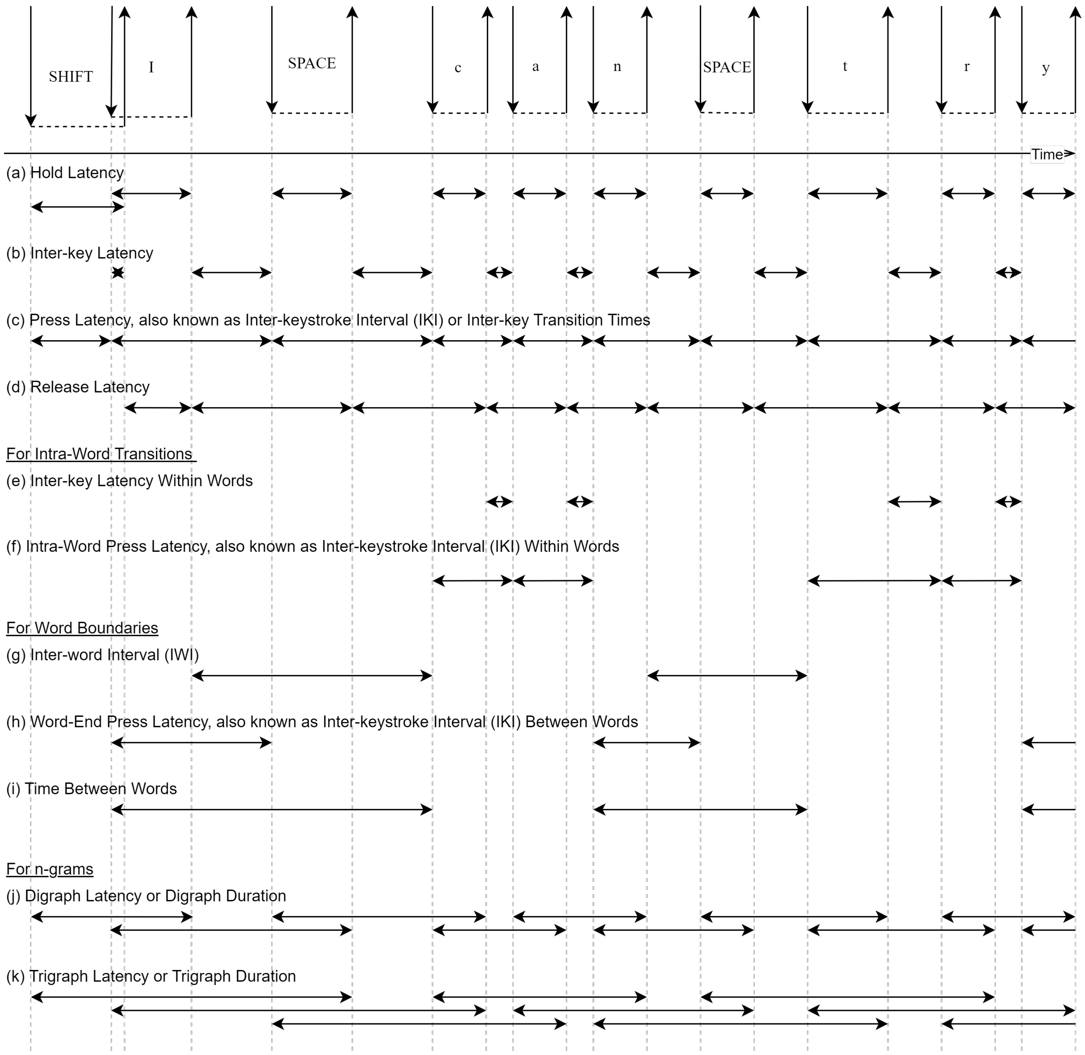
SHIFT
I
SPACE
c
a
n
SPACE
t
r
y
Time
(a) Hold Latency
(b) Inter-key Latency
(c) Press Latency, also known as Inter-keystroke Interval (IKI) or Inter-key Transition Times
(d) Release Latency
For Intra-Word Transitions
(e) Inter-key Latency Within Words
(f) Intra-Word Press Latency, also known as Inter-keystroke Interval (IKI) Within Words
For Word Boundaries
(g) Inter-word Interval (IWI)
(h) Word-End Press Latency, also known as Inter-keystroke Interval (IKI) Between Words
(i) Time Between Words
For n-grams
(j) Digraph Latency or Digraph Duration
(k) Trigraph Latency or Trigraph Duration

Figure 3.1: Time-based features extracted from keystroke logs of typing ”I can try”

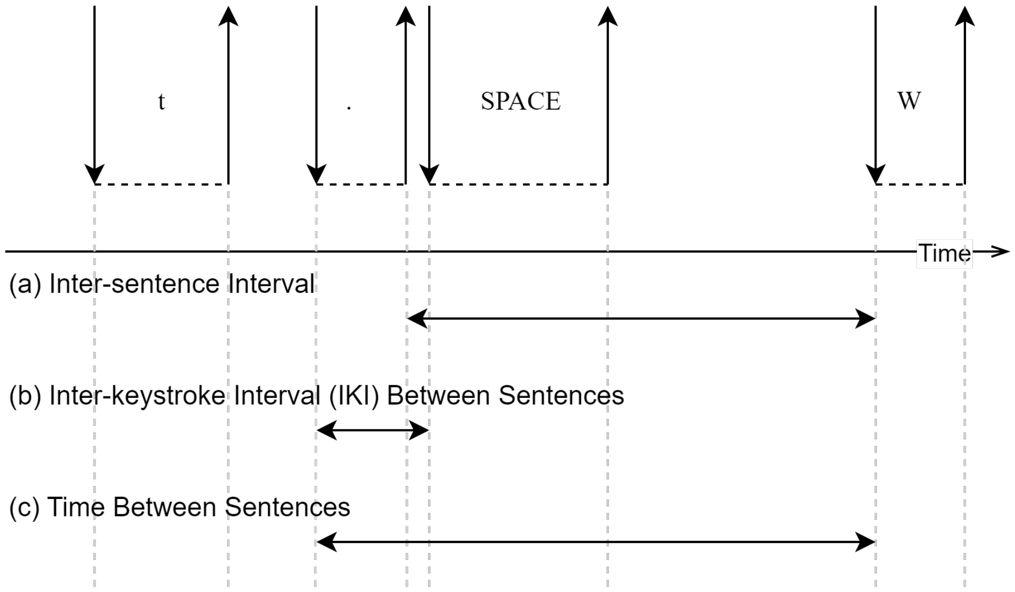
t
.
SPACE
W
Time
(a) Inter-sentence Interval
(b) Inter-keystroke Interval (IKI) Between Sentences
(c) Time Between Sentences

Figure 3.2: Time-based features extracted at a sentence boundary.

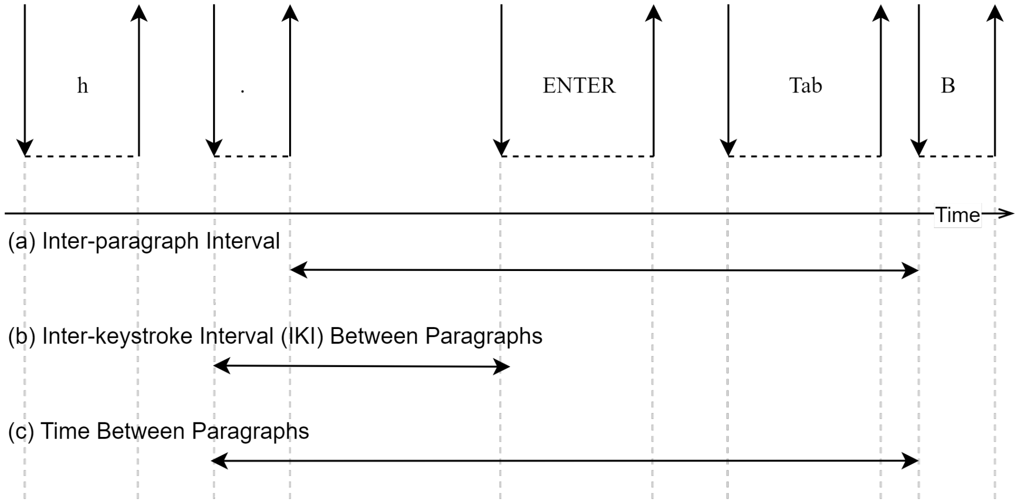
h
.
ENTER
Tab
B
Time
(a) Inter-paragraph Interval
(b) Inter-keystroke Interval (IKI) Between Paragraphs
(c) Time Between Paragraphs

Figure 3.3: Time-based features extracted at a paragraph boundary.

Kaggle-Leaderboard Features

- **Total Time**: Time from the first keypress to the last key release, considered the total duration of the session [11, 18, 23, 30, 49, 50, 53].
- **Hold Latency (Sum, Maximum, Mean, Standard Deviation, Kurtosis, Quantile)**: Metrics of the time from a single key press to its release [1].
- **Inter-key Latency (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantile, Skewness)**: Metrics of the time from one key release to the next

key press [1, 49].

- **Press Latency (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantile, Skewness)**: Metrics of the time from one key press to the next key press [1, 3, 18, 19, 48, 49, 50, 53]. Also known as Inter-keystroke Interval (IKI) or Inter-key Transition Time in some literatures [18, 19, 53].

- **Release Latency (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantile, Skewness)**: Metrics of the time from one key release to the next key release [1].

- **Number of pauses across various time intervals**: Pauses are defined based on inter-key latencies. A total of 10 inter-key latency intervals were extracted, including the number of pauses between 0–50 ms, 50–100 ms, 100–250 ms, 250–500 ms, 0.5–1 s, 1–1.5 s, 1.5–2 s, 2–2.5 s, 2.5–3 s, and greater than 3 s [3, 18, 19].

Existing Literature Features

- **Initial Pause Time**: Time from the first recorded keypress (considered the start of the session) to the keypress of the first Input event [3, 18, 48].

- **Intra-Word Press Latency (Mean, Standard Deviation)**: Metrics of the time from one key press to the next key press within a word, excluding transitions involving spaces and word boundaries [18, 19, 23, 48, 49]. Also known as Inter-keystroke Interval (IKI) Within Words in some literatures [18, 19].

- **Inter-word Interval (IWI) (Mean)**: Metrics of the time between the release of the last key event of one word and the keypress of the first character in the next word [8, 19, 52].

- **Word-End Press Latency (Mean, Standard Deviation)**: Metrics of the time between consecutive keystrokes that span word boundaries, specifically from the keypress of the last character in one word to the keypress of the subsequent space [18, 23, 48, 49]. Also known as Inter-keystroke Interval (IKI) Between Words in some literatures [18].

- **Time Between Words (Mean, Standard Deviation)**: Metrics of the time from the keypress of the last character in one word to the keypress of the first character in the next word [18, 23, 30, 49, 52].

- **Time Between Sentences (Mean, Standard Deviation)**: Metrics of the time from the keypress of the final punctuation in one sentence (considered the sentence end) to the keypress of the first character in the next sentence [18, 23, 28, 49].
- **Digraph Duration (Mean, Standard Deviation)**: Metrics of the time taken to type a two-character sequence, measured from the keypress of the first character to the key release of the second character [39]. Also known as Digraph Latency.
- **Number of Key Events per Digraph (Mean, Standard Deviation)**: Metrics of the number of key events involved in producing each digraph. For this feature, a key event refers to one key press or one key release [39].
- **Trigraph Duration (Mean, Standard Deviation)**: Metrics of the time taken to type a three-character sequence, measured from the keypress of the first character to the key release of the third character [39]. Also known as Trigraph Latency.
- **Number of Key Events per Trigraph (Mean, Standard Deviation)**: Metrics of the number of key events involved in producing each trigraph. For this feature, a key event refers to one key press or one key release [39].

Novel Features

Many of the novel features in this category are informed by patterns observed in the Kaggle leaderboard or existing literature but are extended with additional statistical descriptors to capture more nuanced writing behaviors.

- **Number of times the keystrokes overlap**. Counts the instances where inter-key latencies are calculated as less than 0 milliseconds, indicating overlapping keystrokes.
- **Hold Latency (Minimum, Skewness)**: Metrics of the time from a single key press to its release.
- **Inter-key Latency Within Words (Sum, Maximum, Minimum, Kurtosis, Mean, Standard Deviation, Quantile, Skewness)**: Metrics of the time from one key release to the next key press, calculated only for consecutive keystrokes occurring within a word, excluding transitions involving spaces and word boundaries.
- **Intra-Word Press Latency (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**: Metrics of the time from one key press to the next key press within a

word, excluding transitions involving spaces and word boundaries. Also known as Inter-keystroke Interval (IKI) Within Words in other literatures.

- **Inter-word Interval (IWI) (Sum, Maximum, Minimum, Standard Deviation, Kurtosis, Quantile, Skewness)**: Metrics of the time between the release of the last key event of one word and the keypress of the first character in the next word.

- **Word-End Press Latency (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**: Metrics of the time between consecutive keystrokes that span word boundaries, specifically from the keypress of the last character in one word to the keypress of the subsequent space. Also known as Inter-keystroke Interval (IKI) Between Words in other literatures.

- **Time Between Words (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**: Metrics of the time from the keypress of the last character in one word to the keypress of the first character in the next word.

- **Inter-sentence Interval (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantile, Skewness)**: Metrics of the time from the key release of the final punctuation in one sentence to the keypress of the first character in the next sentence.

- **Inter-keystroke Interval (IKI) Between Sentences (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantile, Skewness)**: Metrics of the time between consecutive keystrokes that span sentence boundaries, specifically from the keypress of the punctuation mark to the keypress of the subsequent space.

- **Time Between Sentences (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**: Metrics of the time from the keypress of the final punctuation in one sentence (considered the sentence end) to the keypress of the first character in the next sentence.

- **Inter-paragraph Interval (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantile, Skewness)**: Metrics of the time spanning paragraph boundaries, measured from the key release of the final punctuation in one paragraph to the keypress of the first character in the next paragraph.

- **Inter-keystroke Interval (IKI) Between Paragraphs (Sum, Maximum, Minimum, Mean, Standard Deviation, Quantile)**: Metrics of the time between consecutive keystrokes at the start of paragraph boundaries, specifically from the keypress of the final punctuation in one paragraph to the keypress of the ENTER key initiating a new paragraph.

- **Time Between Paragraphs (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantile, Skewness)**: Metrics of the time from the keypress of the final punctuation in one paragraph to the keypress of the first character in the next paragraph.

- **Digraph Duration (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**: Metrics of the time taken to type a two-character sequence, measured from the keypress of the first character to the key release of the second character. Also known as Digraph Latency.

- **Number of Key Events per Digraph (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**: Metrics of the number of key events involved in producing each digraph. For this feature, a key event refers to one key press or one key release.

- **Trigraph Duration (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**: Metrics of the time taken to type a three-character sequence, measured from the keypress of the first character to the key release of the third character. Also known as Trigraph Latency.

- **Number of Key Events per Trigraph (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**: Metrics of the number of key events involved in producing each trigraph. For this feature, a key event refers to one key press or one key release.

3.2 Features Related to Revisions

This category includes 75 features related to revision behavior, capturing deletions, edits, and their associated timing, location, and frequency. Some features measure the duration of revision actions, while others capture when and where revisions occur within the evolving text. A distinction is made between revisions at the leading edge and those elsewhere in the document. In this project, the leading edge is defined as any position within two characters of the current end of the text at any given point in time. Features also distinguish between small-scale and large-scale revisions based on the number of characters involved. Additional features describe navigation-related behavior through cursor movement patterns, while ratio-based features quantify revision activity relative to overall writing output.

Kaggle-Leaderboard Features

- **Number of Times Each Cursor Position is Used**. Frequency of cursor positions during Input events. A total of 6 frequency bins were extracted, including the number of cursor positions used exactly once, twice, three times, four times, five times, and more than five times.
- **Cursor Positions Observed (Maximum, Mean, Skewness, Kurtosis)**. Metrics of the cursor position values during Input events.
- **Number of Deletions**. Counts the number of Remove or Cut events that delete characters [3, 18].
- **Deletion-to-Insertion Ratio**. The ratio of the number of deletions to the number of insertions, where insertions are counted from Input or Paste events that add characters [53]. Also known as DIRatio in literature [53].

Existing Literature Features

- **Time in Deletions (Mean, Standard Deviation)**. Metrics of the time spent on deletions, defined as Remove or Cut events that delete characters [23].
- **Global Number of Deletions per Sentence**. The ratio of the total number of deletions to the total number of sentences in the final text [38].
- **Local Number of Deletions per Sentence (Mean, Standard Deviation)**. Metrics of the number of deletions made per sentence during the writing process, representing the local deleting behavior within individual sentences [38].
- **Number of Edits**. Counts the number of Remove/Cut, Paste, Replace, and Move events [38].
- **Time in Edits (Sum)**. The total time spent on editing, defined as Remove/Cut, Paste, Replace, and Move events [28, 38].
- **Local Number of Edits per Sentence (Mean, Standard Deviation)**. Metrics of the number of edits made per sentence during the writing process, reflecting the local editing behavior within individual sentences [38].
- **Global Number of Edits per Sentence**. The ratio of the total number of edits to the total number of sentences in the final text [38].

- **Number of Leading-Edge Revisions**. Counts the number of revisions that occur at the leading edge [9, 18]. Also referred to as precontextual revisions in [9].

- **Number of In-Text Revisions**. Counts the number of revisions that occur away from the leading edge, within the body of the text [9, 18]. Also referred to as contextual revisions in [9].

- **Leading-Edge Ratio**. The ratio of the number of characters typed at the leading edge to the total number of keystrokes [9, 18].

- **Number of Small-Scale Revisions**. Counts the number of revisions that involve a single-character deletion, representing minor text corrections [42, 53].

- **Number of Large-Scale Revisions**. Counts the number of revisions involving multiple characters deleted in one action or consecutive deletion actions. These represent more significant removals [42, 53].

- **Time in Small-Scale Revisions (Mean, Standard Deviation)**. Metrics of the time spent on small-scale revisions, where each revision involves the deletion of a single character [18, 23].

- **Time in Large-Scale Revisions (Mean, Standard Deviation)**. Metrics of the time spent on large-scale revisions, which include deletions of multiple characters or consecutive deletion events [18, 23].

- **Text-Modification Ratio**. The ratio of the number of characters in the final text to the total number of keystrokes [18, 28].

- **Efficiency Ratio**. The ratio of the number of characters in the final text to the total number of characters typed as Input events [19].

Novel Features

- **Time in Deletions (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**. Metrics of the time spent on deletions, defined as Remove or Cut events that delete characters.

- **Time in Edits (Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantile, Skewness)**. Metrics of the time spent on edits, defined as Remove/Cut, Paste, Replace, and Move events.

- **Leading-Edge Typing Ratio**. The ratio of the number of characters typed at the leading edge to the total number of characters typed as Input events.

- **Time in Small-Scale Revisions (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**. Metrics of the time spent on small-scale revisions, where each revision involves the deletion of a single character.

- **Time in Large-Scale Revisions (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness)**. Metrics of the time spent on large-scale revisions, which include deletions of multiple characters or consecutive deletion events.

3.3 Features Related to Verbosity

This category includes 43 features related to verbosity, defined in terms of keystroke activity during the writing session. These features capture the volume of input produced, its variation over time, and its distribution across structural units such as words, sentences, and paragraphs. In addition to total keystroke counts, several features describe the variability of keystroke frequency using fixed time windows. Specifically, 30-second and 60-second intervals were used to segment the writing session, allowing temporal patterns in writing effort to be analysed. This includes pacing and consistency, and the presence of lulls in activity. Another set of features tracks lexical growth by measuring the time taken to reach specific word count milestones.

Kaggle-Leaderboard Features

- **Number of Keystrokes.** Total number of keystrokes recorded during the writing process [3, 14, 18, 49, 53].

- **Number of Keystrokes per Second.** Average number of keystrokes recorded per second throughout the session, as measured in [53], and adapted from [49], which originally measured keystrokes per minute [49, 53].

- **Time to Reach Words.** Time taken (in milliseconds) to reach specific word milestones (100, 200, 300, 400, and 500 words) during the writing process, providing insight into the writer's pace and progress at different stages.

- **Number of Words Observed (Maximum, Mean, Kurtosis, Quantile).** Metrics measuring the number of words observed throughout the writing process.

- **Number of Sentences Observed (Maximum, Mean, Kurtosis, Quantile).** Metrics measuring the number of sentences observed throughout the writing process.

- **Number of Paragraphs Observed (Maximum, Mean, Kurtosis, Quantile).** Metrics measuring the number of paragraphs observed throughout the writing process.

Existing Literature Features

- **Number of Alphabetical Keystrokes.** Counts the number of keystrokes corresponding to alphabetical characters (letters only), excluding spaces, punctuation, and other non-alphabetical characters [19].

- **Number of Characters Typed.** Counts the total number of characters typed as Input events, including both alphabetical and non-alphabetical characters [53].

- **Number of Insertions.** Counts the number of Input or Paste events that add characters [53].

- **Local Number of Keystrokes per Sentence (Mean, Standard Deviation).** Metrics measuring the number of keystrokes recorded per sentence during the writing process [38].

- **Number of Keystrokes per Time Window (Mean, Maximum, Minimum, Standard Deviation, Quantile).** Metrics of the number of keystrokes recorded within each fixed time window, representing the density of writing activity over time [3, 18].

- **Slope of the Number of Keystrokes per Time Window.** Overall trend in keystroke frequency across time windows, calculated using linear regression. A positive slope indicates increasing activity, while a negative slope suggests a decline in writing pace [3, 18].

- **Entropy of the Number of Keystrokes per Time Window.** Normalized Shannon entropy of keystroke counts across time windows, indicating irregularity in writing intensity [3, 18].

- **Uniformity of the Number of Keystrokes per Time Window.** Divergence between a uniform distribution and the actual keystroke distribution, where lower values indicate more even pacing [3, 18].

- **Local Extreme Number of Keystrokes per Time Window.** Number of directional changes in keystroke activity between consecutive time windows, capturing writing bursts or lulls [3, 18].

- **Mean Distance Between Time Windows of More Than One Keystroke.** Average number of time windows between consecutive active windows, defined as those with more than one keystroke. This reflects the spacing of sustained writing periods [3, 18].

- **Standard Deviation of Distance Between Time Windows of More Than One Keystroke.** Variation in spacing between active time windows, used to assess the regularity of writing bursts or pauses [3, 18].

Novel Features

Similar to the previous category, many of the novel features in this category are informed by patterns observed in the Kaggle leaderboard or existing literature but are extended with additional statistical descriptors to capture more nuanced writing behaviors.

- **Local Number of Keystrokes per Word (Mean, Standard Deviation).** Metrics measuring the number of keystrokes recorded per word during the writing process, reflecting word-level typing effort and consistency.

- **Global Number of Keystrokes per Word.** Ratio of total number of keystrokes to the number of words in the final text, indicating overall word-level writing effort.

- **Global Number of Keystrokes per Sentence.** Ratio of total number of keystrokes to the number of sentences in the final text, indicating overall sentence-level writing effort.

- **Local Number of Keystrokes per Paragraph (Mean, Standard Deviation).** Metrics of the number of keystrokes recorded per paragraph during the writing process, reflecting paragraph-level typing effort and consistency.

- **Global Number of Keystrokes per Paragraph.** Ratio of total number of keystrokes to the number of paragraphs in the final text, indicating overall paragraph-level writing effort.

3.4 Features Related to Fluency

This category includes 622 features related to writing fluency, which refers to the smoothness and continuity of text production during composition. In keystroke analysis, fluency is often operationalized through bursts, defined as stretches of uninterrupted typing segmented by pauses or revisions. This project uses a two-second pause threshold to describe burst boundaries, following conventions in many literature [7, 18, 19, 28, 32, 42]

Previous AES studies have typically relied on broad categories such as P-bursts (pause-initiated) or R-bursts (revision-initiated). In contrast, this project adopts a more fine-grained classification framework proposed by [7], identifying 11 distinct burst types based on how each burst begins and ends. While this framework has been applied in cognitive writing research, it has not yet been explored in the context of automated essay scoring. Definitions for each burst type are provided in Table 3.1.

Burst Type	**Definition**
PP	Bursts initiated after and terminated in a pause of at least 2 seconds.
RP1	Bursts initiated after a revision and terminated in a pause of at least 2 seconds, involving the production of new language only.
RP2	Bursts initiated after a revision and terminated in a pause of at least 2 seconds, involving the replacement of previously written text as well as the production of new language.
RP3	Bursts initiated after a revision and terminated in a pause of at least 2 seconds, involving only the replacement of previously written text.
PRL	Bursts initiated after a pause of at least 2 seconds and terminated by a revision at the leading edge.
PRI	Bursts initiated after a pause of at least 2 seconds and terminated by a revision insertion — that is, a revision carried out elsewhere in the text either within the same sentence or elsewhere in the text
PRLI	Bursts initiated after a pause of at least 2 seconds and terminated by both a revision at the leading edge and a revision carried out elsewhere in the text
RRL	Bursts initiated after a revision and terminated by a revision at the leading edge.
RRI	Bursts initiated after a revision and terminated by a revision insertion — that is, a revision carried out elsewhere in the text either within the same sentence or elsewhere in the text
RRLI	Bursts initiated after a revision and terminated by both a revision at the leading edge and a revision carried out elsewhere.
IB	Insertion bursts initiated in the middle of sentence production or after sentence completion but involving a revision carried out over sentence boundaries — that is, elsewhere in the text and no longer than a sentence in length

Table 3.1: Definitions of burst types based on initiation and termination characteristics.

For each burst type, features were extracted to capture their frequency, duration, text output (in characters and words), typing speed, and proportional contribution to the overall writing session. A separate set of aggregate features summarizes these metrics across all burst types. Additional features also measure the total time spent on lexical and non-lexical production, offering further insight into the writer's fluency and rhythm.

Existing Literature Features

- **Time for Non-lexical Production (Sum):** Total time spent producing only punctuations and spaces during the writing process [38].
- **Time for Lexical Production (Sum):** Total time spent producing actual lexical material (letters only) as Input events, excluding spaces and punctuation [38].

Novel Features

For each burst type, the following features were extracted:

- **Number of Bursts:** Total number of bursts during the writing process, as inspired by [18, 48].
- **Time in Burst (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantiles, Skewness):** Metrics of the time spent in each burst, as inspired by [42].
- **Number of Characters per Burst (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantiles, Skewness):** Metrics describing the number of characters produced in each burst, as inspired by [18, 48].
- **Number of Words per Burst (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantiles, Skewness):** Metrics of the number of words produced in each burst, as inspired by [42].
- **Typing Speed within Burst (Sum, Maximum, Minimum, Mean, Standard Deviation, Kurtosis, Quantiles, Skewness):** Metrics of the number of characters typed per second within each burst.
- **Proportion of Bursts:** Ratio of the number of bursts of this type to the total number of bursts.

- **Ratio of Characters in this Burst Type to Total Characters Typed:** Ratio of characters produced in this burst type to the total number of characters typed.
- **Ratio of Characters in this Burst Type to Total Keystrokes:** Ratio of characters produced in this burst type to the total number of keystrokes.
- **Ratio of Characters in this Burst Type to Final Text Characters:** Ratio of characters produced in this burst type to the total number of characters in the final text.

For all bursts (irrespective of burst types):

- **Number of All Bursts:** Total number of bursts for all types of bursts for each session.
- **Time in All Bursts (Maximum, Minimum, Mean, Standard Deviation, Quantile):** Metrics of the duration of all bursts, regardless of burst type.
- **Number of Characters per Burst (Maximum, Minimum, Mean, Standard Deviation, Quantile):** Metrics of the number of characters produced in all bursts.
- **Number of Words per Burst (Maximum, Minimum, Mean, Standard Deviation, Quantile):** Metrics of the number of words produced in all bursts.
- **Typing Speed within All Bursts (Maximum, Minimum, Mean, Standard Deviation, Quantile):** Metrics of the number of characters typed per second across all bursts.

3.5 Features Related to Other Events

This category includes 104 features that capture user interactions not directly tied to character input, such as paste actions, cursor movements, or mouse usage. Many of these features focus on actions that disrupt the natural flow of typing.

In this project, a disruption is defined as any Remove/Cut, Paste, or Jump event, where a Jump refers to a cursor movement of more than five characters. While similar concepts appear in prior studies [18, 23], no standard definition is widely adopted. The threshold used here is a practical approximation.

Kaggle-Leaderboard Features

Although not derived from keystroke data, these features reflect strategies used by top Kaggle submissions. External scoring models, particularly LightGBM predictors trained on other essay datasets, were reported to strongly correlate with the target scores. Their inclusion provides a competitive baseline and aligns this project with state-of-the-art approaches.

- **Predicted External Scores (LightGBM Model Predictions):** Scores generated by training LightGBM [37] models on anonymized essays from 8 external datasets [5, 6, 17, 24, 25, 35, 43, 43] comprising 24 essay types. The models predict essay scores based on word-level and character-level TF-IDF features extracted from these essays.
- **TF-IDF and SVD-Based Features for User Activity:** Numerical features generated by transforming event sequences, including user activities, key events, and inter-key latencies, into numerical representations using Term Frequency-Inverse Document Frequency (TF-IDF) [47] vectorization followed by dimensionality reduction through Singular Value Decomposition (SVD) [29], retaining a total of 64 components.

Existing Literature Features

- **Time in Disruptions (Mean, Standard Deviation):** Metrics of the time spent on disruptions, defined as Cut, Paste, or Jump events during the writing process [18, 23].

Novel Features

- **Number of Disruptions:** Counts the total number of Remove/Cut, Paste, or Jump events.
- **Time in Disruptions (Sum, Maximum, Minimum, Kurtosis, Quantile, Skewness):** Metrics of the time spent on disruptions, defined as Cut, Paste, or Jump events during the writing process.
- **Global Number of Disruptions per Sentence:** Ratio of the total number of disruptions to the total number of sentences in the final text.
- **Local Number of Disruptions per Sentence (Mean, Standard Deviation):** Metrics of the number of disruptions recorded per sentence during the writing process, representing the local disruption behavior within individual sentences.

3.6 Features Related to Final Product

This category includes 117 features derived from the final written product, reconstructed from the keystroke log. Although these features are not based on process-level data, they were commonly used in top-performing Kaggle submissions and serve as a strong baseline for comparison. Including them allows the project to benchmark against state-of-the-art models and assess how well keystroke-based features perform relative to traditional product-level metrics. Their inclusion in this project helps highlight the added value of keystroke-based process features beyond surface-level output.

Kaggle-Leaderboard Features

- **Number of Characters:** Counts the total number of characters in the final reconstructed text.
- **Number of Words:** Counts the total number of words in the final reconstructed text [18, 49].
- **Number of Sentences:** Counts the total number of sentences in the final reconstructed text.
- **Number of Paragraphs:** Counts the total number of paragraphs in the final reconstructed text.
- **Word Length (Mean, Maximum, Minimum, Standard Deviation, Quantile):** Metrics of the number of characters per word in the final reconstructed text [49].
- **Sentence Length (Mean, Maximum, Minimum, Standard Deviation, Quantile):** Metrics of the number of words per sentence in the final reconstructed text [38].
- **Paragraph Length (Mean, Maximum, Minimum, Standard Deviation, Quantile):** Metrics of the number of words per paragraph in the final reconstructed text.
- **Number of Orthographic Mistakes:** Counts the number of orthographic errors, specifically punctuation and capitalization mistakes, in the final reconstructed text.
- **Number of Misplaced Spaces Around Periods:** Counts the instances of misplaced spaces around periods in the final reconstructed text, including extra spaces before periods, missing spaces after periods, and improper spacing before or after periods.
- **Number of Misplaced Spaces Around Commas:** Counts the instances of misplaced spaces around commas in the final reconstructed text, including extra spaces before commas, missing spaces after commas, and improper spacing before or after commas.

- **Number of Incorrect Capitalization:** Counts the number of capitalization errors in the final text (e.g., incorrect use of capital letters at the start of sentences).

- **Number of Words by Length:** Counts the number of words in the final text that fall within various length categories: <5, ≥5, ≥6, ≥7, ≥8, ≥9, ≥10, ≥11, ≥12 characters.

- **Number of Sentences by Length:** Counts the number of sentences in the final text within various sentence-length categories: <50, ≥50, ≥60, ≥75, ≥100 words.

- **TF-IDF Features with SVD:** Numerical features extracted from the final reconstructed text using Term Frequency-Inverse Document Frequency (TF-IDF) [47] followed by dimensionality reduction through Singular Value Decomposition (SVD) [29], retaining a total of 64 components.

Chapter 4

Design

This chapter outlines the design of the experiment in this research, which takes the form of a supervised machine learning pipeline, as shown in Figure 4.1. It provides a comprehensive overview of the key considerations and decisions that shaped the project design. The chapter is divided into 10 sections, each addressing a critical component of the design, including dataset description, data cleaning, feature engineering and extraction, feature reduction, feature scaling, model training, machine learning algorithms, model evaluation, model interpretability, and ablation studies.

4.1 Dataset Description

The project dataset includes approximately 8.4 million keystroke logs collected from 2,471 participants, each of whom completed a timed, prompt-based argumentative writing task [36]. The prompts were adapted from four retired Scholastic Assessment Test (SAT) prompts [16], and participants were asked to compose an essay within a 30-minute window.

As participants wrote, their interactions were recorded in real time with precise timestamps. These logs, stored in the `train_logs.csv` file, capture each writing event alongside metadata such as keypress timing, activity category, cursor position, word count, and the resulting text change. Table 4.1 provides a sample of this data, illustrating the structure and granularity available for modeling the writing process. A complete breakdown of all recorded fields is provided in Appendix A.1.

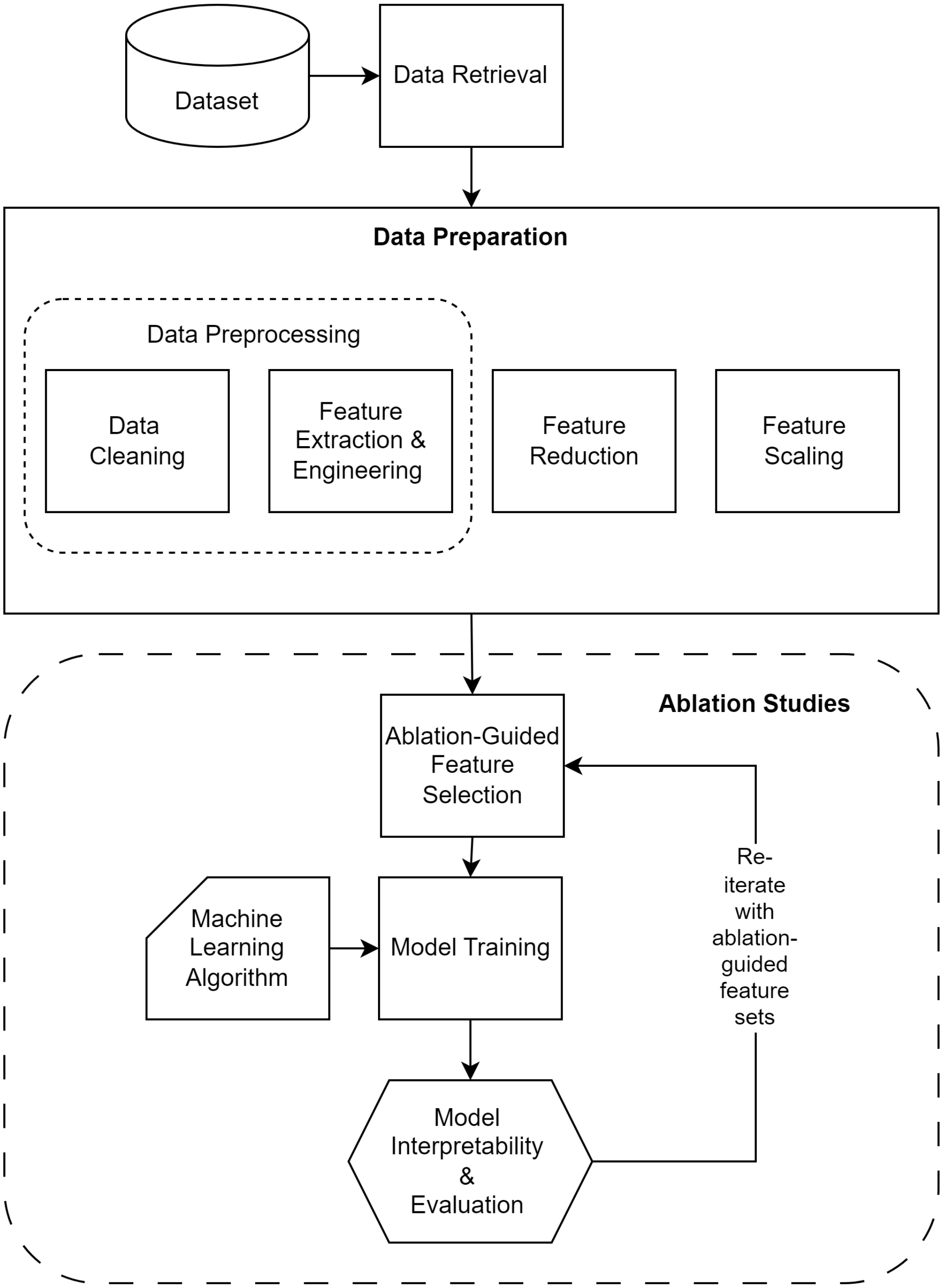
Dataset
Data Retrieval
Data Preparation
Data Preprocessing
Data Cleaning
Feature Extraction & Engineering
Feature Reduction
Feature Scaling
Ablation Studies
Ablation-Guided Feature Selection
Re-iterate with ablation-guided feature sets
Machine Learning Algorithm
Model Training
Model Interpretability & Evaluation

Figure 4.1: Overview of the experiment pipeline.

id	**event_id**	**action_time**	**activity**	**text_change**	**cursor_position**
0c97760e	1962	78	Input	q	1552
0c97760e	1963	117	Nonproduction	NoChange	1574
0c97760e	1964	42	Remove/Cut	.	1573
0c97760e	1965	29	Input		1574
0c97760e	1966	1764	Nonproduction	NoChange	1121
0c97760e	1967	2163	Move	qqqqqqq.	1576

Table 4.1: Simplified sample of keystroke logging data from `train_logs.csv`. Each row represents an event with associated timing, activity type, cursor position, and text change.

Once each essay was completed, it was scored by trained human raters on a scale from 0.0 to 6.0, with scores assigned in half-point intervals. This resulted in a continuous target variable suitable for supervised learning.

To support clearer interpretation of writing quality, this project mapped Kaggle essay scores to CEFR (Common European Framework of Reference for Languages) proficiency levels [20], as shown in Table 4.2. Following conventions from prior work [50], each CEFR level is split into two sub-levels (e.g., B1.i, B1.ii), and a score of 0.0 is left unmapped to indicate a below-threshold submission.

Kaggle Score	**CEFR Level**
0.0	–
0.5	A1.i
1.0	A1.ii
1.5	A2.i
2.0	A2.ii
2.5	B1.i
3.0	B1.ii
3.5	B2.i
4.0	B2.ii
4.5	C1.i
5.0	C1.ii
5.5	C2.i
6.0	C2.ii

Table 4.2: Heuristic mapping of Kaggle essay scores to CEFR proficiency levels.

Although the Kaggle dataset does not indicate whether participants are native or non-native English speakers, the CEFR was originally developed for assessing second language proficiency. However, it is widely recognized as a flexible, non-prescriptive framework that supports consistent descriptions of language ability across diverse contexts. Aligning essay scores with CEFR levels helps position this dataset within broader discussions of writing proficiency and enables comparisons with established international benchmarks.

The dataset was selected for its combination of detailed keystroke logging, human-scored writing outcomes, and, importantly, the availability of top-performing public leaderboard models. These baselines include keystroke-derived features and offer a strong point of comparison, helping to position this project within the current state of the art.

4.2 Data Cleaning

The data cleaning process was partially adapted from the first-place solution on the Kaggle leaderboard [34], providing a reliable baseline while preserving the temporal detail necessary for downstream feature engineering.

The following steps were applied:

- Removed events logged more than 10 minutes before the first input.
- Reset timestamps to start at zero and increase consistently.
- Fixed character encoding issues to ensure the logs could be parsed correctly.
- Label-encoded essay IDs to simplify merging with feature data.

Unlike the original solution, this project retains long pauses and event durations to preserve meaningful variation in writing behavior. Although not a novel contribution, this preprocessing step is essential to ensure a clean and consistent input for feature extraction and modeling.

4.3 Feature Engineering and Extraction

All features described in Chapter 3 were extracted.

4.4 Feature Reduction

Following feature extraction, the feature set contained 1,226 engineered features derived from keystroke logs. Although this high-dimensional representation was designed to capture a broad range of writing behaviors, it also introduced common challenges associated with large feature spaces. These included an increased risk of overfitting, model instability due to multicollinearity, and reduced interpretability. Previous research has shown that irrelevant or redundant features can negatively affect model performance and generalization [31]. To mitigate these risks, a two-stage feature reduction strategy was designed.

1. Low-Information Feature Filtering

This stage was designed to remove features that contribute little to no variation across essay sessions. The following steps were applied:

- Remove non-feature columns (e.g., IDs, scores).
- Discard constant features with zero variance.
- Filter near-constant or low-variance features using a defined threshold.

Variance was computed using:

$$\mathrm{Var}_{\mathrm{sample}}(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

where x_i is the feature value for essay session i, $\bar{x}$ is the mean, and n is the total number of sessions.

2. Redundancy and Multicollinearity Reduction

To address feature redundancy and multicollinearity, this stage involved:

- Computing pairwise Pearson correlation coefficients between features to identify strong linear dependencies:

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

where x and y are feature columns.

- Grouping features that exceed a predefined correlation threshold with each other
- Ranking features within each group based on their Spearman correlation with essay scores:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

where d_i is the rank difference between the feature value and the essay score for session i, and n is the total number of sessions.

- Retaining a subset of features based on the strength of their Spearman correlation

Spearman correlation with essay scores was chosen over Pearson due to the bounded and ordinal nature of the scoring scale, as well as the slight skewness observed in the score distribution, as shown in Figure 4.2. This violates the normality assumptions required for Pearson. Although statistical significance could be assessed via p-values, the feature reduction process was primarily guided by effect size.

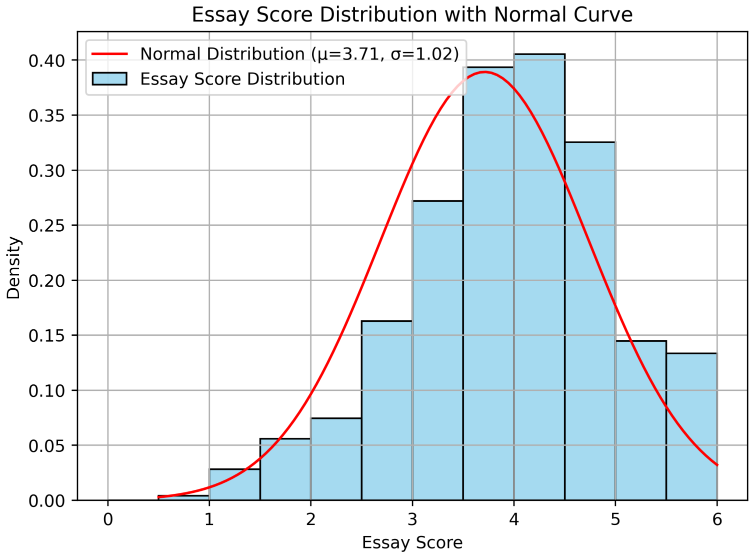

Figure 4.2: Essay score distribution with overlaid normal curve.

4.5 Feature Scaling

Given the wide variety of engineered features, ranging from frequency counts to timing intervals, a single scaling method was unlikely to be appropriate for all cases, as discussed in [22]. Instead, a dynamic approach was adopted, selecting the most suitable transformation based on the statistical properties of each feature. With that, a three-path decision rule was designed:

1. Quantile Uniform Scaling

Features with more than 50 unique values were assumed to be continuous and were transformed using a quantile-based method that mapped the empirical distribution to a uniform distribution over the range $[0, 1]$. This transformation is defined as:

$$x'_i = \frac{\text{rank}(x_i)}{n} \quad \text{so that} \quad x' \sim \text{Uniform}(0, 1)$$

where x was the input feature vector of size n, and $\text{rank}(x_i)$ denoted the rank position of value x_i in the sorted list of all values. This method spread values more evenly across the $[0, 1]$ range, reduces the impact of skewed distributions, and allows continuous features to be treated more consistently across the pipeline.

2. Robust Scaling with Clipping

To address features with extreme values in the training data, a robust scaling strategy was employed. This approach centered the distribution using the median and scaled it based on the interquartile range (IQR), which captured the spread between the 25th and 75th percentiles:

$$x'_i = \frac{x_i - \text{Median}(x)}{\text{IQR}(x)}, \quad \text{where} \quad \text{IQR}(x) = Q_{75}(x) - Q_{25}(x)$$

Unlike standard scaling, this method was less influenced by outliers and performed more reliably on heavy-tailed or skewed distributions. It preserved the central structure of the data while reducing the influence of extreme values.

After scaling, values were clipped to fall within a defined threshold:

$$x''_i = \min\left(\max\left(x'_i, -T\right), T\right)$$

where T was a configurable parameter defining the allowable range. This clipping step acted as a safeguard against residual outliers that might have remained after scaling, helping to improve training stability and generalization.

3. Direct Clipping

If extreme values were observed only in the test data, those values were clipped directly to the range observed in the training data:

$$x_i' = \min\left(\max\left(x_i, x_{\min}\right), x_{\max}\right)$$

This prevents the introduction of unexpected values during inference and eliminates the need for additional transformations.

4.6 Model Training

To guard against overfitting, this project's training strategy combined *k*-fold cross-validation with bagging. This two-level approach provided a more stable performance estimate and improved generalizability.

A visual summary of the training and prediction process is provided in Figure 4.3, illustrating how predictions across bags and folds are combined into the final OOF prediction vector used for evaluation.

The core of the setup was a 5-fold stratified cross-validation, which split the data into five subsets while preserving the distribution of essay scores across each split. In each fold, the model was trained on four folds and validated on the fifth. This process was repeated in six bagging rounds, each with a different random seed to introduce slight variations in how the data were split and how the model was initialized.

Using a different seed for each bag ensured that no two training runs used exactly the same data splits or model state. This helped reduce dependence on any single configuration of the data and increased the overall robustness of the evaluation. It also led to a broader range of learned representations, which were averaged to reduce variability across predictions.

In total, 30 models were trained: one for every combination of the 6 bagging rounds and 5 validation folds. Predictions were generated only when a sample appeared in the validation fold, which meant each essay was evaluated on data not seen during its training phase. These fold-level predictions were then averaged across all bags to produce the final out-of-fold (OOF) prediction for each sample.

4.7 Machine Learning Algorithm

4.7.1 Model Considerations

To identify the most suitable algorithm for predicting essay quality from keystroke-based features, a data-driven approach was used. The modeling phase began by replicating a strong

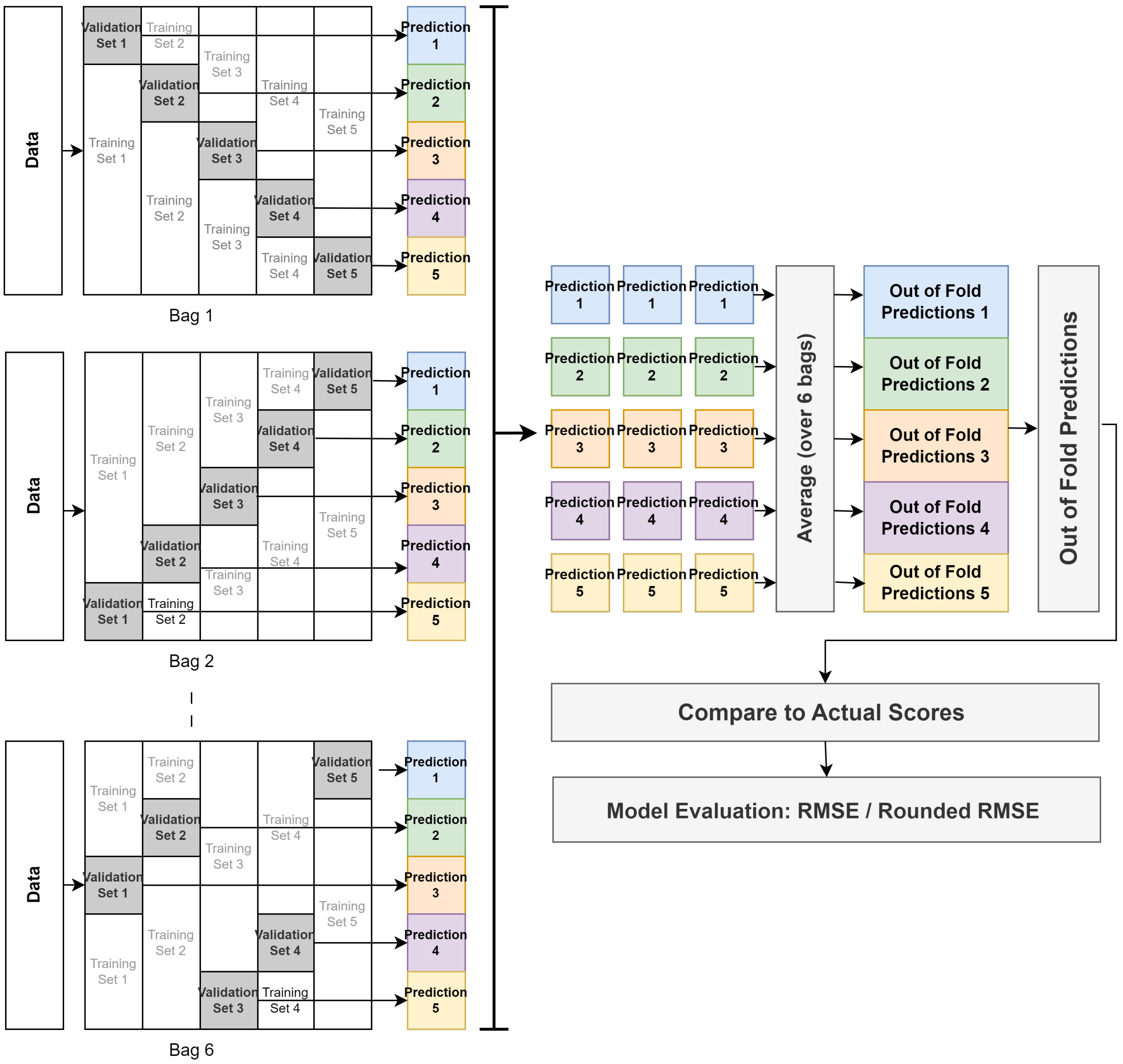

Figure 4.3: Overview of the cross-validation and bagging training pipeline.

baseline inspired by top-performing Kaggle solutions. This initial step helped confirm the integrity of the experimental setup and served as a foundation for further exploration.

Several algorithmic approaches were explored, including tree-based ensembles, deep neural networks, and transformer-based architectures designed for tabular data. Some were implemented as classifiers adapted for ordinal regression, reflecting the ordered nature of essay scores.

To ensure a fair comparison, each algorithm was evaluated using the same training setup described in 4.6. This comparison relied solely on baseline features, without incorporating newly engineered features at this stage, in order to isolate the effect of the learning algorithm itself.

Performance was assessed using OOF Root Mean Squared Error (RMSE). The results are summarized in Table 4.3.

Model	**OOF RMSE**
LightGBM Regressor [37]	0.5769
XGBoost Regressor [13]	0.5808
LightGBM Classifier [37]	0.5815
CatBoost Regressor [26]	0.5819
XGBoost Classifier [13]	0.5848
LightAutoML ResNet [46]	0.5867
LightAutoML DenseNet [46]	0.5934
Bagging Regressor [12]	0.5949
TabNet Regressor [4]	0.6029
LightAutoML FTTransformer [46]	0.6037

Table 4.3: Out-of-fold RMSE performance for each model evaluated during the baseline replication phase.

4.7.2 Final Model Selection: LightGBM Regressor

Based on the comparison, the LightGBM Regressor was selected as the final model. It demonstrated the best predictive performance across experiments and trained efficiently both in terms of time and computational resources without requiring a GPU. This made it especially suitable for the iterative experimentation in the ablation studies. Additionally, it includes built-in tools for computing feature importance, which aligned well with the project's need for interpretability.

Alternative strategies, such as ensembling different model types or applying automated stacking, were also considered. However, a single-model approach was ultimately preferred for its transparency and simplicity, particularly in support of ablation studies.

4.8 Model Evaluation

To evaluate the accuracy of the model's predictions, this project used Root Mean Squared Error (RMSE) as the primary performance metric, since the final model selected followed a regression approach. RMSE is widely used in automated essay scoring and writing analytics to measure how closely predicted scores align with human ratings [2, 18, 33, 48]. It reflects the average magnitude of prediction error, with greater sensitivity to large discrepancies. This makes it especially appropriate for our use case with continuous scoring scales in essay evaluation.

RMSE is calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

where:

- N is the number of essays,
- y_i is the actual human-assigned score, and
- $\hat{y}_i$ is the model's predicted score from OOF validation.

In addition to standard RMSE, a rounded RMSE was also computed. In this variant, each OOF prediction was rounded to the nearest 0.5 to match the scoring scale used by human raters:

$$\hat{y}_i^{(\text{rounded})} = \frac{\text{round}(\hat{y}_i \times 2)}{2}$$

The same RMSE formula was then applied to these rounded predictions.

By reporting both RMSE and rounded RMSE on OOF predictions, the evaluation was designed not only to capture the model's fine-grained accuracy, but also how well its outputs align with the discrete scoring scale used in human assessment.

4.9 Model Interpretability

Interpretability was a central consideration in the model design. Since this project aimed to identify meaningful features, it was important that the model not only achieved strong predictive performance but also offered insight into which features contributed to its decisions. This was designed to be particularly relevant in educational settings, where transparency supports trust, fairness, and the potential for actionable feedback.

Two complementary interpretability techniques were adopted: feature importance and SHAP (SHapley Additive Explanations) values.

1. Feature Importance

LightGBM provides two measures of feature importance:

- **Gain**: The total improvement in model performance when a feature is used in a split.

- **Split**: The number of times a feature is used to split data across all decision trees.

To produce stable feature rankings, both were computed across all folds and bagging rounds, then averaged. Gain highlighted features with strong predictive influence, while split revealed features the model relied on consistently [37].

2. SHAP Values

To complement global importance scores with more detailed, instance-level explanations, SHAP values were used. SHAP assigns each feature a contribution score for individual predictions, helping to explain how specific feature values influenced a given outcome. This is especially useful for capturing feature interactions and non-linear effects [41].

Because SHAP is computationally expensive, it was applied only to a single LightGBM model: the bag and fold combination with the lowest validation RMSE. This approach was designed to provide a representative view of model behavior without incurring the cost of running SHAP on all 30 models.

Since SHAP is model-agnostic, the same method can be applied to other model types or ensemble strategies in future work. This design choice ensures that the interpretability framework used in this project remains flexible and extensible beyond the current model configuration.

By integrating both global and local interpretability methods, the model design aimed to support a transparent assessment process and reinforce the project's broader goal of connecting keystroke patterns to writing quality in a meaningful and explainable way.

The results from feature importance and SHAP analysis were then used to guide the ablation studies.

4.10 Ablation Studies

To better understand the contribution of different types of features, ablation studies were conducted. These experiments systematically evaluated model performance when specific groups of features were removed, isolated, or incrementally introduced. The goal was to assess whether certain sources, categories, or individual features meaningfully improved predictive accuracy, and whether the model could remain performant with a reduced or alternative feature set. This analysis is part of the broader design-feedback loop, as shown in Figure 4.1, where findings from SHAP and feature importance guided the construction of ablation setups, and in turn, ablation findings informed subsequent feature refinement and model decisions.

All models in the ablation studies followed the same training setup in 4.6 with consistent hyperparameters across experiments. This controlled design ensured that any observed changes in performance could be confidently attributed to the feature subset used.

In this project, three sets of ablation experiments were performed:

1. Source-Based Ablation

To evaluate the added value of new contributions and validate the utility of features drawn from prior work, features were grouped based on their sources:

- **Kaggle-Leaderboard Features**: Replicated from top-performing models in the original competition.
- **Existing Literature Features**: Informed by prior research in keystroke logging and writing analytics.
- **Novel Features**: New features introduced in this project that capture patterns not previously used in AES models.

Models were trained using only one source group, pairwise combinations (e.g., Kaggle-Leaderboard + Novel), and the full set of features.

2. Category-Based Ablation

To provide insight into both the standalone predictive power of each category and its contribution within the broader feature set, features were also grouped by conceptual category, based on theoretical frameworks identified in literature. These categories reflect underlying cognitive and linguistic processes involved in writing:

- Pause Dynamics
- Revision
- Fluency

- Verbosity
- Product-Related Features
- Other Events

For each category, two experiments were conducted:

1. Model trained using only that feature group
2. Model trained using all features except that group (leave-one-out)

3. Individual Feature Ablation

To assess the marginal contribution of individual features, a fine-grained ablation was performed. Starting with the full set of Kaggle-Leaderboard Features, the top-ranked features from the Existing Literature and Novel groups were added one at a time.

Importantly, this process did not involve cumulative addition. Each experiment tested a single new feature in isolation, added to the baseline set, to enable a clear and interpretable comparison of its individual contribution.

Feature ranking was determined using gain scores from LightGBM models trained exclusively on the Existing Literature and Novel features during source-based ablation. Once again, these gain values were averaged across all folds and bagging rounds to produce a stable ranking. Only the highest-ranked features were selected for testing to reduce computational cost.

Chapter 5

Implementation

This chapter outlines the end-to-end technical implementation of the proposed system, detailing each stage of the machine learning pipeline. While the Design chapter described a conceptual top-down pipeline, the actual implementation reflects practical software development considerations. Specifically, modular code organization and utility function reuse. All experiments were conducted locally or in a Kaggle-based environment.

5.1 Environment and Dependencies

This project was carried out in a Jupyter Notebook using Python 3.12. Jupyter was chosen for its modular and interactive environment, allowing individual logic blocks, such as data cleaning, training and ablation, to be executed, tested, and debugged independently.

To support execution both locally and on Kaggle's GPU-accelerated environment, a boolean flag `KAGGLE` was defined at the top of the notebook. Depending on its value, the notebook dynamically configures directories for loading input data and saving output artifacts, ensuring seamless portability.

The project relied on a wide range of third-party Python libraries, organized below by functional category:

Data Handling and Numerical Computing

- **`numpy`**: For efficient numerical array operations and mathematical functions.
- **`pandas`**: Used extensively for tabular data manipulation, preprocessing, and merging feature sets.

- `scipy`: Provided statistical tools including regression (`linregress`), correlation (`pearsonr`, `spearmanr`), entropy calculation (`entropy`), and similarity measures such as Jensen-Shannon divergence (`jensenshannon`).

Natural Language Processing (NLP)

- `TfidfVectorizer` (from `sklearn`): Converted essay text into sparse TF-IDF vectors based on word frequency and informativeness.

- `TruncatedSVD` (from `sklearn`): Applied dimensionality reduction to TF-IDF vectors using Latent Semantic Analysis (LSA) for efficient modeling.

- `ftfy`: Normalized and cleaned Unicode anomalies often found in dataset.

Machine Learning and Model Evaluation

- `lightgbm`: A gradient boosting framework optimized for speed and performance on large tabular datasets. It served as the primary machine learning algorithm in this project.

- `scikit-learn (sklearn)`: A general-purpose machine learning library used for preprocessing, evaluation, and dimensionality reduction. Key components included:

  - `StratifiedKFold`: Performed cross-validation while maintaining balanced distributions of essay scores.

  - `Metrics`: Included `mean_squared_error` for regression accuracy, and `roc_auc_score`, `auc`, and `roc_curve` for exploratory classification evaluations.

Visualization and Interpretability

- `matplotlib.pyplot`: Used for generating a wide range of visualizations including scatter plots, essay score distributions, loss curves, feature importance plots, and beeswarm plots for SHAP analysis.

- `seaborn`: Used to simplify the creation of histograms and for plotting ranked feature importances as bar plots.

- `shap`: Enabled SHAP value visualizations to interpret model predictions and identify the most influential features.

Text Cleaning and Preprocessing

- **re**: Used for regular expression matching and cleaning textual input patterns in raw essay data.

Utilities and Workflow Management

- **os**, **shutil**, **glob**: Handled file system operations, such as directory access and path resolution.
- **time**, **copy**, **warnings**: Managed runtime behavior, object duplication, and suppressed unnecessary warnings during execution.
- **humanize**: Transformed raw durations and file sizes into readable formats for logging purposes.
- **tqdm.notebook**: Provided progress bars for long-running tasks such as training and data processing.
- **collections.OrderedDict**: Maintained ordered configuration dictionaries to ensure consistent experiment setup.

Serialization and Persistence

- **dill**: Used to serialize complex Python objects, including custom classes and trained models, for checkpointing and reproducibility.

5.2 Configuration Management

For ease of experimentation, all system-level settings were managed using a central configuration dictionary implemented as an `OrderedDict` named `cfg`. The parameters within `cfg` are grouped below according to their functional roles in the pipeline:

5.2.1 General Experiment Setup

- `cfg.experiment_name`: Defines the name of the experiment for saving outputs.
- `cfg.run_name`: Identifies the current execution run, used for tracking logs and results. This is set dynamically via `RUN_NAME`.
- `cfg.seed`: Sets the random seed for reproducibility. (Final value: `42`)

5.2.2 Model Training Setup

- `cfg.train`: Enables or disables the training stages. (Final values: `True`)
- `cfg.nfolds`: Specifies the number of cross-validation folds. (Final value: `5`)
- `cfg.nbags`: Determines the number of bagging rounds for model training. (Final value: `6`)
- `cfg.is_classification`: Toggles between regression and classification tasks. (Final value: `False`)
- `cfg.plot_top_features`: Sets the number of features to display in feature importance or SHAP plots. (Final value: `30`)
- `cfg.ablation_top_n_feature`: Sets the number of top ranked features to conduct individual feature ablation studies. (Final value: `70`)

5.2.3 Data Preparation Options

- `cfg.clean`: Indicates whether to clean the dataset. (Final value: `True`).
- `cfg.preprocess`: Enables or disables preprocessing of raw input data. (Final values: `True`)
- `cfg.preprocess_ext_train`: Enables the inclusion of external datasets in training. (Final values: `True`)
- `cfg.augment`: Indicates whether data augmentation is applied. (Initial value: `True` for external dataset, then set to `False` for the main dataset)
- `cfg.scale_feats`: Determines whether to apply feature scaling to numerical features. (Final value: `True`)

5.2.4 Target and Output Control

- `cfg.target_col`: Specifies the name of the target variable to predict. (Final value: `"score"`)
- `cfg.clip_value`: Defines the threshold for clipping numeric feature values during scaling. (Final value: `5.5`)

5.2.5 Time Window and Keystroke Parameters

- `cfg.first_margin`: Time buffer (in milliseconds) included before the first meaningful key event in a session. (Final value: `600000` ms = 10 minutes)

- `cfg.last_margin`: Time buffer (in milliseconds) after the last meaningful event in a session. (Final value: `np.inf`)

5.2.6 Feature Reduction Thresholds

- `cfg.corr_strict_threshold`: Removes perfectly correlated features (correlation ≥ threshold). (Final value: `1.0`)

- `cfg.corr_soft_threshold`: Removes highly correlated features, retaining only the top N by target relevance. (Final value: `0.999`)

- `cfg.feature_variance_thresh`: Eliminates features with very low variance. (Final value: `1e-5`, i.e., 0.00001)

- `cfg.near_constant_buffer`: Identifies near-constant features based on dominant value frequency. (Final value: `10`)

- `cfg.high_corr_feat_keep_top_n`: Retains the top N features from each correlated group. (Final value: `2`)

5.2.7 File Paths and Environment Routing

- `cfg.kaggle`: Indicates whether the code is running in a Kaggle environment or locally. This flag is set dynamically via `KAGGLE`.

- `cfg.data_dir`: Path to the raw input data directory, typically derived from the experiment name.

- `cfg.mydata_dir`: Path to the preprocessed or intermediate data used during training and evaluation.

- `cfg.out_dir`: Path to the output directory where models, logs, plots, and other results are saved.

5.3 Data Retrieval

Three CSV files were provided as part of the dataset:

- `train_logs.csv`: Keystroke logs for the training set, loaded into `train_log_df`.
- `test_logs.csv`: Keystroke logs for the test set, loaded into `test_log_df`.
- `train_scores.csv`: Target scores for the training essays, loaded into `train_scores_df`.

These files were loaded using `pandas.read_csv()`, with file paths managed through the `cfg.data_dir` configuration variable.

Two basic integrity checks were performed:

- File shapes and headers were inspected using `.shape` and `.head()` on each DataFrame.
- Missing values were identified using `.isnull().sum()`.

For local development and debugging, a fallback mechanism was included to create a small test set from `train_log_df` if the original test file contained fewer than 100 rows.

5.4 Data Cleaning

The `clean_data()` function prepares the raw keystroke logs through a sequence of standardized cleaning operations:

- `label_encode()`

  Uses `sklearn.preprocessing.LabelEncoder()` to convert the `id` column into a numeric `id_encode` for consistent session grouping.

- `remove_margin()`

  Removes keystrokes where `up_event` is `Unidentified`, then identifies the first and last meaningful keypresses in each session, where `activity` is not `Nonproduction` or `up_event` is `Shift` or `CapsLock`. It retains only events within a time window defined by `cfg.first_margin` and `cfg.last_margin`, applies trimming per session using `groupby('id_encode')`, and resets the `event_id`.

- `reset_timestamps()`

  Adjusts `down_time` and `up_time` so that each session begins at time 0, while preserving relative timing between events.

- `clean_str_columns()`

  Cleans non-standard characters in `up_event`, `down_event`, and `text_change` using the helper function `clean_latin()`, which internally relies on the `ftfy` library.

The full cleaning logic is encapsulated in the `clean_data()` function, shown in Listing 5.1.

```python
def clean_data(log_df):
    log_df, _ = label_encode(log_df, test_df=None, col="id", replace=False)
    log_df = remove_margin(log_df, cfg.first_margin, cfg.last_margin)
    log_df = reset_timestamps(log_df)
    log_df = clean_str_columns(log_df, ['up_event', 'down_event', 'text_change'])
    return log_df
```

Listing 5.1: clean data() function

It is applied to both `train_log_df` and `test_log_df`. The cleaned DataFrames are then passed into the feature extraction pipeline.

5.5 Feature Engineering & Extraction

Feature extraction was implemented as a modular pipeline designed to transform raw keystroke logs into the comprehensive set of 1,226 features described in the Chapter 3

5.5.1 Utility Functions

The pipeline was built from reusable utility functions, organized into four functional categories:

1. **Statistical Summary Functions**

   Functions that compute aggregated statistics such as mean, standard deviation, skewness, and kurtosis over subsets of the keystrokes.

   e.g. `compute_grouped_statistics()`

2. **Base-Level Functions**

   Functions that operate directly on raw keystroke logs to extract foundational units such as burst events and inter-paragraph boundaries. All functions in this category begin with one of the following prefixes: `get_`, `append_`, or `validate_`.

   e.g. `get_inter_paragraph_boundaries()`, `append_word_bursts()`

3. **Helper Functions**

   General-purpose utilities that support the feature extraction process through tasks such as encoding, reconstruction, time normalization, etc.

e.g. `onehot_encode()`, `split_to_sentence()`, `recon_writing()`

4. **Main Feature Functions**

   Prefixed with `add_`, these high-level functions implement standalone feature logic or coordinate multiple lower-level utilities to generate behaviorally meaningful representations. They combine statistical and structural signals into interpretable features and append new columns directly to the main feature matrices `train_df` and `test_df`.

   e.g. `add_location_based_revision_features()`, `add_local_sentence_level_features()`

Although the features are conceptually grouped into behavioral categories (Pause Dynamics, Verbosity, Revision, Fluency, Other Events, and Product-Related), the actual implementation prioritized computational efficiency over strict alignment with these groups. In practice, many features from different conceptual categories are extracted within the same function to reduce redundant iteration and reuse shared computations.

5.5.2 Orchestration and Execution

The overall pipeline is executed via two main orchestration functions:

- `extract_writing_features()`

  Applies the writing-related feature pipeline to the main dataset.

- `extract_external_dataset_features()`

  Incorporates AI-derived score features from an external TF-IDF-enhanced dataset. For each `essay_set` (24 total), model predictions are generated using LightGBM regressors trained on the external dataset. These predictions are appended to `train_df` and `test_df` as additional features. Model training and inference logic is encapsulated within `train_model()` and `infer_model()`, described in Section 5.8.

A simplified view of this orchestration is shown in Listing 5.2. The output from both steps is stored in `train_df` and `test_df`, which serve as the basis for subsequent feature reduction and model training steps.

```
# Writing-related features
train_df, train_log_df, preprocessors, preprocessors_event =
    extract_writing_features(
    train_log_df,
    score_df=train_scores_df,
    preprocessors=None,
    preprocessors_event=None
)

test_df, test_log_df, _, _ = extract_writing_features(
    test_log_df,
    score_df=None,
    preprocessors=preprocessors,
    preprocessors_event=preprocessors_event
)

# Score-based features from external dataset
train_df, test_df, preprocessors, preprocessors_event =
    extract_external_dataset_features(
    train_df,
    test_df,
    ext_df,
    preprocessors,
    cfg,
    params,
    ext_feats,
    preprocessors_event
)
```

Listing 5.2: Simplified view of feature extraction execution)

Although both `train_df` and `test_df` were generated for completeness, only `train_df` was used for model training and evaluation. Due to the structure of the competition dataset, the majority of the test set was not publicly released, and full labels were unavailable for local evaluation.

A detailed summary of all 1,226 extracted features is provided in the file `features_summary.csv`. This includes each feature's variable name, description, behavioral category, and the function used for its extraction. The file is included in the preprocessed dataset, and selected examples are shown in Appendix A.2.

5.6 Feature Reduction

Using modular utility functions, the two-stage feature reduction pipeline was implemented by the `reduce_features()` orchestration function and configuration options were passed into it via the `cfg` object.

1. Low-Information Feature Filtering

- `get_non_feature_columns()`

  Returns columns to be excluded from modeling, including IDs (`id`, `id_encode`), reconstructed essays (`reconstructed`), fold assignments, target scores (`score`, `score_group`, `score_*`), and auxiliary score rank indicators (e.g., `score_*_order`).

- `get_constant_features()`

  Returns feature names with zero variance across all samples.

- `get_near_constant_features()`

  Identifies features where a single value dominates. Sensitivity is configured via `cfg.near_constant_buffer`.

- `get_low_variance_features()`

  Returns features with very low variance. The threshold is defined by `cfg.feature_variance_thresh`.

2. Redundancy and Multicollinearity Reduction

- `get_feature_target_correlations()`

  Computes Spearman and Pearson correlations between each feature and the essay score for ranking purposes.

- `get_feature_feature_correlations()`

  Generates a pairwise correlation matrix using Pearson correlation (set via `method='pearson'`).

- `drop_correlated_features()`

  Returns features to be removed from highly correlated groups, retaining the most predictive ones based on their Spearman correlation with the target. The number of top features retained per group is controlled by `cfg.high_corr_feat_keep_top_n`. This step is applied twice:

  1. Using `cfg.corr_strict_threshold` to remove perfectly correlated features.
  2. Using `cfg.corr_soft_threshold` for high-correlation filtering.

The full pipeline is orchestrated by the `reduce_features()` function, which applies all filtering steps and returns both the final feature list and a breakdown of removed features, as shown in Listing 5.3.

```python
def reduce_features(df,
                    corr_threshold_strict=1.0,
                    corr_threshold_soft=0.999,
                    variance_thresh=1e-5,
                    near_constant_buffer=10,
                    save_path=f'{cfg.out_dir}{cfg.run_name}/',
                    verbose=True,
                    top_n_keep=2):
    ...
    return final_features, removed_features
```

Listing 5.3: reduce_features() function. Key arguments control variance and correlation thresholds; additional arguments (`save_path`, `verbose`) handle logging and output paths.

5.7 Feature Scaling

Feature scaling was implemented via the `scale_features()` utility, which applied transformation logic to each feature in `train_df` and `test_df`. This is how the three-path decision rule was implemented:

- **Quantile Uniform Scaling:** Applied when a feature had more than 50 unique values. The transformation was performed using `scale_col()` with `mode='quantile_uniform'`, which wraps `sklearn.preprocessing.QuantileTransformer` with `output_distribution='uniform'`.

- **Robust Scaling with Clipping:** Used when extreme values were detected outside the range defined by `cfg.clip_value`). Scaling was performed using `scale_col()` with `mode='robust'`, which internally uses `sklearn.preprocessing.RobustScaler`. After scaling, values were clipped to the range [−`cfg.clip_value`, +`cfg.clip_value`] to reduce the influence of outliers.

- **Direct Clipping:** If extreme values appeared only in the test set, values were directly clipped to the same range to ensure consistency with the training distribution, without applying any transformation.

As they were already scaled, TF-IDF features were excluded from this process. The fitted scalers were returned in an `encoders` dictionary for potential reuse during inference.

Although the final algorithm, LightGBM, does not rely on feature scaling due to its tree-based architecture, the scaling pipeline was retained for consistency with earlier experimental stages and to support compatibility with alternative model types, as outlined in Section 4.7.

Standardizing feature ranges also improved the interpretability of SHAP plots by allowing feature contributions to be visualized on a consistent scale.

The full pipeline is orchestrated by the `scale_features()` function, as shown in Listing 5.4.

```python
def scale_features(train_df, test_df, feats, clip_value=10):
    concat_test = len(test_df) >= 30
    encoders = {}
    for col in [f for f in feats if 'tfidf' not in f]:
        n_unique = len(train_df[col].unique())
        # Case 1: Quantile Uniform Scaling
        if n_unique > 50:
            train_df, test_df, encoder = scale_col(train_df, test_df, col, mode='quantile_uniform', ...)
            encoders[col] = encoder
        # Case 2: Robust Scaling + Clipping
        elif (train_df[col].min() < -clip_value) or (train_df[col].max() > clip_value):
            train_df, test_df, encoder = scale_col(train_df, test_df, col, mode='robust', ...)
            encoders[col] = encoder
            train_df[col] = np.clip(train_df[col], -clip_value, clip_value)
            test_df[col] = np.clip(test_df[col], -clip_value, clip_value)
        # Case 3: Direct Clipping (test only)
        elif (test_df[col].min() < -clip_value) or (test_df[col].max() > clip_value):
            test_df[col] = np.clip(test_df[col], -clip_value, clip_value)
    return train_df, test_df, encoders
```

Listing 5.4: scale_features() function

5.8 Model Training

Model training was orchestrated via the `train_model()` function, which unified cross-validation, bagging, data augmentation, model inference, model evaluation, and model interpretability into a single pipeline.

5.8.1 Bagging & Cross-Validation

- For each bagging iteration (`cfg.nbags`), a new seed was used to shuffle and reassign fold splits.

- Within each bag, Stratified K-Fold cross-validation (`cfg.nfolds`) was performed, with fold assignment done by (`add_fold()`).
- Models were trained per fold, and validation predictions were collected and averaged across folds and bags to generate Out-of-Fold (OOF) predictions.

5.8.2 Data Augmentation

- If `cfg.augment = True`, PCA-based data augmentation was applied using the `aug_array()` utility to synthetically expand the training set within each fold.

5.8.3 Model Training & Inference

- Models were trained using the `train_fn()` utility, with LightGBM (`mode='lgb'`) as the backend.
- Validation predictions were generated via `infer_fn()` and stored in `train_df` under a suffix-specific column.
- If `save_model = True`, each trained model was serialized using `pickle_dump()` and saved to disk.

5.8.4 Model Interpretability & Logging

- if `log_importance = True`, LightGBM feature importances are logged.
  - `log_feature_importance()` was called after each fold to log gain and split importances.
  - `log_average_feature_importance()` aggregated these into global importances at the end of training.
- if `log_shap = True`, SHAP values were calculated.
  - `log_feature_importance_shap()` was used to compute and log global and local SHAP values for the best-performing model.

5.8.5 Model Evaluation

- `val_metric`: Mean RMSE across all validation folds and bags.
- `oof_metric`: RMSE computed on the final Out-of-Fold predictions.

- oof_metric_round: RMSE computed after rounding predictions to the nearest 0.5 to reflect the scoring rubric.
- If show_fig = True, visualizations were generated using plot_pred().

A simplified overview of the train_model() orchestration is provided in Listing 5.5.

```
def train_model(...):
    Initialize prediction column and metrics list
    Create model output directory

    for bag in range(cfg.nbags):
        Assign folds using Stratified K-Fold (with new seed)

        for fold in range(cfg.nfolds):
            Split train/validation data
            If cfg.augment is True:
                Apply PCA-based augmentation via aug_array()

            Train LightGBM model using train_fn()
            Predict on validation set using infer_fn()

            If log_importance is True:
                Log feature importance using log_feature_importance()

            Save model to disk (if enabled)
            Store predictions for this fold

    If log_shap is True:
        Compute SHAP values using log_feature_importance_shap()

    If log_importance is True:
        Aggregate global importance using log_average_feature_importance()

    Compute mean validation RMSE across folds/bags
    Average predictions across all bags

    Compute OOF predictions and RMSE and rounded RMSE)
    Plot performance graph (if enabled)

    return train_df, val_metric
```

Listing 5.5: Simplified view of train$_model()inpseudocode$

5.8.6 Hyperparameter Configuration

The final model was trained using LightGBM with manually tuned hyperparameters in Table 5.1, selected based on cross-validation performance to balance regularization, generalization, and efficiency. All runs used a fixed random seed (`cfg.seed`) for reproducibility.

Parameter	**Value**	**Description**
`boosting`	`gbdt`	Gradient Boosted Decision Trees as the boosting method.
`objective`	`regression`	Specifies the task as regression.
`metric`	`rmse`	Root Mean Squared Error used for evaluation.
`lambda_l1`	0.0	L1 regularization term.
`lambda_l2`	0.0	L2 regularization term.
`num_leaves`	16	Maximum number of leaves in one tree.
`learning_rate`	0.01	Step size shrinkage used in each boosting iteration.
`max_depth`	4	Maximum depth of the tree.
`feature_fraction`	0.4	Fraction of features to consider at each iteration.
`bagging_fraction`	0.4	Fraction of data to be randomly sampled for each iteration.
`bagging_freq`	8	Frequency of bagging. A value of 8 means bagging is performed every 8 iterations.
`extra_trees`	`True`	Enables use of extremely randomized trees.
`min_data_in_leaf`	5	Minimum number of data points in one leaf.
`random_state`	`cfg.seed`	Random seed for reproducibility.

Table 5.1: LightGBM Hyperparameter Configuration

5.9 Ablation Studies

The implementation of the ablation studies was handled using a set of modular utilities. The overall workflow consisted of three stages: metadata alignment, ablation set generation, and experiment execution.

1. Metadata Alignment

Before grouping features, the metadata file `features_summary.csv`, which contains the source and category information for each feature, was aligned with the final list of reduced features using `sync_features_with_metadata()`. This ensured that only retained features were considered when generating ablation groups and prevented mismatches caused by name formatting or prior filtering.

2. Ablation Set Generation

The utility `generate_ablation_sets()` was used to construct dictionaries mapping different feature groupings to their corresponding feature lists. These groupings were created in three formats:

- **Source-based sets**: `baseline`, `existing`, `novel`, `all`, and their pairwise combinations.
- **Category-based sets**: e.g., `fluency`, `revision`, `pause_dynamics`, as well as `minus_` variants.
- **Individual feature add-back sets**: Top-ranked features from the `existing_novel` group were added one by one to the `baseline` set. These were selected based on LightGBM gain importance.

Each of these was defined in dictionary format and used directly in the training loop.

3. Experiment Execution

All experiments were executed using `run_ablation_experiments()`, which takes a dictionary of feature sets and loops through them, training separate models for each subset.
For each experiment:

- The model was trained using `train_model()` with a consistent configurations in Table 5.1.
- RMSE and rounded RMSE were computed.
- All model outputs were logged to a structured directory under the main output path and summary tables were saved to a `.csv` file and used for later analysis.

```python
# Run source-based ablation
run_ablation_experiments(..., feature_sets=source_ablation_sets, ablation_type="source")

# Run category-based ablation
run_ablation_experiments(..., feature_sets=category_ablation_sets, ablation_type="category")
```

```

# Run individual feature ablation
run_ablation_experiments(..., feature_sets=individual_ablation_sets,
    ablation_type="individual")
```

Listing 5.6: Orchestration of ablation experiments

5.10 Reproducibility

To support reproducible experimentation and avoid redundant computation, key outputs from each stage of the pipeline were saved to disk. This allowed the workflow to be resumed from any point and made it easier to reuse processed data, trained models, and evaluation results across multiple runs.

Two custom utility functions were used for serialization:

- `pickle_dump(obj, path)`
  Saves a Python object to the specified path using `dill` serialization, overwriting the file if it already exists. It uses protocol 4 to ensure compatibility with large objects.

- `pickle_load(path)`
  Restores a previously saved Python object from a `dill`-serialized file. Raises an error if the file is missing or improperly formatted.

These utilities were used throughout the project to store:

- **Processed Keystroke Logs**: Preprocessed versions of `train_log_df` and `test_log_df`.

- **Engineered Features**: Both `train_df` and `test_df` after feature extraction.

- **Preprocessors and Encoders**: Any fitted transformation objects (e.g., scalers, token encoders).

- **Trained Models**: All LightGBM models trained per fold and bag under a consistent naming scheme based on model type, fold, and bag index.

- **Evaluation Outputs**: Feature importance rankings, SHAP values, and ablation study results.

- **Plots**: Visualizations such as SHAP beewarm plots and prediction diagnostics.

Chapter 6

Results and Evaluation

This chapter presents the key findings from the experimental phase of the project, examining how engineered keystroke features relate to writing quality and affect the performance of automated scoring models.

It opens with an overview of the final feature set after the reduction process. This is followed by an analysis of feature importance using several interpretability techniques, including LightGBM's gain and split metrics, as well as SHAP values. Next, a series of ablation studies assess the individual and group-level contributions of specific feature subsets to prediction accuracy.

The chapter concludes with a summary of key insights and a discussion of limitations and future directions, including the generalisability of the features and their practical relevance for real-world applications.

6.1 Final Feature Set

A total of 30 features were removed during feature reduction, reducing the feature set from 1,226 to 1,196. A closer inspection of the removed features revealed a diverse distribution across both feature sources and categories, as shown in Figure 6.1. Notably, none of the product-related features were removed, indicating their relative uniqueness and high potential value in capturing final writing quality.

With that, this distribution concludes that redundancy and low information value were not confined to any one source or category.

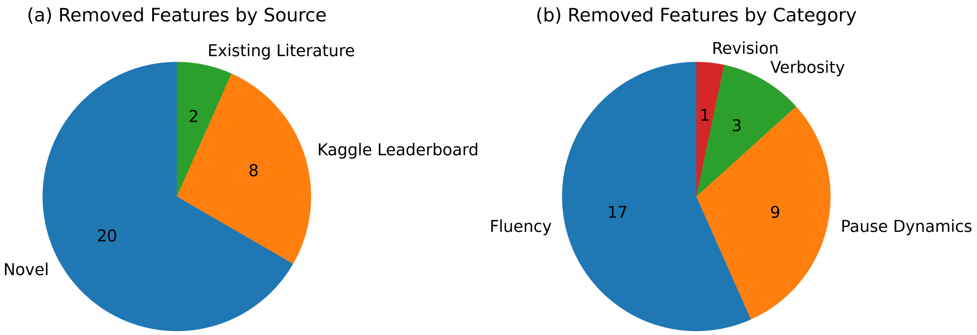

Figure 6.1: Distribution of removed features by (a) source and (b) conceptual category.

6.2 Feature Importance

The top 30 features ranked by LightGBM gain, LightGBM split count, and SHAP values are presented in Appendix A.1, Appendix A.2, and Appendix A.3, respectively. The gain and split scores are averaged across all bags and folds, while the SHAP values are derived from the best performing model.

Among the top-ranked features across the LightGBM gain and SHAP values, nine were newly engineered features introduced in this project:

1. `alphabetical_keystroke_count`: Number of alphabetical keystrokes
2. `character_input_count`: Number of characters typed
3. `digraph_num_events_sum`: Total number of key events per digraph
4. `RRLI_burst_char_count_sum`: Total number of characters across all RRI bursts
5. `RRLI_burst_duration_q50`: Median duration of RRLI bursts
6. `RRLI_burst_word_count_sum`: Total number of words across all RRLI bursts
7. `RRI_burst_char_count_sum`: Total number of characters across all RRI bursts
8. `trigraph_num_events_sum`: Total number of key events per trigraph
9. `insertion_count`: Number of insertions

While these features ranked highly in terms of gain and SHAP values, they did not feature prominently in the split-based importance ranking. This suggests that although they were not frequently used to split decision trees during training, they provided high information gain when selected, contributing substantial predictive value in fewer, more impactful instances.

Although features developed in this project (both novel and literature-based) made up the majority of the feature set, they were not disproportionately represented among the top-ranked features. Of the top 30 most important features based on LightGBM Gain and SHAP values, only 9 were sourced from this project: 6 of which were novel and 3 adapted from existing literature. This is notable considering that the novel group alone accounts for approximately 69% of the full feature set, while literature-based features comprise just 6%. The presence of several project-developed features among the top ranks indicates that the feature engineering and adaptation efforts were successful in identifying meaningful features in predicting essay scores. However, the relatively small share of high-ranking features from such a large pool also suggests that further filtering or selection could enhance the overall signal-to-noise ratio of the feature set.

As shown in the SHAP beeswarm plot, several of these features, particularly those related to verbosity and fluency, offer further insights into how they influenced predictions:

- `alphabetical_keystroke_count` tended to increase predicted scores when its values were high, with red SHAP points concentrated on the right. Conversely, lower values were associated with decreased predictions, particularly in responses with minimal lexical keystrokes.

- `character_input_count` followed a similar pattern. A higher number of typed characters was generally linked to increased predictions, while very low input exerted a stronger negative effect.

- `RRI_burst_char_count_sum` and `RRLI_burst_word_count_sum` also showed positive effects. These features represent the total size of RRI and RRLI bursts in terms of characters and words, respectively. Larger bursts were generally linked to higher predicted scores.

- `RRLI_burst_duration_q50`, by contrast, was negatively associated with predicted scores. Longer durations within RRLI bursts tended to lower predictions, potentially reflecting struggle, hesitation, or inefficiency. Shorter RRLI bursts tended to correspond with higher predicted scores, possibly indicating more fluent or confident revision.

6.3 Ablation Studies

6.3.1 Source-Based Ablation

Table 6.1 presents the RMSE and rounded RMSE for each source-based configuration.

Ablation Group	**RMSE**	**Rounded RMSE**
Kaggle	0.5791	0.5957
Kaggle + Literature	0.5802	0.5962
Kaggle + Novel	0.5859	0.6054
All	0.5863	0.6025
Literature + Novel	0.6420	0.6598
Novel	0.6441	0.6640
Literature	0.6593	0.6733

Table 6.1: Source-Based Ablation Results.

The Kaggle leaderboard feature set achieved the lowest error overall (RMSE = 0.5791, Rounded RMSE = 0.5957), establishing a strong performance baseline.

Adding the full set of features from either existing literature or those developed in this project resulted in a slight increase in error. However, the fully combined model, which incorporates features from all sources, remained competitive. This suggest that these additional features may still provide complementary value.

In contrast, models trained exclusively on the literature-based or novel feature sets performed significantly worse than the Kaggle-only baseline. This highlights the strength and efficiency of the leaderboard features. Still, the relatively small performance gap in the combined model suggests that project-developed features can enhance predictions when paired with a strong core.

The size of each feature group is also notable. The novel set includes 843 features. This is nearly three times the size of the Kaggle set of 305 features. Yet novel feature set did not outperform it when used in isolation. This implies that while the novel features contain valuable signals, their predictive efficiency per feature may be lower. Nevertheless, the solid performance of the combined model indicates that both novel and literature-based features likely capture behavioral patterns that complement the baseline.

These findings motivated a more detailed analysis by behavioral category and individual features, as presented in the following sections.

6.3.2 Category-Based Ablation

Table 6.2 presents the performance of models trained using only a single feature category, while Table 6.3 shows the impact on performance when each category was removed from the full feature set.

Feature Category (Only)	RMSE	Rounded RMSE
Product-Related	0.5855	0.5996
Other Events	0.6328	0.6472
Verbosity	0.6572	0.6724
Pause Dynamics	0.6670	0.6805
Fluency	0.6674	0.6888
Revision	0.6783	0.6912

Table 6.2: Performance using individual feature categories in isolation. Lower RMSE values indicate better predictive performance.

Feature Category Removed	RMSE	Rounded RMSE
Minus Fluency	0.5837	0.6001
Minus Revision	0.5853	0.6029
Minus Pause Dynamics	0.5868	0.6073
Minus Verbosity	0.5877	0.6074
Minus Other Events	0.5884	0.6078
Minus Product-Related	0.6088	0.6273
All Features Included	0.5857	0.6059

Table 6.3: Performance after removing feature categories from the full model. Higher RMSE indicates greater impact when the feature group is excluded.

Models trained using only product-related features performed best among the individual categories, with an RMSE of 0.5855 and a rounded RMSE of 0.5996. This suggests that final product features capture essential aspects of writing quality, even without access to process-based data. In contrast, categories such as pause dynamics, fluency, and revision performed noticeably worse in isolation, highlighting their dependence on interaction with other behavioral features.

The exclusion-based results further reinforce the value of product-related features. When this category was removed from the full model, performance declined more than for any other group (RMSE = 0.6088), underscoring its strong standalone contribution. Interestingly, removing fluency and revision features did not lead to a meaningful increase in error. This suggests that while these categories may carry useful signals, their contributions are either partially redundant or less essential in the presence of stronger feature types.

Taken together, these findings indicate that all behavioral categories provide complementary value, but product-based features offer particularly robust and self-contained predictive strength.

6.3.3 Individual Feature Ablation

Table 6.4 presents the results of an ablation study in which individual features were added to the Kaggle baseline model to assess their standalone contribution. A total of 70 features—selected from the novel and literature-based groups based on their LightGBM gain ranking—were tested one at a time.

Feature Added	**RMSE**	**Rounded RMSE**	**Source**	**Category**
PRL_burst_word_count_sum	0.5782	0.5963	Novel	Fluency
RRLI_burst_duration_q50	0.5784	0.5941	Novel	Fluency
trigraph_duration_q75	0.5784	0.5956	Novel	Pause Dynamics
lexical_production_time	0.5787	0.5954	Existing Literature	Fluency
RRLI_burst_typing_speed_q95	0.5789	0.5951	Novel	Fluency
digraph_duration_mean	0.5789	0.5965	Existing Literature	Pause Dynamics
PP_burst_typing_speed_mean	0.5789	0.5945	Novel	Fluency
all_burst_typing_speed_q95	0.5789	0.5970	Novel	Fluency
RP3_burst_typing_speed_sum	0.5789	0.5953	Novel	Fluency
RRI_burst_typing_speed_q75	0.5789	0.5970	Novel	Fluency
digraph_num_events_sum	0.5789	0.5963	Novel	Pause Dynamics
character_input_count	0.5789	0.5957	Existing Literature	Verbosity
keystroke_count_per_60s_q75	0.5789	0.5937	Existing Literature	Verbosity
PRL_burst_typing_speed_sum	0.5789	0.5980	Novel	Fluency

Table 6.4: Top-performing individual features that improved the baseline model. Full results for all 70 tested features are included in the Appendix A.4.

Of these, 14 features resulted in improved performance, yielding RMSE values lower than the baseline (RMSE = 0.5791). A closer examination of the 14 features that outperformed the baseline reveals that 10 were novel features developed in this project, while 4 originated from existing literature. Most of these features fall within the fluency and pause dynamics categories, with two from verbosity. High-performing fluency features included several burst-based typing speed and volume metrics such as `RRLI_burst_typing_speed_q95`, `PP_burst_typing_speed_mean`, and `PRL_burst_word_count_sum`, which reflect productive, fast-paced bursts of writing. Pause-related features like `digraph_duration_mean` and `trigraph_duration_q75` captured lower-level temporal rhythms that also contributed positively to score prediction.

Verbosity-related features such as `character_input_count` and `keystroke_count_per_60s_q75` suggest that overall typing volume is another useful signal of writing performance, even when considered in isolation from other process-based features.

These results support earlier findings from the feature importance analysis, where many of these same features ranked highly. While not all engineered features enhanced performance on their own, this subset demonstrates that carefully designed features grounded in writing theory can provide measurable standalone value.

6.4 Summary of Findings

Across the feature importance rankings and individual ablation results, several features emerged as consistently strong predictors of essay quality. In particular, three features performed reliably across both evaluation methods: `character_input_count`, `digraph_num_events_sum`, and `RRLI_burst_duration_q50`. These features consistently ranked highly across SHAP and LightGBM gain metrics, and individually enhanced predictive performance when added to the baseline model.

- `RRLI_burst_duration_q50` (**RMSE = 0.5784**): This feature captures the median duration of revision bursts that begin after a revision and are terminated by both a leading-edge and non-leading-edge revision. Its consistent negative SHAP contribution and standalone ablation impact suggest that shorter durations in these revision episodes may reflect more fluent or decisive editing behavior. This aligns with cognitive models of writing that associate revision fluency with higher proficiency.

- `character_input_count` (**RMSE = 0.5789**): This feature captures the total number of characters typed. In the SHAP beeswarm plot, higher values were associated with increased predicted scores. This aligns with findings from previous research [53], which found that the number of insertions made by students during a persuasive writing task significantly varied across different score groups, with a higher number of insertions associated with better performance (F(3, 757) = 143.80).

- `digraph_num_events_sum` (**RMSE = 0.5789**): This feature measures the total number of keystroke events that make up digraph units, offering insight into the fluidity of lower-level motor processes in writing. Its high feature importance and positive ablation performance suggest that a higher density of digraph-level typing may reflect smoother typing rhythm and lexical access.

Together, these features provide robust evidence for the predictive utility of engineered keystroke features. While many other features showed promise in one evaluation, these three were unique in demonstrating value across both importance and ablation perspectives. However, it is worth noting that although they improved performance over the baseline, none outperformed the product-related features when used in isolation, hence highlighting the continued dominance of final-text features in scoring accuracy.

6.5 Limitations and Future Work

This study provides promising evidence for the role of keystroke features in automated language assessment, but it also has several limitations that point toward future directions.

One limitation of this study lies in the narrow scope of the dataset. All keystroke logs were collected from a single writing task: a persuasive essay written in response to a SAT-style prompt. This uniformity means that the engineered features, particularly those capturing burst structures, fluency dynamics, and revision timing, were optimised for a specific genre and context. As a result, their generalisability to other writing modes, such as narrative, descriptive, or argumentative essays, remains uncertain. Future work should evaluate the transferability of these features across a broader set of prompts, genres, and learner profiles to assess their robustness and domain independence.

While product-based features remain strong standalone predictors of writing quality, this study demonstrates that keystroke-derived features offer complementary insights not visible in final text alone. The aim was not to replace product-based metrics, but to contribute behavioural indicators that enhance scoring models when used in tandem. Future research should explore more integrated architectures such as multimodal or hierarchical models that can jointly leverage product and process signals for more interpretable and accurate assessment.

Due to the constraints of the Kaggle competition, final model evaluation relied on out-of-fold cross-validation, without access to a held-out test set. While this offers a reliable estimate of generalisation within the dataset, future work should validate the models on external datasets or via cross-task generalisation to assess performance in real-world settings.

Several practical and methodological considerations also highlight avenues for future improvement:

- **Limited Error Analysis:** The current evaluation focused on RMSE-based metrics, without analysing which specific types of essays or scoring errors were most problematic. Future work could include qualitative or cluster-based error analysis to identify underperforming subgroups, such as low-scoring essays with high predictions, and refine features accordingly.

- **Computational Cost of Feature Engineering:** Some of the more complex features, particularly those involving RRLI and RRI burst structures, required extensive preprocessing and multiple stages of temporal segmentation. While effective, these approaches may be difficult to scale in real-time or classroom settings. Future research could investigate lightweight proxies for these features or develop end-to-end systems where feature extraction is embedded in the learning architecture.

- **Lack of Real-Time Evaluation:** This study focused exclusively on retrospective scoring. However, many educational tools today are used in real-time, providing feedback during the writing process. Future research should investigate whether partial keystroke streams (e.g., the first 5 minutes of typing) can be used to generate meaningful predictions or adaptive feedback mid-task.

Collectively, these limitations suggest several directions for extending the current work. While this project serves as a proof of concept, future research should focus on validating the proposed features across different contexts and populations, and on integrating them into more interactive, scalable, and pedagogically relevant applications.

Chapter 7

Legal, Social, Ethical and Professional Issues

7.1 Legal and Professional Considerations

This project complies with the British Computer Society (BCS) *Code of Conduct* and *Code of Good Practice*. Key principles applied include respect for public interest (by safeguarding data privacy), acting with integrity and competence (through transparent reporting and reproducibility), fulfilling duties to relevant authorities (by adhering to Kaggle's data usage terms), and contributing responsibly to the profession.

All data used was drawn from the publicly available Kaggle Feedback Prize dataset, which is released for academic research and teaching purposes. The dataset is anonymised, and no personally identifiable information was processed. Baseline models replicated during this project were also publicly shared on the competition platform and reused in accordance with open-source licenses.

Although no new user data was collected, this study acknowledges the sensitive nature of keystroke logging. In real-world applications, similar methods must comply with data protection regulations such as the UK GDPR, including secure data handling, transparency, and obtaining informed consent from participants.

7.2 Ethical, Social, and Environmental Implications

This research explores how keystroke data can enhance automated language assessment. While product-based features are well-established, the inclusion of behavioral signals from writing processes may allow for a more comprehensive and fair assessment of student writing.

Ethically, the project aims to support transparency and fairness in scoring. It does not propose replacing human evaluators but instead seeks to complement existing feedback mechanisms with insights into the writing process. Any future application should ensure accessibility and inclusivity across diverse learner populations.

From an environmental perspective, the project focused on lightweight machine learning models rather than resource-intensive deep learning architectures. This contributes to a more sustainable approach to educational AI research and development.

Overall, the project was conducted with professional responsibility and with consideration for its broader implications on education, technology trust, and responsible deployment.

Chapter 8

Conclusion

This project set out to identify novel patterns in keystroke behavior that could serve as meaningful features for automated language assessment. By designing and extracting a wide range of process-oriented features, the study aimed to capture aspects of the writing process that are not visible in the final text alone.

The findings demonstrate that several of these engineered features, particularly those related to character input, typing fluency, and revision timing, contribute positively to essay score prediction. A small number of features, such as `character_input_count`, `digraph_num_events_sum`, and `RRLI_burst_duration_q50`, consistently ranked highly in both importance metrics and ablation studies. These results confirm the value of keystroke-derived features as complementary indicators of writing quality.

While the analysis was based on a single prompt and retrospective scoring setup, the work establishes a strong foundation for future exploration. Next steps may include extending the feature set across different genres and proficiency levels, embedding process features into multimodal assessment models, and applying them in real-time feedback systems to support writers during composition.

In summary, this study provides promising evidence that novel keystroke features can enhance automated scoring systems, offering deeper insights into writing behavior and paving the way for more responsive and interpretable writing analytics tools.

References

[1] Alejandro Acien, Aythami Morales, John V. Monaco, Ruben Vera-Rodriguez, and Julian Fierrez. Typenet: Deep learning keystroke biometrics. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 4(1):57–70, Jan 2022.

[2] Dimitrios Alikaniotis, Helen Yannakoudakis, and Marek Rei. Automatic text scoring using neural networks. *CoRR*, abs/1606.04289, 2016.

[3] Laura K. Allen, Matthew E. Jacovina, Mihai Dascalu, Rod D. Roscoe, Kevin M. Kent, Aaron D. Likens, and Danielle S. McNamara. ENTERing the time series SPACE: Uncovering the writing process through keystroke analyses. In *Proceedings of the International Conference on Educational Data Mining (EDM)*, pages 1–8, Raleigh, NC, June 2016. International Educational Data Mining Society. Accessed: 2024-03-07.

[4] Sercan Ö. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6679–6687, May 2021.

[5] ASAP. Asap automated essay scoring. Accessed: 2025-03-10.

[6] ASAP. Asap scored essay. Accessed: 2025-03-10.

[7] Veerle M. Baaijen, David Galbraith, and Kees de Glopper. Keystroke analysis: Reflections on procedures and measures. *Written Communication*, 29(3):246–277, 2012.

[8] Ritwik Banerjee, Song Feng, Jun Seok Kang, and Yejin Choi. Keystroke patterns as prosody in digital writings: A case study with deceptive reviews and essays. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1469–1473, Doha, Qatar, October 2014. Association for Computational Linguistics.

[9] KHALED BARKAOUI. What and when second-language learners revise when responding

to timed writing tasks on the computer: The roles of task type, second language proficiency, and keyboarding skills. *The Modern Language Journal*, 100(1):320–340, 2016.

[10] Anna L. Barnett and Nichola Stuart. Understanding typing skill in students with developmental disorders. *Current Developmental Disorders Reports*, 11(2):63–74, Jun 2024.

[11] Robert Bixler and Sidney D'Mello. Detecting boredom and engagement during writing with keystroke analysis, task appraisals, and stable traits. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces*, IUI '13, page 225–234, New York, NY, USA, 2013. Association for Computing Machinery.

[12] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, Aug 1996.

[13] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[14] Ikkyu Choi and Paul Deane and. Evaluating writing process features in an adult efl writing assessment context: A keystroke logging study. *Language Assessment Quarterly*, 18(2):107–132, 2021.

[15] Evgeny Chukharev-Hudilainen. *Chapter 5 Empowering Automated Writing Evaluation with Keystroke Logging*, pages 125 – 142. Brill, Leiden, The Netherlands, 2019.

[16] College Board. What's on the sat, 2024. Accessed: 2024-02-20.

[17] CommonLit. Commonlit evaluate student summaries. Accessed: 2025-03-10.

[18] Rianne Conijn, Christine Cook, Menno van Zaanen, and Luuk Van Waes. Early prediction of writing quality using keystroke logging. *International Journal of Artificial Intelligence in Education*, 32(4):835–866, Dec 2022.

[19] Rianne Conijn, Jens Roeser, and Menno van Zaanen. Understanding the keystroke log: the effect of writing task on keystroke features. *Reading and Writing*, 32(9):2353–2374, Nov 2019.

[20] Council of Europe. Common european framework of reference for languages: Level descriptions, 2024. Accessed: 2024-02-20.

[21] Ronan Cummins and Marek Rei. Neural multi-task learning in automated assessment, 2018.

[22] Lucas B.V. de Amorim, George D.C. Cavalcanti, and Rafael M.O. Cruz. The choice of scaling technique matters for classification performance. *Applied Soft Computing*, 133:109924, January 2023.

[23] Paul Deane. Using writing process and product features to assess writing quality and explore how those features relate to other literacy tasks. *ETS Research Report Series*, 2014(1):1–23, 2014.

[24] Deepachalapathi. Essay grade v1. Accessed: 2025-03-10.

[25] Deepachalapathi. Essay grade v2. Accessed: 2025-03-10.

[26] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support, 2018.

[27] Noura Farra, Swapna Somasundaran, and Jill Burstein. Scoring persuasive essays using opinions and their targets. In Joel Tetreault, Jill Burstein, and Claudia Leacock, editors, *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 64–74, Denver, Colorado, June 2015. Association for Computational Linguistics.

[28] David Galbraith and Veerle M. Baaijen. *Chapter 13 Aligning Keystrokes with Cognitive Processes in Writing*, pages 306 – 325. Brill, Leiden, The Netherlands, 2019.

[29] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420, Apr 1970.

[30] Hongwen Guo, Paul D. Deane, Peter W. van Rijn, Mo Zhang, and Randy E. Bennett. Modeling basic writing processes from keystroke logs. *Journal of Educational Measurement*, 55(2):194–216, 2018.

[31] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3(null):1157–1182, March 2003.

[32] John Hayes. From idea to text. *The Sage handbook of writing development*, pages 65–79, 01 2009.

[33] Xinyun He, Qi Shu, Mo Zhang, Wei Huang, Han Zhao, and Mengxiao Zhu. Beyond final products: Multi-dimensional essay scoring using keystroke logs and deep learning. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, LAK '25, page 601–610, New York, NY, USA, 2025. Association for Computing Machinery.

[34] Tomoo Inubushi. 1st place solution - linking writing processes to writing quality. `https://www.kaggle.com/competitions/linking-writing-processes-to-writing-quality/discussion/466873`, 2024. Accessed 2 April 2025.

[35] Kaggle. Feedback prize: English language learning. Accessed: 2025-03-10.

[36] Kaggle. Linking writing processes to writing quality, 2024. Accessed: 2024-03-30.

[37] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, Red Hook, NY, USA, 2017. Curran Associates Inc.

[38] Elma Kerz, Fabio Pruneri, Daniel Wiechmann, Yu Qiao, and Marcus Ströbel. Understanding the dynamics of second language writing through keystroke logging and complexity contours. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 182–188, Marseille, France, May 2020. European Language Resources Association.

[39] Agata Kołakowska and Agnieszka Landowska. Keystroke dynamics patterns while writing positive and negative opinions. *Sensors*, 21(17):5963, September 2021.

[40] Mariëlle Leijten and Luuk Van Waes. Keystroke logging in writing research: Using inputlog to analyze and visualize writing processes. *Written Communication*, 30(3):358–392, 2013.

[41] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *CoRR*, abs/1705.07874, 2017.

[42] Donia Malekian, James Bailey, Gregor Kennedy, Paula de Barba, and Sadia Nawaz. Characterising students' writing processes using temporal keystroke analysis. In *Proceedings of the International Conference on Educational Data Mining (EDM)*, pages 1–6, Montreal, Canada, July 2019. International Educational Data Mining Society.

[43] Mazlumi. Ielts writing scored essays. Accessed: 2025-03-10.

[44] Atsushi Mizumoto and Masaki Eguchi. Exploring the potential of using an ai language model for automated essay scoring. *Research Methods in Applied Linguistics*, 2(2):100050, 2023.

[45] Erin Pacquetet. *The Effect of Linguistic Properties on Typing Behaviors and Production Processes*. PhD thesis, State University of New York at Buffalo, 2024. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2024-08-28.

[46] Alexander Ryzhkov, Anton Vakhrushev, Dmitry Simakov, Rinchin Damdinov, Vasilii Bunakov, Alexander Kirilin, and Pavel Shvets. Lightautoml. `https://github.com/sb-ai-lab/LightAutoML`. Accessed: 2025-04-01.

[47] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[48] Sandip Sinharay, Mo Zhang, and Paul Deane and. Prediction of essay scores from writing process and product features using data mining methods. *Applied Measurement in Education*, 32(2):116–137, 2019.

[49] Mobina Talebinamvar and Forooq Zarrabi. Clustering students' writing behaviors using keystroke logging: a learning analytic approach in efl writing. *Language Testing in Asia*, 12(1):6, Feb 2022.

[50] Georgios Velentzas, Andrew Caines, Rita Borgo, Erin Pacquetet, Clive Hamilton, Taylor Arnold, Diane Nicholls, Paula Buttery, Thomas Gaillat, Nicolas Ballier, and Helen Yannakoudakis. Logging keystrokes in writing by English learners. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 10725–10746, Torino, Italia, May 2024. ELRA and ICCL.

[51] Ruosong Yang, Jiannong Cao, Zhiyuan Wen, Youzheng Wu, and Xiaodong He. Enhancing automated essay scoring performance via fine-tuning pre-trained language models with combination of regression and ranking. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1560–1569, Online, November 2020. Association for Computational Linguistics.

[52] Mo Zhang, Jiangang Hao, Chen Li, and Paul Deane. Classification of writing patterns using keystroke logs. In L. Andries van der Ark, Daniel M. Bolt, Wen-Chung Wang, Jeffrey A. Douglas, and Marie Wiberg, editors, *Quantitative Psychology Research*, pages 299–314, Cham, 2016. Springer International Publishing.

[53] Mengxiao Zhu, Mo Zhang, and Paul Deane. Analysis of keystroke sequences in writing logs. *ETS Research Report Series*, 2019(1):1–16, 2019.