

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ

ПЕРЕДБАЧЕННЯ ЧАСУ ЧИТАННЯ

RESEARCH AND SOFTWARE IMPLEMENTATION OF READING TIME

PREDICTION METHODS

Виконав: студент(ка) 2 курсу, групи КНТ-213м

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

ЗУЄВ Б.В.

(прізвище та ініціали)

Керівник СЕРДЮК С.М.

(прізвище та ініціали)

Рецензент ПОЛЯКОВ М.О.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЗУЄВА Богдана Віталійовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів передбачення часу читання. Research and Software Implementation of Reading Time Prediction Methods

керівник проєкту (роботи) к.т.н., доцент, СЕРДЮК Сергій Микитович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 7 ” жовтня 2024 року № 398

2. Строк подання студентом проєкту (роботи) 1 грудня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.

2. Матеріали і методи. 3. Основні рішення щодо реалізації системи.

4. Експлуатація, тестування та експериментальне дослідження програми.

5. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СЕРДЮК С.М., доцент		
Нормоконтролер	КАМІНСЬКА Ж.К. асистент		

7. Дата видачі завдання “ 7 ” вересня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2 тиждень	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	3-5 тижні	Розділ 2
4	Розробка програми	6-7 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	8-9 тижні	Розділ 4
6	Оформлення пояснювальної записки та документів до неї.	10 тиждень	Додатки
7	Нормоконтроль та рецензування.	11 тиждень	
8	Захист роботи.	12 тиждень	

Студент(ка)

_____ Богдан ЗУЄВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Сергій СЕРДЮК
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра 76 с., 8 табл., 8 рис., 3 дод., 20 джерел.

КОГНІТИВНІ ПРОЦЕСИ, МОДЕЛЬ, ЛЕКСИЧНА ОБРОБКА,
ЧАСТОТНИЙ СЛОВНИК, САКАДА, ФІКСАЦІЯ

Об'єкт дослідження – процес читання тексту людиною.

Предмет дослідження – моделі передбачення швидкості читання текстів.

Метою роботи є розробка моделі передбачення часу читання вебсторінок та документів та її програмна реалізація.

Матеріали, методи та технічні засоби: мова програмування Python, середовище розробки графічних інтерфейсів Qt Designer.

Результати. Розроблено вдосконалену модель читання, що, на відміну від існуючих, дозволить здійснювати оцінку необхідного часу читання, враховуючи типографічні параметри.

Практична цінність роботи полягає в тому, що точне передбачення часу читання людиною-оператором інформаційних повідомлень та технічної документації дозволяє мінімізувати ризик прийняття запізнілих рішень у критичних ситуаціях, що сприяє підвищенню безпеки та зниженню ймовірності виникнення аварій, надзвичайних подій та загроз для життя людей та навколишнього середовища.

Висновки. Розроблено вдосконалену модель передбачення часу, необхідного для читання. Створено алгоритми опрацювання документів та вебсторінок. Описано спосіб оцінки часу виконання сакади. Створено метод обчислення розмірів літери для врахування у загальному алгоритмі.

Галузь використання – людино-машинна взаємодія.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 76 pages, 8 tables, 8 figures, 3 appendixes, 20 sources.

COGNITIVE PROCESSES, MODEL, LEXICAL PROCESSING, FREQUENCY VOCABULARY, SACCADIC, FIXATION

Object of study is a process of human text reading.

Subject of study are models for predicting text reading speed. Objective: to develop a model for predicting the reading time of web pages and documents and its software implementation.

Materials, methods, and technical tools: the Python programming language and the Qt Designer environment for developing graphical interfaces.

Results. An improved reading model has been developed, which, unlike existing models, enables the assessment of required reading time while considering typographic parameters.

Practical value of the work is accurate prediction of reading time by a human operator for informational messages and technical documentation minimizes the risk of delayed decision-making in critical situations, enhancing safety and reducing the likelihood of accidents, emergencies, and threats to human life and the environment.

Conclusions. An improved model for predicting the required reading time has been developed. Algorithms for processing documents and web pages have been created. A method for estimating the duration of saccades has been described. A technique for calculating letter dimensions for integration into the overall algorithm has been established.

Field of use is human–computer interaction.

ЗМІСТ

Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз проблеми та постановка завдання дослідження	11
1.1. Аналіз проблеми	11
1.2 Огляд моделей передбачення часу читання.....	15
1.3 Аналіз програмних аналогів та їх особливостей	16
1.4 Постановка завдань дослідження.....	19
1.5 Висновки за розділом 1	20
2 Матеріали та методи	21
2.1 Метод обчислення розмірів літери на основі параметрів шрифту.....	21
2.2 Оцінка часу виконання сакади	22
2.3 Розробка алгоритму опрацювання документів.....	24
2.4 Розробка алгоритму опрацювання вебсторінок	24
2.5 Розробка моделі передбачення часу необхідного для читання	25
2.6 Визначення функціональних вимог до програмного застосунку	28
2.7 Висновки за розділом 2	29
3 Основні рішення щодо реалізації системи	30
3.1 Розробка класів програмного забезпечення	30
3.2 Проєктування користувацького інтерфейсу	33
3.3 Висновки за розділом 3	35
4 Експлуатація, тестування та експериментальне дослідження програми	36
4.1 Призначення та умови застосування програми.....	36
4.2 Інструкція по експлуатації програмного забезпечення	36
4.2.1 Звернення до програми.....	36
4.2.2 Вхідні та вихідні дані.....	37
4.2.3 Повідомлення системи.....	37
4.3 Приклади експлуатації програмного застосунку	39
4.4 Висновки за розділом 4	41
Висновки	42
Перелік джерел посилання	44
Додаток А Технічне завдання	46

Додаток Б Текст програми	50
Додаток В Слайди презентації.....	70

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

UML – уніфікована мова моделювання що використовує графічні позначення для створення абстрактної моделі системи.

МС – одиниця часу в Міжнародній системі одиниць, яка дорівнює одній тисячній секунди.

ОС – операційна система.

Розбірливість (legibility) – здатність букв і символів бути розпізнаними індивідуально, незалежно від їх контексту в словах чи реченнях.

Сакада – швидкі, суворо узгоджені рухи очей, що відбуваються одночасно і в одному напрямку.

Фіксація – буква слова, на якому зупиняється око після здійснення сакади.

Читабельність (readability) – властивість тексту, що визначає легкість його сприйняття й читання на основі структури, шрифту та візуального оформлення.

ВСТУП

На сьогодні, передача інформації між людьми є основною складовою суспільної взаємодії, проте саме людина, її здібності та можливості є основним джерелом обмежень цього процесу. При цьому, сама людина має низку обмежень опрацювання та опанування цими знаннями. Швидкість сприймання інформації, розпізнавання тексту, усвідомлення прочитаного – все це когнітивні процеси, що потребують дослідження та виявлення власних обмежень.

Актуальність теми «Передбачення часу читання документів» обумовлена зростаючою потребою в автоматизованих системах обробки та аналізу тексту в різноманітних галузях, включаючи освіту, маркетинг, інтерфейси користувача, видавничу справу та інші сфери, де важливо враховувати часові витрати і сприйняття інформації.

Важливість передбачення часу, необхідного для читання зростає в контексті операторської діяльності в таких галузях, як авіація, медицина, енергетика та військова справа. В умовах обмеженого часу на прийняття рішень постійно виникає необхідність швидкого аналізу та обробки текстової інформації. У випадку, коли оператор, використовуючи дисплей, не встигає опрацювати текстовий матеріал у відведені йому терміни, наслідки можуть привести до збитків, при цьому можлива загроза людському життю та здоров'ю.

У системах електронного навчання знання часу, необхідного для засвоєння матеріалу, дозволяє персоналізувати освітні траєкторії, покращити оцінку прогресу студентів та допомогти у розподілі навчального часу.

Моделі передбачення часу читання допомагають розробникам інтерфейсів та програм надавати користувачам інформацію про час, який знадобиться для виконання завдань (наприклад, для читання угод або інструкцій), що покращує взаємодію з інтерфейсом.

Побудова моделей з урахуванням частотного словника, сакадичного руху очей та інших аспектів читання дозволяє глибше зрозуміти механізми

сприйняття тексту. Ці дослідження можуть дати нові знання про когнітивні процеси, пов'язані з читанням, і створити більш точні алгоритми для прогнозування поведінки користувачів.

У маркетингових та рекламних кампаніях передбачення часу читання допомагає оптимізувати контент, адаптуючи його під цільову аудиторію та покращуючи взаємодію з клієнтами.

Сучасні технології, такі як електронні книги та агрегати новин, змінюють способи споживання інформації, що робить актуальною задачу передбачення часу читання для адаптації тексту до різних пристроїв і форматів.

З розвитком електронних пристроїв, появою дисплеїв, а згодом й різноманітних комп'ютерних шрифтів було проведено чимало досліджень щодо впливу різноманітних типографічних характеристик написаного тексту та сприймання його людиною. Зазвичай, для виявлення закономірностей поведінки очей під час читання, дослідники збирали якомога більше даних, проводили систематизацію накопичених матеріалів та згодом підводили підсумки.

Одним з найскладніших аспектів у вивченні поведінки під час читання є аналіз руху очей. Так, моделі руху очей та їх поведінки були детально описані схожими способами та алгоритмами [1]-[3].

Безумовно, у процесі дослідження читання можливо виявити численні фактори, що впливають на загальний час сприйняття тексту, в тому числі й особисті вподобання читачів.

Таким чином, дослідження, спрямоване на передбачення часу читання має як практичне, так і теоретичне значення, а також має суттєвий внесок у розвиток технологій аналізу тексту та когнітивних досліджень. Можливість попередньо оцінити необхідний час читання тексту виступає інструментом, що прогнозує швидкість аналізу тексту як оператора, від якого залежить людське життя, так і фахівця, задачею якого є розміщення змістовного та зрозумілого тексту, враховуючи встановлені обмеження.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАННЯ ДОСЛІДЖЕННЯ

1.1. Аналіз проблеми

Важливість передбачення часу читання документів зростає в контексті операторської діяльності у різних галузях. В умовах обмеженого часу на прийняття рішень оператори часто стикаються з необхідністю швидкого аналізу та обробки текстової інформації. Можливість точно прогнозувати, скільки часу знадобиться на читання та сприйняття критичних даних допомагає знизити когнітивне навантаження та мінімізувати ризики, пов'язані із затримкою у реагуванні.

У кризових ситуаціях, коли необхідне оперативне реагування, правильне розподілення часу на читання та оцінку інформації може виявитися критичним фактором. Моделі передбачення часу читання, що базуються на методах аналізу тексту та когнітивних аспектах сприйняття, можуть допомогти операторам своєчасно приймати рішення, оптимізувати їх дії та зменшити ймовірність помилок.

У системах моніторингу та управління (наприклад, на виробництві, диспетчерських центрах та інших галузях) здатність передбачити час, необхідний для ознайомлення з важливими документами, може підвищити ефективність роботи. Це особливо актуально в умовах, коли оператори мають швидко оцінювати текстові звіти, інструкції чи інформаційні повідомлення у реальному часі.

У деяких галузях оператори стикаються з перевантаженням інформації через велику кількість текстових даних, що надходять за короткий проміжок часу. Можливість передбачити, скільки часу займе обробка тексту, дозволить більш ефективно управляти інформаційними потоками, розподіляти завдання між операторами та коригувати обсяг інформації, що подається в залежності від доступного часу.

Моделі передбачення часу читання можуть бути інтегровані в системи підтримки прийняття рішень, забезпечуючи операторів даними про те, скільки часу потрібно для обробки критично важливих текстів, що допомагає прискорити і поліпшити процес прийняття рішень в обмежені терміни.

Таким чином, дослідження передбачення часу читання документів не тільки має теоретичну та прикладну цінність для аналізу тексту, але й має критичну важливість для операторської діяльності, особливо в умовах обмеженого часу на прийняття рішень. Це дозволить підвищити ефективність роботи операторів, знизити навантаження в екстремальних ситуаціях та мінімізувати помилки, пов'язані із затримками в обробці інформації.

Численні наукові праці в різних галузях дослідження людської поведінки, впливу різноманітних факторів на неї мають довгу історію [3], [4]. Важливо, що хоча багато досліджень щодо впливу тих чи інших факторів на процес читання надають суперечливі результати, окремі аспекти, такі як рух очей і його взаємозв'язок з когнітивними процесами мають наукове підґрунтя.

Велика кількість публікацій зарубіжних авторів була присвячена дослідженням окомоторної, та пов'язаної з нею когнітивної активності, під час читання [1]-[3]. Всі дослідження демонструють результати, засновані на виявленні закономірностей поведінки очей звичайних користувачів.

Час, що витрачається на читання є досить індивідуальним показником і залежить від багатьох факторів, наприклад – втома, емоційний стан, особисті вподобання щодо відображення даного тексту на моніторі. У роботі буде розглянуто та використано лише науково підтверджені фактори, які можливо проаналізувати та врахувати. Наприклад, наявність або відсутність засічок у шрифтах має мінімальний вплив на швидкість читання, тоді як значення відхилень у часі слід віднести до індивідуального користувацького досвіду [2].

Деякі дослідження показують, що естетичність шрифтів та їх зручність для читання є важливими аспектами під час аналізу руху очей [2]. Тому, призначення шрифтів не обмежується виключно функціональними характеристиками, але також може надавати дані для корегування висновків

щодо поведінки очей під час когнітивних процесів. Так, шрифти мають суттєвий вплив на здатність краще запам'ятовувати інформацію, або навпаки, деякі комбінації шрифтів та кеглів можуть лише сповільнити та ускладнити роботу з текстом [2].

Окремою проблемою виступає коректне використання деяких термінів іншомовного походження, наприклад: «readability», «legibility». Причина у тому, що переклад деяких термінів та словосполучень, може бути неправильним, а отже, зміст або значення може бути помилковим або частково втраченим. В даному випадку, термін «readability» є показником комфорту сприйняття, розуміння та можливості запам'ятовувати прочитане [4]. Поняття «legibility» найчастіше пояснюється як розбірливість самого слова, зручність розпізнавання символів, представлених певним шрифтом [4].

Зауважимо, що розмір кеглів впливає на кількість символів, що здатне одночасно охопити людське око. Було визначено що під час фіксації погляду область чіткого сприйняття обмежена 3-4 символами ліворуч та 14-15 символами праворуч від точки фіксації [2]. Враховуючи особливості ока, при зміні розміру шрифту тексту, кількість символів, що одночасно потрапляють у область сприйняття буде збільшено або зменшено. Так, нижній рядок імітує гостроту зору у відповідних відсотках (рис. 1.1). При цьому, труднощі розпізнавання тексту навколо точки фіксації «fixation point» збільшуються з відстанню [6].

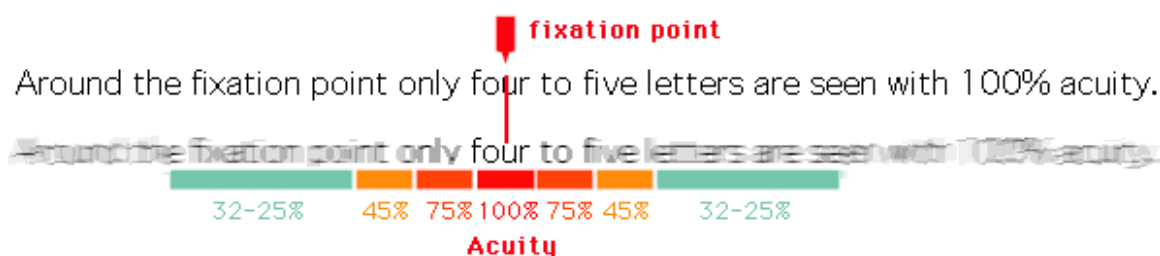


Рисунок 1.1 – Діаграма гостроти зору під час читання [6]

Відомо, що під час читання, око виконує короткі й швидкі рухи вздовж рядку, між якими відбуваються короткі зупинки. Під час переведення погляду

на наступний фрагмент тексту, око має зупинитися на деякому символі для подальшої фіксації.

Виявлено, що розподіл точок приземлення на слові не є деякою випадковістю, натомість має тенденцію мати гаусову форму (рис. 2.1).

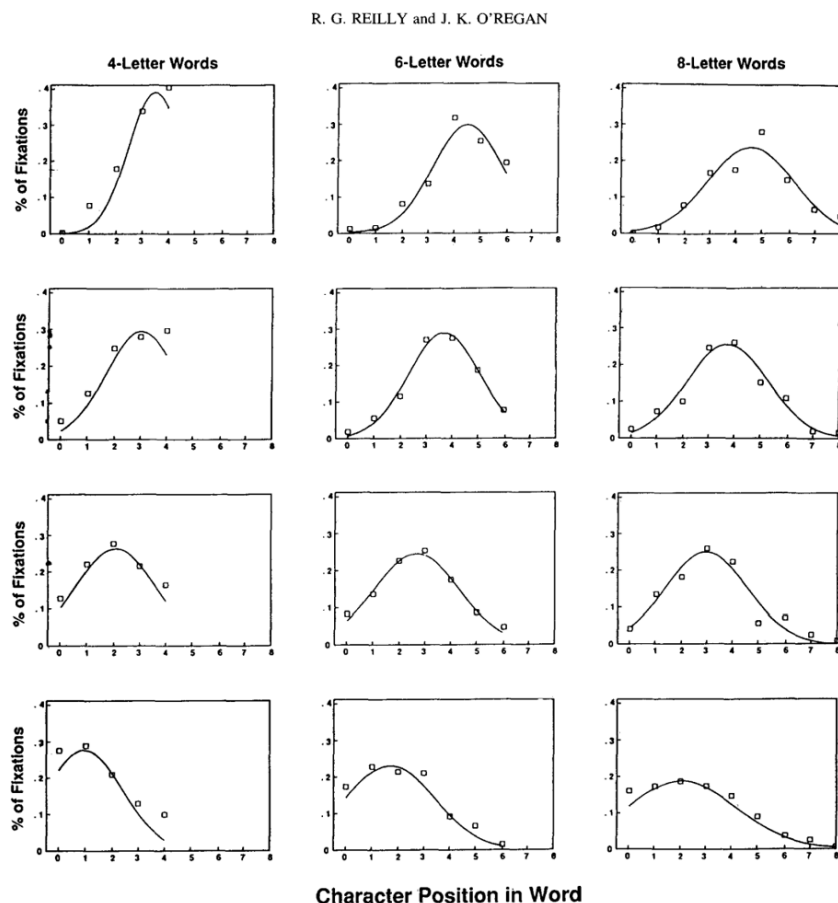


Рисунок 2.1 – Тенденція концентрації погляду навколо центра слова [1]

Існує загальне правило погляду зупинитися навколо центра слова, з деяким відхиленням, залежно від дистанції, що подолало око під час сакади [1]. Відповідно, чим більшою є амплітуда сакади, тим більшим спостерігатиметься розкид розподілу точок приземлення навколо центру слова. Процес зміщення погляду та вибір наступного цільового слова має бути доповненим певною стратегією або чітко визначеною поведінкою когнітивних процесів читачів.

Далі буде розглянуто основні запропоновані моделі, що пояснюють природу переміщення погляду, сакад та інших аспектів даних досліджень.

1.2 Огляд моделей передбачення часу читання

Технології спостереження за рухом та поведінкою очей дозволяють точно визначити тривалість фіксацій та сакад під час читання, зафіксувати напрямок погляду та траєкторію стрибків очима. Завдяки вивченню рухів очей, були отримані відповіді на питання, як локалізується увага, яким чином ідентифікуються слова, скільки часу необхідно для лінгвістичної обробки тексту, які когнітивні процеси беруть участь у дешифруванні друкованого тексту та багато інших [5]. Зауважимо, що питання, що саме керує рухом очей під час читання, довгий час залишалося невирішеним. Тим не менш, частина дослідників зосередила свої зусилля на спробах краще зрозуміти зв'язок ока та свідомості, вивчити когнітивні процеси, що відповідають за прийняття рішень, що керуються сакадами. Однією з головних метрик є час, впродовж якого очі фіксуються на словах, а також як часто ці фіксації переміщуються з одного слова на інше через сакади. Моделі, які описують ці процеси, дозволяють передбачити не лише час розпізнавання окремих слів але й загальний час, необхідний для ознайомлення з текстом.

На сьогодні, відомі наступні моделі процесу читання [1]:

- окорухові моделі – одним з базових підходів є модель «Word by word» або «Слово за словом». В основі моделі є переміщення погляду в сторону на постійну величину під час кожної з сакад, а шум в окоруховій системі викликає варіації в положенні при посадці, які виявляються емпірично. Однак, результати симуляцій відображають постійні пропуски через короткі слова або навпаки, переміщення ока зупиняється ще на попередньому слові замість цільового. Це значно зменшує ефективність стратегії в реальному читанні;
- лінгвістичні стратегії – прийнято вважати, що якщо лінгвістичні чинники беруть участь у миттєвому контролі очей, швидше за все, вони діють на лексичному рівні, наприклад, враховується частота використання слова, а не синтаксичні чи семантичні властивості тексту вищого рівня. Таким чином, передбачуваність слова у контексті може суттєво впливати на тривалість

фіксації. Високочастотні слова зазвичай опрацьовуються швидше, при цьому вони можуть пропущені периферійним зором. Так, модель «Skip High-Frequency Words» або «Пропуск слів з високою частотою» передбачає ймовірність читача пропустити високочастотне слово, орієнтуючись при цьому на слова з меншою частотою.

Зауважимо, що наступна модель «Attention Shift», також відома як «Стратегія зміщення уваги» є більш розгорнутою версією попередньої стратегії. Вважається, що обробка наступного слова може бути повністю опрацьована за допомогою «ефекту попереднього перегляду» [1]. Дана модель більш точно описує можливу поведінку руху очей, враховуючи при цьому вплив лексичної обробки, що відбувається під час читання слова.

Використовуються також змішані стратегії, що дають можливість поєднання декількох моделей одночасно, беручи до уваги як лексичні так і окорухові чинники. Наприклад, процес читання моделюється як послідовне читання слова за словом, при цьому у місцях, де текст вважається легким для сприйняття користувачем, будуть застосовані правила з інших моделей.

Таким чином, моделі передбачення часу читання можуть містити різну основу, підходи та фактори, що необхідно враховувати під час аналізу. Найбільш успішною є модель «Стратегія зміщення уваги», так як саме ця модель враховує можливості сприйняття людського мозку, особливості роботи периферійного зору, що надає більшої точності у подальших розрахунках.

1.3 Аналіз програмних аналогів та їх особливостей

Більшість з доступних рішень в мережі Інтернет орієнтовані на швидку поверхневу оцінку часу, необхідного для читання наданого фрагменту тексту. Знаючи приблизну техніку читання користувача можливо розраховувати скільки часу займе перегляд певного тексту. Завдяки інструментам з таким підходом, оцінка часу відбувається майже миттєво, тому користувач може з легкістю перевіряти час, необхідний для перегляду швидко та без затримок.

Одним з таких програмних аналогів є плагін «Words to Time» (рис. 1.3). Програмний застосунок приймає на вхід користувацькі налаштування, такі як показник техніки читання користувача про себе та в голос. У такому випадку, відповідальність за похибку в обрахунках накладається на користувача, адже читач вже розуміє принцип оцінки часу на основі заданих параметрів.

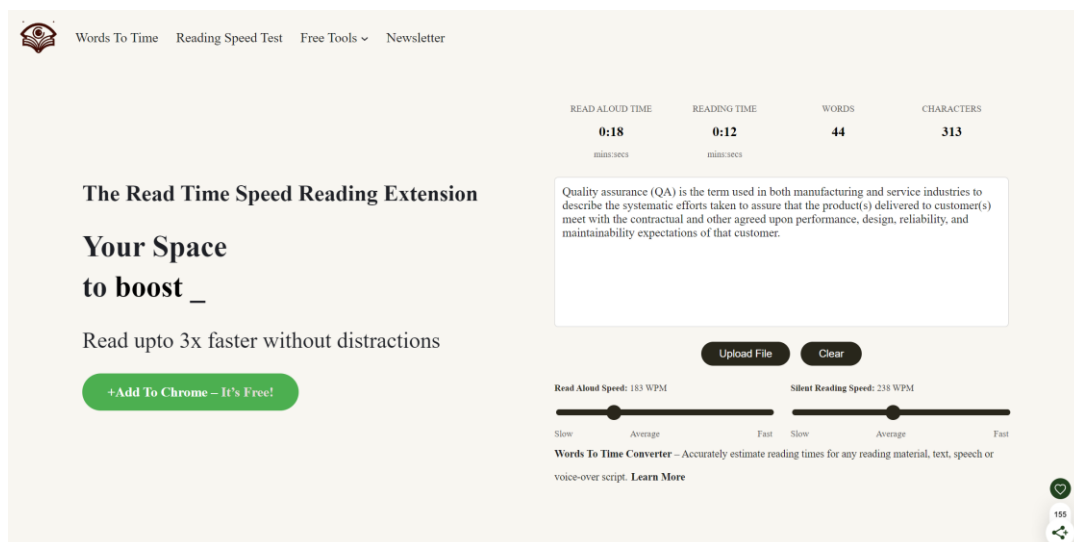


Рисунок 1.3 – Інтерфейс плагіну «Words To Time»

При цьому, так як програма враховує індивідуальний параметр кожного з користувачів, а саме – їх техніку читання, дані результати будуть наближені до реальних показників. Повний алгоритм програми для оцінки часу відомий та представлений самими розробниками [7]. Метод оцінки полягає в знаходженні пропорції часу, необхідного для читання між поточним користувачем та середнім показником попередніх учасників експерименту (або за вказаним користувачем числом слів в хвилину). Також, даний плагін надає можливість завантажити документ який буде зчитано та оцінено, що може бути корисним доповненням до основних можливостей.

Алгоритм даного застосунку не враховує складність сприйняття слів, тому будь-який введений символ вважається програмою словом, що є суттєвим недоліком. Так як цифри та розділові знаки можуть розпізнаються читачем інакше, врахування їх у якості окремих слів є некоректним.

Найбільш близьким аналогом є програма симуляції процесу читання [1], що базується на розглянутих у п. 1.2 стратегіях читання. Дана програма містить загальний алгоритм, окремі кроки якого можуть змінюватися в залежності від обраної моделі. Приведена програма не демонструє результатів оцінки загального часу, при цьому відображає загальну тенденцію приземлення очей та їх поведінку. Зауважимо, що загальний алгоритм програми містить деякі зразки розрахунків в залежності від позицій очей, розповсюдженості слів а також окремі правила та обмеження для більшої схожості на реальний процес читання. Дана модель не враховує типографічні особливості, що можуть суттєво вплинути на кінцевий результат. Наприклад, розмір, шрифт та розташування тексту є індивідуальними параметрами для текстових документів, тому ці фактори є важливими при оцінюванні часу читання.

Дані аналоги стануть основою для створення нового програмного застосунку, що буде оснований на моделях, розроблених у даній роботі. Порівняльний аналіз основних можливостей аналогів разом з розроблюваним застосунком наведено у табл. 1.1.

Табл.1.1 – Аналіз програмних аналогів

Назва	Words To Time	Модель «Зміщення уваги»	Програмний застосунок
1	2	3	4
Оцінка часу на основі документів	Так	Ні	Так
Оцінка часу перегляду вебсторінки	Ні	Ні	Так

Продовження таблиці 1.1

1	2	3	4
Врахування ймовірного відхилення результатів	Ні	Так	Так
Обробка тексту, написаного різними мовами	Так	Так	Ні
Врахування положення тексту	Ні	Ні	Так
Врахування впливу шрифтів	Ні	Ні	Так

Зважаючи на всі ключові особливості які було виявлено під час аналізу програмних аналогів, буде складено функціональні вимоги до розроблюваного застосунку. Так, існуючі аналоги демонструють суттєву перевагу, а саме – здатність працювати з текстом будь-якої мови. Розроблюваний застосунок буде використовувати лише один частотний словник англійської мови, при цьому плагін «Words To Time» здатен проаналізувати файл будь-якою мовою. Модель «Зміщення уваги» описує лише спосіб використання словника, не обмежуючись конкретною мовою, а значить матиме більшу гнучкість. Однак, приведені аналоги мають обмеження щодо врахування додаткових параметрів тексту та часу, необхідного для зміщення уваги та пошуку наступного блоку тексту.

1.4 Постановка завдань дослідження

Завданням даного дослідження є розробка вдосконаленої моделі читання, що, на відміну від існуючих, дозволить здійснювати оцінку

необхідного часу читання, враховуючи типографічні параметри. Таким чином, для реалізації поставленого завдання дослідження, необхідно розробити:

- метод обчислення розмірів літери на основі параметрів шрифту;
- алгоритм оцінки часу, необхідного для виконання сакади;
- алгоритм опрацювання документів;
- алгоритм опрацювання вебсторінок;
- загальну модель передбачення часу необхідного для читання;
- програмний застосунок для відображення результатів роботи моделі.

1.5 Висновки за розділом 1

Було розглянуто актуальність дослідження, яка полягає в необхідності передбачення часу, необхідного для читання текстів. Визначено перелік галузей, в яких важливим фактором проектування інтерфейсів є можливість точного передбачення часу, необхідного для сприйняття наведеного тексту.

Проведено огляд існуючих моделей передбачення часу, серед них: окомоторні, лексичні та змішані стратегії. Виявлено, що кожна з моделей має свої особливості та обмеження, які необхідно враховувати для забезпечення більш точного результату оцінки. Визначено завдання дослідження для реалізації вдосконаленої моделі читання.

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Метод обчислення розмірів літери на основі параметрів шрифту

Важливою особливістю моделі, що розробляється, має бути можливість зберігання розміру літер для кожної комбінації назви шрифту та кеглю, що відображаються на екрані, для їх подальшого використання при обчисленні загального часу, необхідного для читання. Для цього передбачено створення та використання спеціального файлу, що міститиме усі обчислені показники фізичних розмірів літер. Далі розглянуто алгоритм, який описує метод обчислення розмірів літери на основі його параметрів, результатами якого будуть середня ширина та висота кожної з малої літери англійського алфавіту.

Вхідними даними даного алгоритму є назва виявленого на сайті чи в документі шрифту, та розмір кеглю, що досі не зустрічався і не є збереженим у спеціальному файлі з вже розрахованими розмірами літер.

Вважаємо, що стандартним шрифтом є Times New Roman, 14 кеглю, при цьому всі інші комбінації вважатимуться новими та нестандартними.

Нижче представлено повний алгоритм обчислення середнього розміру малої літери довільного шрифту зі своїми параметрами.

Крок 1. Генерується набір текстових блоків, кожен з яких складається з однієї літери латинського алфавіту.

Крок 2. Встановлюється необхідний розміру кеглю та назва шрифту.

Крок 3. Після створення текстових блоків, повертаються значення координат створених текстових фрагментів, визначені в пікселях.

Крок 4. Отримавши значення всіх координат текстового блоку, розраховуються розміри кожної з літер у пікселях.

Крок 5. Роздільна здатність дисплею, що має бути адаптована в подальшому, розраховується за формулою [8]:

$$PPI (monitor) = \frac{Number\ of\ Pixels}{Size\ in\ Inches}, \quad (2.1)$$

де PPI – роздільна здатність дисплею;

$Numbers\ of\ Pixels$ – кількість пікселів по горизонталі або вертикалі;

$Size\ in\ Inches$ – розмір дисплею за тієї ж віссю що й $Number\ of\ Pixels$.

Крок 6. Для знаходження розміру літери адаптуємо формулу (2.1). Враховуючи, що 1 дюйм = 2,54 см, формула для розрахунку висоти та ширини текстового блоку:

$$size = \frac{Size\ in\ Inches}{PPI} \times 2.54 \quad (2.2)$$

де $size$ – фізичний розмір літери, що видно на дисплеї, перетворений у см.

Крок 7. Далі, встановивши розміри кожної з літер, розраховується середньо арифметичний показник для досліджуваного шрифту та його розміру.

Крок 8. Отриманий результат зберігається в файл з іншими прорахованими комбінаціями кеглів та шрифтів для подальшого швидкого доступу до них.

2.2 Оцінка часу виконання сакади

Наступним необхідним методом, необхідним для розробки вдосконаленої моделі є алгоритм оцінки часу, необхідного для виконання сакади. Задачею алгоритму є визначення розміру відстані, що долає око при зміщенні погляду з кінця одного слова до початку іншого. В [9] розглянуто скільки часу займатиме сакада і яка залежність виникає між її розміром та часом, необхідним для переведення погляду. Процес зміщення ока можна зобразити у вигляді прямокутного трикутника. Нижче описано повний алгоритм, необхідний для оцінки часу виконання сакади.

Крок 1. Використовуючи стандартні формули прямокутного трикутника (рис.2.1), необхідно знайти кут (амплітуду), що робить око під час сакади.

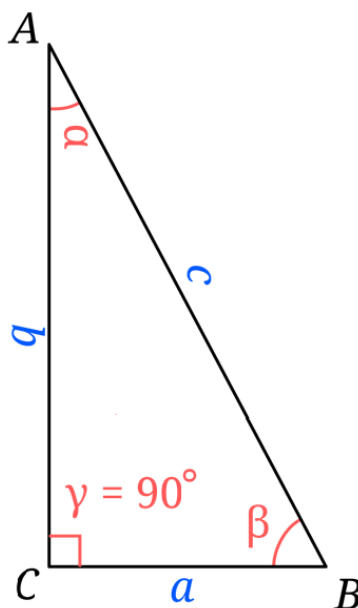


Рисунок 2.1 – Загальний вигляд прямокутного трикутника

Крок 2. Згідно рис.2.1 кут альфа (λ) є кутом, сакада якого буде виконана на відстані b (дистанція до екрану). Тоді катетом СВ є відстань сакади, визначена у см. Далі визначимо кут огляду:

$$\alpha = \arctan\left(\frac{BC}{AC}\right) = \arctan\left(\frac{a}{b}\right) \quad (2.3)$$

Крок 3. Знайшовши кут λ , обраховується час, витрачений безпосередньо на сакаду, згідно формули [7]:

$$T = 2.2 * (A - 5) + 21 \text{ (ms)}, \quad (2.4)$$

де A – кут огляду;

T – тривалість сакади.

У якості результату повертається час, що необхідний людині для виконання сакади. Так як при розрахунках не використовується середнє значення та стандартне відхилення [8], результат алгоритму повертає лише час, виражений в мс, необхідний для виконання сакади.

2.3 Розробка алгоритму опрацювання документів

Необхідно розробити окремий алгоритм, що здійснюватиме опрацювання документів. Для виконання алгоритму буде використано бібліотеку PyMuPDF (бібліотека для обробки документів формату PDF засобами мови програмування Python). Дана бібліотека дозволяє отримувати інформацію про місце положення, властивості елементів сторінки). Нижче наведено повний алгоритм, що дозволяє проаналізувати всі сторінки завантаженого документу.

Крок 1. Відкриття файлу що необхідно проаналізувати.

Крок 2. Обробка наступної існуючої сторінки.

Крок 3. Знаходження найближчого слова.

Крок 4. У випадку, якщо всі слова були оброблені, перехід до кроку 9.

Крок 5. Якщо знайдене слово має власні, унікальні характеристики шрифту, запустити метод обчислення розмірів (п. 2.1).

Крок 6. Визначити час лексичної ідентифікації слова згідно пункту 2.5.

Крок 7. Пошук наступного найближчого слова.

Крок 8. У випадку, якщо слово існує, виконати обрахунок часу сакади за п. 2.2, інакше – перевірити наявність наступної сторінки.

Крок 9. Програма повертає орієнтовну оцінку часу читання, враховуючи стандартне відхилення.

2.4 Розробка алгоритму опрацювання вебсторінок

Перш ніж перейти до безпосереднього алгоритму, необхідно деталізувати процес опрацювання вебсторінки. Загалом, алгоритм читання вебсторінок досить схожий на перегляд документу, проте важливо описати деякі особливості.

Передусім, так як вебсторінка складається не лише з тегів, а також містить коментарі та текст, усі ці елементи узагальнюються як «вузли». В даному

контексті слід також розглянути термін DOM (об'єктна модель документа), тобто кожна вебсторінка представлена у вигляді ієрархічного дерева елементів, які вважаються вузлами.

Для програмної реалізації алгоритму буде використано засоби бібліотеки Selenium (інструмент для автоматизації роботи в браузері). Даний інструмент дозволяє отримувати стилі поточного текстового вузла у вигляді словника властивостей що повністю відповідає потребам алгоритму.

Крок 1. Завантаження вказаної користувачем вебсторінки.

Крок 2. Пошук наступного існуючого вузла.

Крок 3. Якщо вузол є текстовим, слова розділяються одне від одного, окремо відділяються розділові знаки та пробіли, інакше перейти до кроку 7.

Крок 4. На основі отриманих даних, формується програмний об'єкт, що представляє текстовий вузол.

Крок 5. Якщо знайдене слово має нестандартні характеристики шрифту, запустити метод обчислення розмірів (п. 2.1).

Крок 6. Визначення часу лексичної ідентифікації слова (п. 2.5).

Крок 7. Відбувається пошук наступного найближчого вузла.

Крок 8. Якщо вузла не виявлено, перехід до кроку 10, інакше до кроку 3.

Крок 9. Визначення часу, необхідного для здійснення сакад за пунктом 2.3 та повернення до кроку 4.

По завершенні аналізу DOM сайту, програма повертає орієнтовну оцінку часу читання, враховуючи стандартне відхилення.

2.5 Розробка моделі передбачення часу необхідного для читання

В основу моделі передбачення часу читання було покладено дослідження [1]. Оптимальною позицією приземлення ока в слові є середина слова [1], її прийнято за нуль, в усіх інших випадках приземлення ока буде обраховано враховуючи відхилення точки приземлення від оптимальної позиції. Також, важливо враховувати вплив розповсюдженості слова на час

розпізнавання слова. Існують словники, що містять інформацію, наскільки часто вживаються ті чи інші слова. Будь-який запропонований користувачем документ або вебсторінка можуть містити числа, час їх розпізнавання необхідно врахувати в загальній моделі передбачення часу. Для цього, можливо перетворити цифровий запис кожного окремого числа у текстовий вираз. Наприклад, число «2024» може сприйматися користувачем як «дві тисячі двадцять чотири». Тоді час лексичної ідентифікації конкретного числа буде сумою всіх ідентифікацій кожного окремого наведеного слова.

Розглянемо алгоритм роботи моделі передбачення часу, необхідного для читання та розпізнавання цільового слова.

Крок 1. Розрахунок середнього місця приземлення по формулі [1]:

$$m = 3.3 + 0.49d * k \quad (2.5)$$

де d – дистанція запуску кількості символьних позицій від попередньої точки фіксації до центра наступного слова;

k – коефіцієнт для врахування впливу знайденого розміру шрифту відносно стандартного.

Крок 2. Після визначення середнього місця приземлення, для формули (2.5) необхідно визначити стандартне відхилення. Середня точка приземлення, що розраховується, необхідна лише для обрахунку в формулі (2.7) вірогідності приземлення для кожного окремого символу цільового слова.

Крок 3. Стандартне відхилення визначаємо по формулі [1]:

$$Sd = 1.318 + 0.000518 d^3 \quad (2.6)$$

Крок 4. Щоб дізнатися місце наступного приземлення, для кожного символу слова та навколо нього необхідно розрахувати вірогідність приземлення саме на певну позицію в слові [1]:

$$f(x; m; sd) = \frac{1}{\sqrt{2\pi \cdot sd}} e^{\frac{(x-m)^2}{2 \cdot sd^2}} \quad (2.7)$$

Крок 5. Визначення, чи буде відбуватися рефіксація погляду [1]:

$$y = a + 0.03x^2 \quad (2.8)$$

де x – відхилення в символічних позиціях від оптимальної позиції перегляду слова.

x вимірюється як символи від оптимальної позиції перегляду слова (приблизно 0.5 символічного простору зліва від центра слова). Для рівняння (2.9) визначимо a , це найменша точка лінійної функції, що залежить від довжини слова [1]:

$$a = 0.15L - 0.0034, \quad (2.9)$$

де L – довжина слова в буквах.

В наступних рівняннях нульова позиція буде вважатися першим пробілом зліва від слова а не його центром.

Крок 6. Якщо за формулою (2.8) визнано необхідність рефіксації, використаємо формулу (2.10) для знаходження середньої затримки в мс [1]:

$$m = base + range \left[1 - \frac{|loc-middle|}{(length/2)} \right] \quad (2.10)$$

Крок 7. Визначення стандартного відхилення [1]:

$$sd = 0.1m \quad (2.11)$$

Знайдена середня затримка у формулі (2.10) буде додана в загальний час, а стандартне відхилення буде враховане при оголошенні результату.

Крок 8. Після цього, для визначення часу, витраченого на розпізнавання слова, скористаємося формулою лексичної ідентифікації [1]:

$$m' = base5 + 15(length - 5) + 40(1 - freq) \quad (2.12)$$

Крок 9. Так як формула (2.12) дає час розпізнавання центрально зафіксованого слова, необхідно узагальнити значення для різних позицій фіксації в цьому ж слові [1]:

$$m = m' + \frac{range}{(length/2)} |loc - middle| \quad (2.13)$$

Крок 10. Отже, стандартне відхилення за формулою (2.11) використовується для обрахунку нормального розподілу, і після цього отримане середнє значення буде результатом даного алгоритму. Таким чином, розроблено модель здатна точно розрахувати час, необхідний для лексичної ідентифікації будь-якого слова в залежності від його довжини та точки фіксації.

2.6 Визначення функціональних вимог до програмного застосунку

Зважаючи на розроблену модель (пункт 2.5), необхідно визначити перелік функціональних вимог до програмного застосунку.

До загальних функціональних вимог слід віднести:

- врахування положення всіх фрагментів тексту на сторінці документу чи сайту шляхом визначення додаткового часу, що потребує користувач для виконання сакади до тексту, що читається;
- можливість введення розміру дисплею та його роздільної здатності користувачем під час роботи з програмою;

- обчислення та подальше зберігання фізичних розмірів літери залежно від характеристик дисплею, вказаних користувачем;
- обробка вказаного користувачем документу або вебсторінки;
- надання графічного інтерфейсу користувачу для зручності використання програмного рішення;
- автоматичне перетворення файлів формату .docx та .doc до формату PDF зі збереженням всіх властивостей документу;
- визначення впливу особливостей різних комбінацій шрифтів та їх кеглів на процес читання;
- використання встановлених шрифтів на ОС Windows в директорії «C:\Windows\Fonts» для розрахунку реальних розмірів літер;
- результат виконання алгоритму має бути приведений до часу згідно одиниць Сі та поданий, враховуючи стандартне відхилення кінцевої оцінки часу, якщо таке буде.

2.7 Висновки за розділом 2

Було розроблено метод обчислення розмірів літери на основі параметрів шрифту. Реалізовано розпізнавання чисел під час читання. Створено алгоритм оцінки часу, необхідного для виконання сакади. Створено алгоритм опрацювання документів та алгоритм опрацювання вебсторінок. Описано загальну модель передбачення часу необхідного для читання. Визначено перелік функціональних вимог до програмного застосунку для реалізації методів та алгоритмів. Для можливості опрацювання файлів з розширенням .docx та .doc, було запропоновано перетворювати користувацькі документи у формат pdf. Таким чином, програмний застосунок стане більш гнучким у порівнянні з існуючими аналогами. Обробка вебсторінок відбуватиметься за допомогою синтаксичного аналізу дерева тегів та виокремлення тексту що відображається. Для попередньої підготовки аналізу тексту, визначено директорію в ОС Windows, де розташовані стандартні шрифти.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ СИСТЕМИ

3.1 Розробка класів програмного забезпечення

В ході проектування програмного застосунку було розроблено діаграму класів та визначено залежності і зв'язки між ними за допомогою нотацій мови UML (рис.3.1). Загальний алгоритм передбачення часу читання реалізований в класі Model, що вважається основою всієї програми.

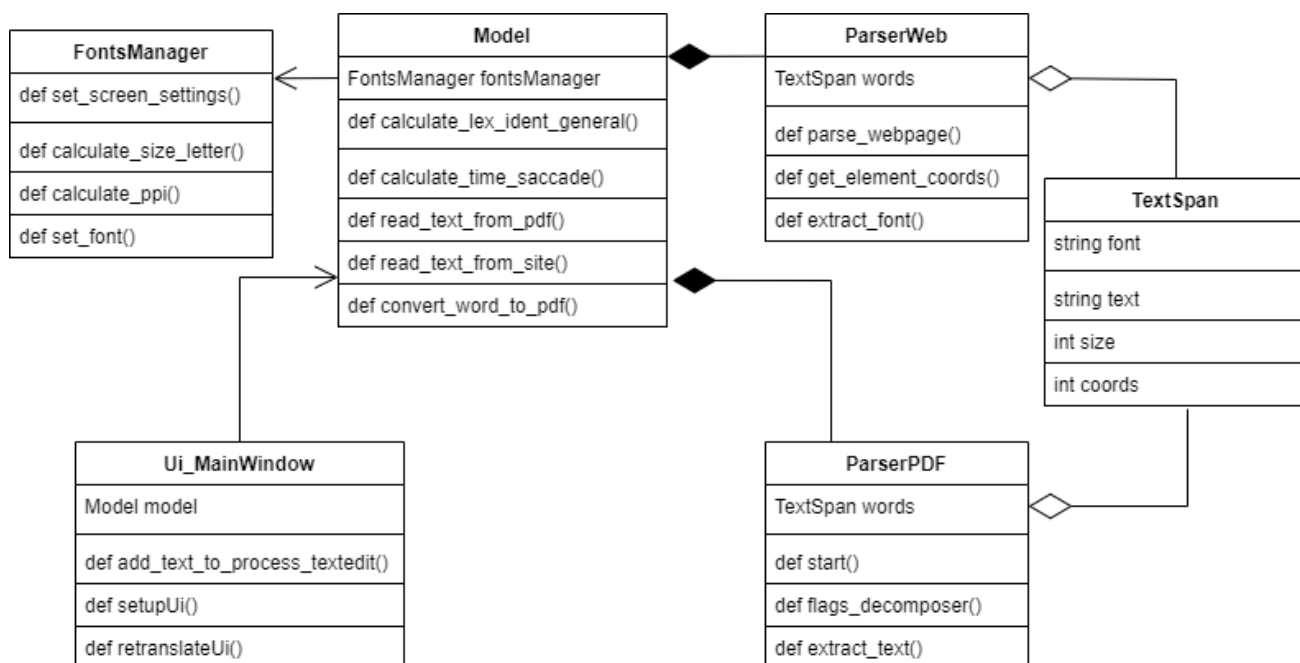


Рисунок 3.1 – Діаграма класів програми

Дана діаграма відображає шість класів, пов'язаних одне з одним спеціальними відношеннями. Клас Model пов'язаний з класом FontsManager спрямованою бінарною асоціацією. При цьому, клас Model відображає зв'язок «Композиція» з класами ParserPDF та ParserWeb, так як екземпляри даних класів не можуть існувати без моделі. Класи ParserWeb та ParserPDF пов'язані з TextSpan через зв'язок «Агрегація», оскільки об'єкт TextSpan створюється на основі текстового вузла і використовується в програмі як екземпляр класу, що може існувати самостійно. Клас Ui_MainWindow, що представляє собою головну форму програми та пов'язаний через зв'язок «Бінарна асоціація» з

моделлю, так як завдяки класу Model, на який посилається Ui_MainWindow, результати оцінки часу читання відображаються у відповідних текстових полях.

Клас Model є ключовим класом, що містить в собі реалізації алгоритмів та методів розробленої моделі (пункт 2.1-2.5). Окрім цього, дана структура даних містить екземпляр класу FontManager для можливості врахування параметрів шрифтів. Перелік основних методів класу наведений в табл. 3.1.

Таблиця 3.1 – Методи класу Model

Метод	Опис
calculate_lex_ident_general()	Метод обчислення часу, витраченого на лексичну ідентифікацію
calculate_time_saccade()	Метод обрахунку часу сакади
read_text_from_pdf()	Метод запуску аналізу документу з розширенням .pdf
read_text_from_site()	Метод запуску аналізу вебсторінки
convert_word_to_pdf()	Метод конвертування файлів формату .docx та .doc до формату .pdf

Клас TextSpan необхідний для зберігання усієї інформації знайденого текстового вузла на сайті чи в документі – текстовий зміст та властивості шрифту. Ключові параметри приведено нижче, в табл. 3.2.

Таблиця 3.2 – Параметри класу TextSpan

Параметр	Опис
string font	Назва шрифту текстового вузла
string text	Текстовий зміст вузла
int size	Розмір кеглю
int coords	Координати текстового вузла

FontsManager – клас, що призначений для визначення впливу виявлених шрифту та кеглю знайденого тексту та адаптації формул, розміщених в класі Model для врахування при обрахунку. Нижче наведено опис основних методів (табл. 3.3).

Таблиця 3.3 – Методи класу FontsManager

Метод	Опис
set_screen_settings()	Встановлення параметрів дисплею
def calculate_size_letter()	Розрахунок розмірів літери
def set_size_font()	Встановлення розмірів літери
def set_font()	Встановлення назви шрифту

ParserWeb це клас, що займається первинною обробкою вебсторінки та збереження усіх текстових вузлів у вигляді об’єктів класу TextSpan. Нижче наведено основні методи даного класу (табл. 3.4).

Таблиця 3.4 – Методи класу ParserWeb

Метод	Опис
def parse_webpage()	Метод завантаження та обробки вебсторінки
def get_element_coords()	Метод отримання координат розташування текстового вузла
def extract_font()	Метод визначення назви шрифту

ParserPDF це клас, що аналізує зчитану інформацію з файлу з розширенням .pdf та зберігає її в перелік об’єктів класу TextSpan. Опис головних методів наведено у табл. 3.5.

Таблиця 3.5 – Методи класу ParserPDF

Метод	Опис
def start()	Метод запуску обробки документу
def flags_decomposer()	Метод виявлення всіх параметрів текстового вузла
def extract_text()	Метод зчитування текстового змісту

Ui_MainWindow – даний клас забезпечує головне вікно програми та містить в собі графічний інтерфейс. З його допомогою користувач вказує шлях до необхідного файлу або вебсторінки та встановлює параметри дисплею. Форма відображає детальний процес обробки та результат оцінки часу читання. Нижче наведено методи даного класу (табл. 3.6).

Таблиця 3.6 – Методи класу Ui_MainWindow

Метод	Опис
def add_text_to_process_textedit()	Метод сповіщення та відображення про наступний крок алгоритму
def setupUi()	Метод завантаження графічного інтерфейсу
def retranslateUi()	Метод перейменування елементів керування

3.2 Проєктування користувацького інтерфейсу

Для використання розроблюваного застосунку передбачається одна форма, на якій розміщено всі необхідні елементи керування (рис.3.2). Дана форма містить поля для вводу всіх необхідних параметрів дисплею.

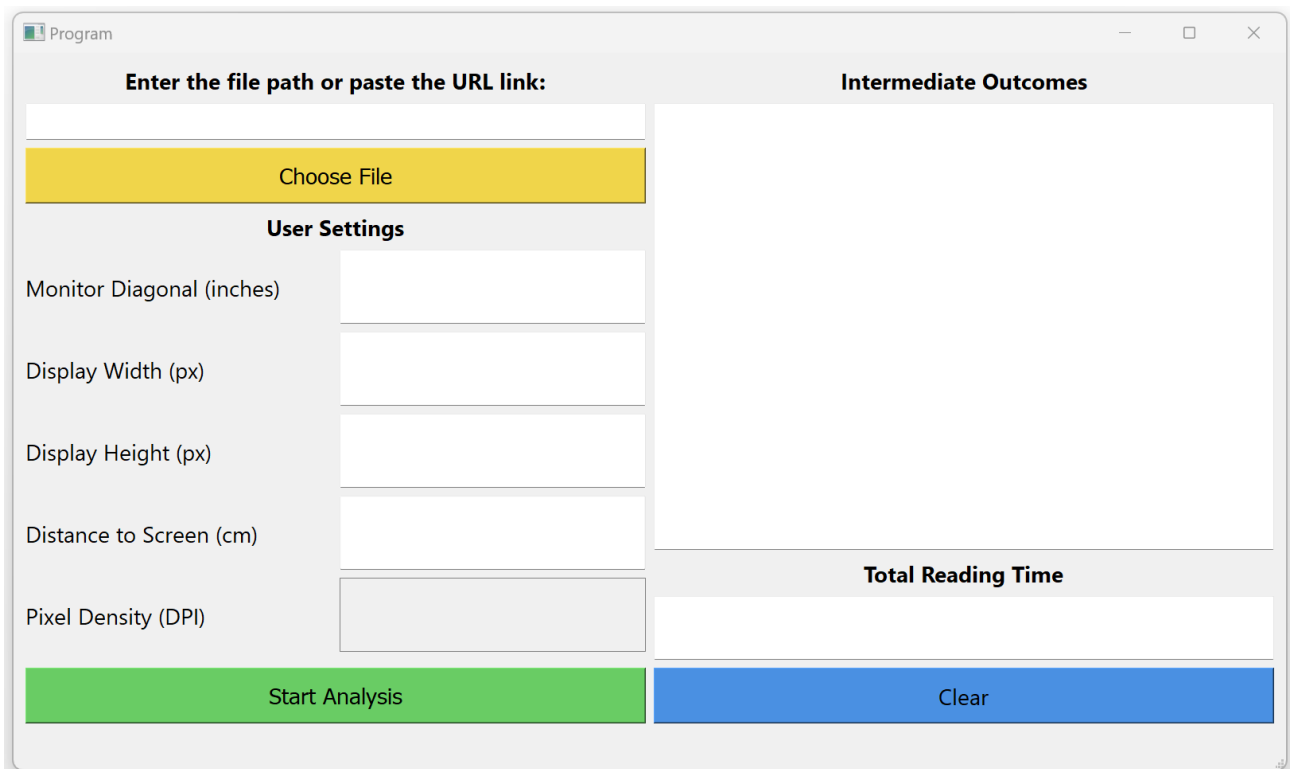


Рисунок 3.2 – Головне вікно програми

Окремо розміщене поле дозволяє введення посилання на вебсторінку або вказання файлу до аналізу. При натисканні на кнопку «Choose File», буде відкрито діалогове вікно «Select a file» де користувачу будуть доступні файли з розширенням .docx, .doc та .pdf. Розглянемо фрагмент коду, що дозволяє виокремити лише ті розширення файлів, що підтримує модель:

```
def get_path_chose_file(self):
    options = QFileDialog.Options()
    options |= QFileDialog.DontUseNativeDialog

    file_filter = "Word Files (*.doc *.docx);;PDF Files (*.pdf);;
    HTML Files (*.html *.htm)"

    file_path, _ = QFileDialog.getOpenFileName
    (None, "Select File", "", file_filter, options=options)

    return file_path
```

В даному методі містяться налаштування з вказаними форматами, а саме «doc», «docx», «pdf» а також формати, що використовуються для створення вебсторінок: «html», «htm». Клас QFileDialog розгортає діалогове вікно для вибору користувацького файлу та надає шлях розташування в ОС.

Поле «Pixel Density» є неактивним, так як щільність пікселів визначається автоматично за введеними користувачем параметрами дисплея [20]. Діагональ дисплея, його ширина та висота є стандартними характеристиками кожного з моніторів, і заноситься у відповідні текстові поля.

Програмний застосунок обраховує щільність пікселів на основі введених параметрів екрану та здійснює аналіз, враховуючи відстань до екрану, що вказав користувач. Поточний стан процесу моделювання відображається у полі "Intermediate Outcomes" та демонструє кожен крок опрацювання тексту.

Елемент QPushButton «Start Analysis» запускає обробку документу або вебсторінки та безпосередній аналіз. Елемент «Clear» дозволяє прибрати результати моделювання, залишаючи при цьому введені параметри дисплею.

3.3 Висновки за розділом 3

Було описано класи, що реалізують програмний застосунок та визначено залежності і зв'язки між ними. Описано основні методи класів та наведено їх призначення. Загальний алгоритм передбачення часу читання використано та реалізовано в класі Model, що вважається основою всієї програми. Розроблено користувацький інтерфейс та наведено призначення кожного з елементів керування. Програмний застосунок здатний виконувати обробку як документів зазначених форматів, а саме: doc, docx та pdf, а також окремі вебсторінки.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення та умови застосування програми

Точне передбачення часу читання людиною-оператором інформаційних повідомлень та технічної документації дозволяє мінімізувати ризик прийняття запізнених рішень у критичних ситуаціях, що сприяє підвищенню безпеки та зниженню ймовірності виникнення аварій, надзвичайних подій та загроз для життя людей та навколишнього середовища.

Розроблений програмний застосунок дозволить покращити підготовку до проєктування інтерфейсів дизайнерами та когнітологами інформаційних систем з метою збільшення ефективності роботи операторів.

Важливою вимогою забезпечення працездатності системи є наявність компонентів «Microsoft Visual C++ Redistributable Packages». Даний набір компонентів включає деякі важливі DLL-бібліотеки, які використовуватимуться розробленою програмою.

4.2 Інструкція по експлуатації програмного забезпечення

Буде описано спосіб звернення до застосунку, дані, що використовуватимуться і оброблятимуться під час експлуатації застосунку, а також повідомлення, що інформуватимуть користувача щодо стану процесу обробки та аналізу ресурсів, за вказаним користувачем шляхом.

4.2.1 Звернення до програми

Для використання програмного застосунку, необхідно завантажити архів з програмою, після цього запустити виконуваний файл main.exe. Програма має графічний інтерфейс та не містить командного рядку, тому не вимагає задання додаткових аргументів командного рядку.

4.2.2 Вхідні та вихідні дані

Вхідними даними є частотний словник англійської мови, а також перелік адрес шрифтів що встановлені в кожній ОС Windows за замовчуванням. До вхідних даних належать введені користувачські параметри дисплею, та адреса вебсторінки чи файлу для обробки.

Вихідними даними програмного застосунку є новий файл з форматом .PDF у випадку якщо користувач вказав файл .doc чи .docx. Вихідними даними є проміжні дії та результат оцінки часу читання.

4.2.3 Повідомлення системи

Було визначено основні повідомлення системи, що сповіщають користувача про результат кожного з кроків у полі «Intermediate Outcomes». Дані повідомлення наведені нижче, у табл. 4.1.

Таблиця 4.1 – Повідомлення системи

Текст	Пояснення
1	2
Loading frequency dictionary	Завантаження частотного словника
Converting document to PDF	Конвертація документу .docx або .doc в формат .pdf
Next word	Відображення наступного знайденого слова
Font properties	Властивості шрифту слова
Calculating index of the word	Визначення позиції точки приземлення слова
Fixation in word <example> on character 'x' at index 2	Фіксація в слові «example» на символі «x» з позицією в слові 2

Продовження таблиці 4.1

1	2
Probability of refixation: x	Вірогідність рефіксації визначено як x
Refixation required	Рефіксація необхідна
Refixation not required	Рефіксація не потрібна
Refixation delay time: x	Час затримки рефіксації становить: x
Frequency of word <word> per million: x	Поширення слова «word» на мільйон слів
Time required for reading word <word> = x ms	Час x, необхідний для розпізнавання слова «word», виражений в мс
Performing saccade...	Виконання саклади
Moving to the next block... Time required x ms	Пошук наступного текстового блоку, виражений в мс
Word was skipped! Landing on the next character!	Цільове слово було пропущене, око зупинилося на наступному символі після слова
Pronounced: six hundred	Знайдене число аналізується як словосполучення «шість сотень»
Analysis completed	Аналіз завершено
Required: x sec(min). for reading.	Оцінка часу становить x секунд або хвилин
Standard deviation is: x sec(min).	Стандартне відхилення складає x секунд або хвилин

4.3 Приклади експлуатації програмного застосунку

Для демонстрації роботи програми, було створено файл формату .docx, який було автоматично перетворено програмним застосунком до файлу типу .pdf. Після цього програмна модель використала всі наявні алгоритми (пункти 2.1-2.5) та відобразила оцінку часу, необхідного для читання заданого тексту (рис. 4.1).

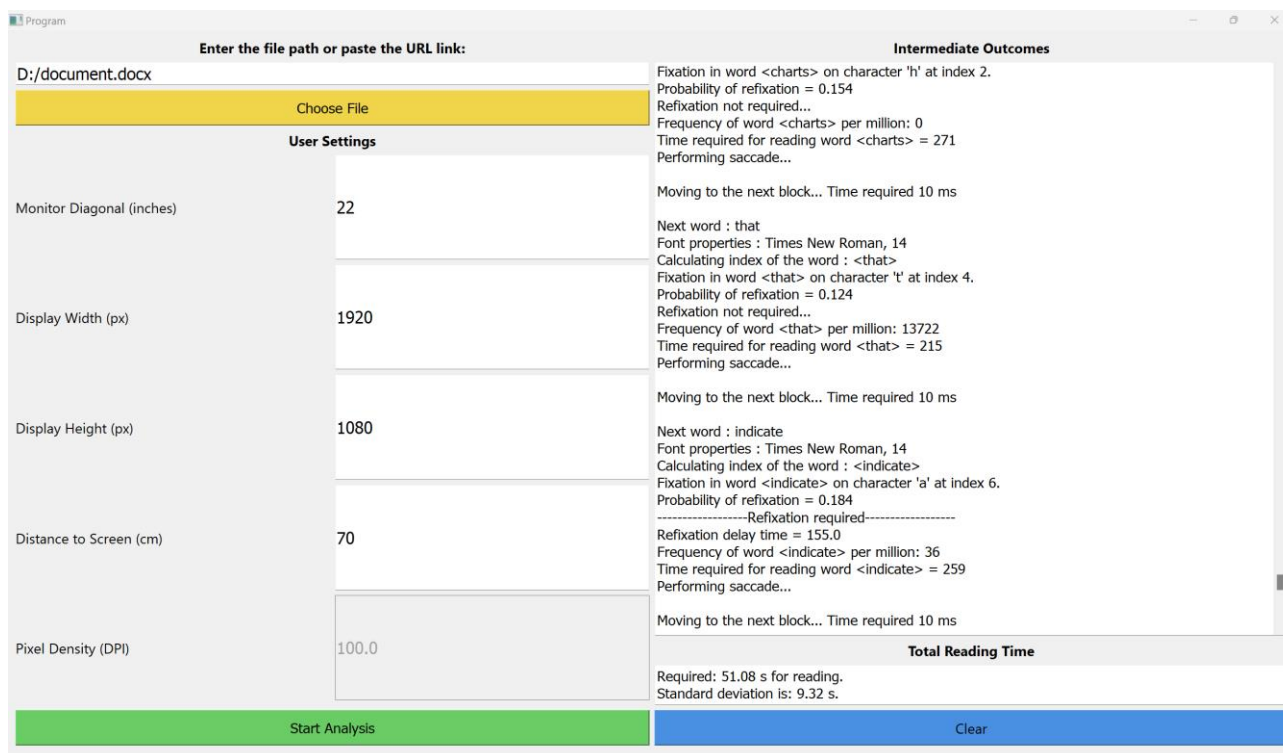


Рисунок 4.1 – Аналіз та обробка docx файлу

Програмний застосунок обраховує щільність пікселів на основі введених параметрів екрану та здійснює аналіз, враховуючи відстань до екрану, що вказав користувач. Поточний стан процесу моделювання відображається у полі "Intermediate Outcomes" та демонструє кожен крок опрацювання тексту.

Як приклад, розглянемо обробку текстового документу document.docx. Для слова «that» було визначено, що рефіксація не є необхідною, тому час розпізнавання слова не потребує повторної фіксації на слові. Для виявленого в ньому слова «indicate» вірогідність рефіксації склала 0.184, тому модель

визначила додатковий час, що буде витрачений на процес рефіксації, що складає 155 мс. Крок «Moving to the next block» означає кінець лексичної ідентифікації поточного слова й перехід до пошуку наступного.

Для знайдених слів було знайдено стиль шрифту «Times New Roman PostScript Medium Type» з розміром кеглю 14, що вважається стандартним для документу (пункт 2.1).

Зауважимо, що для кожного знайденого слова відображається розповсюдженість даного слова на кожен мільйон слів. Так, слово «indicate» зустрічається всього 36 разів на кожен мільйон слів, водночас «that» 13722. Частота слова, як і його довжина, впливає на загальний час лексичної ідентифікації за формулою 2.12.

Загальний час читання складається з точної оцінки та стандартного відхилення. Для даного прикладу оцінка часу, визначеного моделлю складає 51,27 с., а показник стандартного відхилення 9,65 с.

Далі, буде перевірено роботу програмного застосунку на прикладі деякого сайту (рис. 4.2).

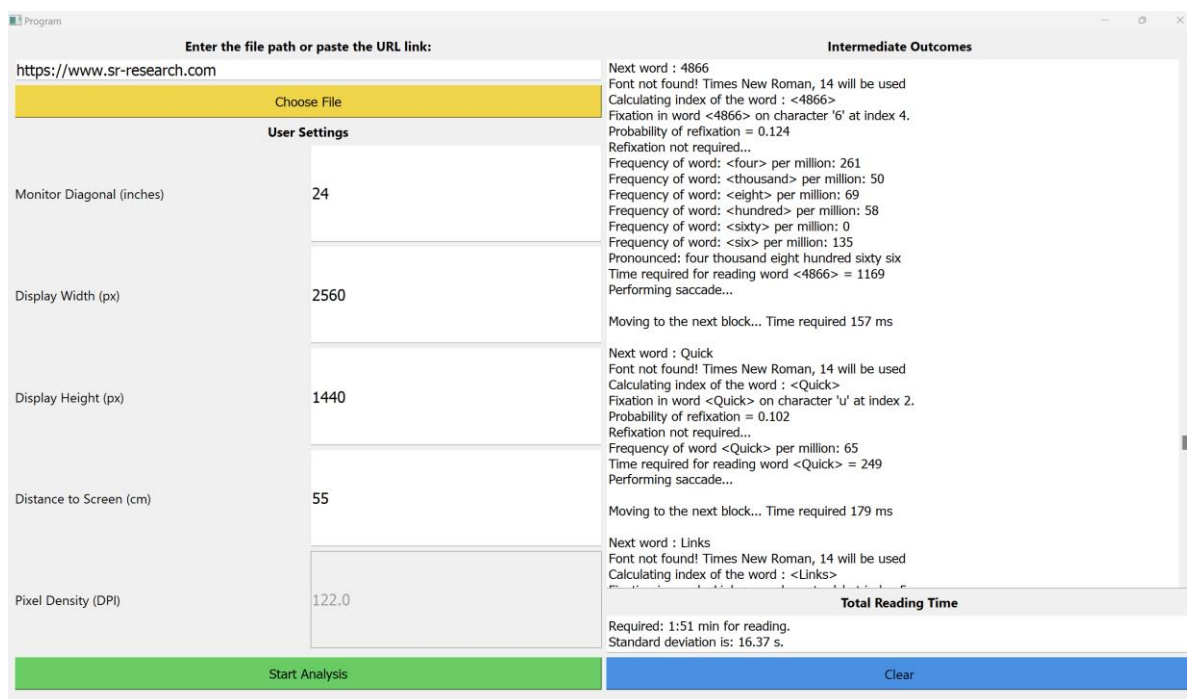


Рисунок 4.2 – Аналіз та обробка вебсторінки

За вказаною адресою програмний застосунок виконує синтаксичний аналіз вебсторінки, після чого починається обробка відокремлених текстових вузлів. В розглянутому прикладі, знайдене число «4866» має більше кроків обробки, так як модель розпізнає число як перелік числівників «чотири», «тисячі», «вісім», «сотень», «шість». Після цього, додатковий текстовий надпис «Pronounce» вказує, яким чином число було представлено моделі. Оцінка часу, необхідного для прочитання всіх текстових надписів на вебсторінці обрахована в 1 хв. та 51 с. Стандартне відхилення, при цьому, складатиме 16,37 с.

4.4 Висновки за розділом 4

Було описано призначення та необхідні вимоги для ефективного застосування програми. Визначено повідомлення системи, що відображають стан процесу моделювання. Продемонстровано роботу моделі на документі, завантаженому з розширенням .docx. Наведено приклад експлуатації програмного застосунку за вказаним посиланням на вебсторінку. Описано особливості роботи моделі з числами.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес читання тексту людиною.

Було розглянуто актуальність дослідження, яка полягає в необхідності передбачення часу, необхідного для читання текстів. Проведено огляд існуючих моделей передбачення часу, серед них: окомоторні, лексичні та змішані стратегії. Виявлено, що кожна з моделей має свої особливості та обмеження, які необхідно враховувати для забезпечення більш точного результату оцінки. Визначено завдання дослідження для реалізації вдосконаленої моделі читання.

Розроблено метод обчислення розмірів літери на основі параметрів шрифту, такі як назва шрифту та його розмір. Розроблено спосіб оцінки часу, необхідного для виконання сакади. Створено алгоритм опрацювання документів та вебсторінок. Описано спосіб розпізнавання та врахування часу, що витрачається на перегляд чисел. Розроблено загальну вдосконалену модель передбачення часу, необхідного для читання. Кожен крок моделі відображає зміну стану процесу моделювання та сповіщає користувача про результат обробки текстового блоку.

Для відображення процесу моделювання було реалізовано програмний застосунок. Розглянуто всі класи, що реалізовані при створенні програмного застосунку, їх основні методи та взаємозв'язки.

Для врахування параметрів дисплею, відносно якого відбуватиметься аналіз документу чи вебсторінки, передбачено відповідні елементи керування для заповнення. Поле «Pixel Density» визначається автоматично, і впливає на визначення розмірів літери відносно роздільної здатності дисплею та діагоналі.

Новизна розробленої моделі передбачення часу читання у тому, що на відміну від існуючих, вона враховує типографічні параметри тексту, параметри відображення (роздільна здатність монітора) та умови читання (відстань до монітору), що впливають на когнітивні процеси людини.

Практичне значення роботи полягає в тому, що розроблено програмний застосунок передбачення часу, необхідного для читання тексту. Це актуально при проєктуванні людино-машинної взаємодії, особливо в умовах обмеженого часу, доступного для прийняття рішень та є важливим фактором успішності дій оператора. Розроблена модель відкриває нові можливості та підходи до моделювання когнітивної діяльності людини.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Reilly, R. G., & O'Regan, J. K. Eye Movement Control During Reading: A Simulation of Some Word-targeting Strategies [Text] / R. G. Reilly, J. K. O'Regan - Kidlington: Pergamon, 1998. – p.303-317.
2. Tarasov, D. A. Vision and reading: monograph [Text] / D. A. Tarasov. – Ekaterinburg: Ural Federal University, 2015. – 76 p.
3. Dubasova, A. V. Eye movements during reading: From general theories to specific [Electronic resource]. – Access mode: <http://www.academia.edu/3074458>.
4. Felici, J. The Complete Manual of Typography [Text] / J. Felici. – Berkeley: Peachpit Press, 2003. – 39 p.
5. Binder, S. K., & Rayner, K. Contextual strength does not modulate the subordinate bias effect: Evidence from eye fixations and self-paced reading [Text] / S. K. Binder, K. Rayner // Psychonomic Bulletin & Review, 1988. – p.271–276.
6. Рух ока під час читання [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Рух_ока_під_час_читання.
7. The Read Time Speed Reading Extension [Electronic resource]. – Access mode: <https://thereadtime.com>.
8. Pixel density [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Pixel_density.
9. Fast calculation method with saccade suppression for a computer-generated hologram based on Fresnel zone plate limitation [Electronic resource]. – Access mode: <https://doi.org/10.1364/OE.390547>.
10. Сердюк, С.М. Ергономічні питання проектування людино-машинних систем [Текст] / С.М. Сердюк — Запоріжжя: ЗНТУ, 2014. — с. 156–162.
11. Kowler, E., Anton, S. Reading twisted text: implications for the role of saccades [Text] / E. Kowler, S. Anton - Kidlington: Pergamon, 1987. – p. 45–60.
12. Rayner K. The span of the effective stimulus during a fixation in reading [Text] / K. Rayner – New York: Cornell University, 1975. – p.3-5.

13. Rayner K. Eye Movements in Reading: Models and Data [Text] / K. Rayner – San Diego: University of California, 2009. – p.1-10.
14. Rayner K., Well A.D. Effects of contextual constraint on eye movements in reading: A further examination [Text] / K. Rayner, A.D. Well – Massachusetts: University of Massachusetts, 1996. – p. 504–509.
15. McConkie G. W., Rayner K. The span of the effective stimulus during a fixation in reading. Perception and Psychophysics [Text] /G. McConkie, K. Rayner – New York: Springer 1975. – Vol. 17. p. 578–586.
16. McConkie, G. W., Kerr, P. W., Reddix, M. D., Zola, D. Jacobs, A. M., Eye movement control during reading: II. Frequency of refixating a word. Perception and Psychophysics [Text] / G.W. McConkie, P.W. Kerr, M.D. Zola, D. Jacobs – Paris: Centre Nationale de la Recherche Scientifique, 1989. – p. 245-253.
17. Bringhurst R. The elements of typographic style [Text] / R. Bringhurst. – Vancouver: Hartley & Marks, 1992. – p.43-45.
18. Visual Angle – Fast, Accurate, Reliable Eye Tracking [Electronic resource]. – Access mode: <https://www.sr-research.com/eye-tracking-blog/background/visual-angle>.
19. English Corpora: most widely used online corpora. Billions of words of data [Electronic resource]. – Access mode: <https://www.english-corpora.org/resources.asp>.
20. Buari, N. H., & Md-Isa, A. N. F. Eye Movements Behaviour in Reading Different Text Sizes among University Students [Text] / N.H. Buari, Md-Isa, A.N.F., – UK: Environment-Behaviour Proceedings Journal, 2019. – p. 75-80.

ДОДАТОК А
Технічне завдання

Вступ

У роботі розглядається галузь розробки програмного забезпечення, пов'язана з дослідженням та програмною реалізацією методу оцінки часу читання з врахуванням типографічних особливостей тексту.

А.1. Підстава для розробки

Підставою для розробки є завдання на дипломний проєкт на тему «Дослідження та програмна реалізація методів передбачення часу читання», затверджене наказом по національному університету «Запорізька політехніка», наказ № 398 від 07 жовтня 2024 р.

А.2. Призначення розробки

Програмний застосунок призначений для оцінки передбачення часу, необхідного для читання тексту. Реалізована програма повинна відображати стан процесу моделювання читання. Програмний застосунок повинен забезпечити користувачі налаштування параметри дисплею, що використовується.

А.3. Основні вимоги до програмного забезпечення, що розробляється

А.3.1 Вимоги до функціональних характеристик

До загальних функціональних вимог слід віднести:

- врахування положення всіх фрагментів тексту на сторінці документу чи сайту шляхом визначення додаткового часу, що потребує користувач для виконання сакади до тексту, що читається;

- можливість введення розміру дисплею та його роздільної здатності користувачем під час роботи з програмою;
- обчислення та подальше зберігання фізичних розмірів літери залежно від характеристик дисплею, вказаних користувачем;
- обробка вказаного користувачем документу або вебсторінки;
- надання графічного інтерфейсу користувачу для зручності використання програмного рішення;
- автоматичне перетворення файлів формату .docx та .doc до формату PDF зі збереженням всіх властивостей документу;
- визначення впливу особливостей різних комбінацій шрифтів та їх кеглів на процес читання;
- використання встановлених шрифтів на ОС Windows в директорії «C:\Windows\Fonts» для розрахунку реальних розмірів літер;
- результат виконання алгоритму має бути приведений до часу згідно одиниць Сі та поданий, враховуючи стандартне відхилення кінцевої оцінки часу, якщо таке буде.

А.3.2 Вимоги до надійності

Програмний застосунок повинен містити достатньо перевірок на виявлення критичних помилок та збійних ситуацій, при цьому, забезпечувати коректну подальшу роботи програми. Необхідно забезпечити неушкодженість користувацьких документів, що оброблятимуться.

А.3.3 Умови експлуатації

Під час експлуатації програмного забезпечення важливо забезпечити безперебійне енергопостачання комп'ютера та підключення до мережі Інтернет.

Обов'язковою вимогою є наявність компонентів «Microsoft Visual C++ Redistributable Packages». Даний набір компонентів включає DLL-бібліотеки, які використовуватимуться програмою.

А.3.4 Вимоги до складу і параметрів технічних засобів

Мінімальними вимогами до апаратного забезпечення є встановлена ОС Windows 10 або новіша. ПК користувача повинно мати процесор з тактовою частотою 1 ГГц та вище, 1 Гб вільного місця на жорсткому диску та оперативний запам'ятовуючий пристрій з об'ємом 4 Гб. Для повноцінного використання програмного застосунку, передбачається встановлення браузеру Google Chrome.

А.3.5 Вимоги до програмної документації

Система має бути поставлена разом із технічним завданням та інструкцією з експлуатації, наданою у відповідному розділі пояснювальної записки.

А.4 Порядок контролю та приймання

Строки виконання дипломної кваліфікаційної роботи магістра визначені в завданні на роботу безпосередньо, включаючи строк подання роботи, календарний план виконання роботи з зазначеними етапами, результатами та строками. Захист роботи відбувається за оголошеним завчасно графіком захисту дипломних кваліфікаційних робіт і відповідно графіком роботи екзаменаційної комісії.

ДОДАТОК Б
Текст програми

Б1. Текст файлу Model.py

```

import math
import random
import urllib
import numpy as np
from ParserWeb import ParserWeb
from pdf_parser import ParserPDF
import re
from docx2pdf import convert
import os

class Model:

    list_text_spans = []
    average_saccade_latency = 150
    standard_deviation_latency = 50
    distance_to_display = 0
    width_px = 0
    height_px = 0
    DPI = 0
    diagonal_inches = 0
    path = ""
    PPI = 0

    def __init__(self):
        self.__full_time_to_read = 0
        self.__full_time_standard_deviation = 0
        self.parserPDF = ParserPDF()
        self.parserWeb = ParserWeb()

    def reset_results(self):
        self.__full_time_to_read = 0
        self.__full_time_standard_deviation = 0

```

```

def read_text_from_pdf(self):
    self.reset_results()
    if not os.path.isfile(self.path):
        raise FileNotFoundError
        (f"The file {self.path} does not exist.")
    self.list_text_spans = self.parserPDF.start(self.path)
def read_text_from_site(self, url):
    self.reset_results()
    self.list_text_spans = self.parserWeb.parse_webpage(url)

def set_distance_to_display(self, distance_in_cm):
    self.distance_to_display = distance_in_cm

def calculate_ppi(self):
    diagonal_pixels = math.sqrt((self.width_px *
        self.width_px + self.height_px * self.height_px))
    self.PPI = diagonal_pixels / self.diagonal_inches

def set_diagonal_inches(self, size_in_inches):
    self.diagonal_inches = size_in_inches

def set_width_px(self, width):
    self.width_px = width

def set_height_px(self, height):
    self.height_px = height

def set_path(self, user_path):
    self.path = user_path

def split_string(self, text):
    pattern = r'\w+|\s*[^w\s]+\s*|\s+'
    split_text = re.findall(pattern, text)
    return split_text

```

```

def get_text_list_spans(self):
    return self.list_text_spans

def add_average_latency_time(self):
    self.__full_time_to_read += self.average_saccade_latency

def calculate_average_landing_position(self, word, d, k):
    m = 3.3 + 0.49 * d * k
    return m

def calculate_standard_deviation(self, d, word):
    sd = 1.318 + 0.000518 * d ** 3
    return sd

def calculate_probability_letter_landing(self, x, m, sd):
    q = math.sqrt(2 * math.pi) * sd
    ch = (x - m) ** 2
    zn = 2 * (sd * sd)
    prob = 1 / q * math.exp(-ch / zn)
    return prob

def calculate_deviation_x(self, word, position):
    opt_position = len(word) / 2 - 0.5
    deviation = abs(opt_position - position)
    return deviation

def calculate_sd(self, m):
    return 0.1 * m

def should_refixate(self, probability_refixation):
    return random.random() < probability_refixation

def calculate_normal_distribution(self, m, sd):
    return np.random.normal(m, sd)

```

Б.2 Текст файла Pdf_Parser.py

```

import fitz
import re
from TextSpan import TextSpanPDF
from PIL import Image
import io

class ParserPDF:
    def __init__(self):
        self.list_spans = []

    def flags_decomposer(self, flags):
        l = []
        if flags & 2 ** 0:
            l.append("superscript")
        if flags & 2 ** 1:
            l.append("italic")
        if flags & 2 ** 2:
            l.append("serifed")
        else:
            l.append("sans")
        if flags & 2 ** 3:
            l.append("monospaced")
        else:
            l.append("proportional")
        if flags & 2 ** 4:
            l.append("bold")
        return ", ".join(l)

    def points_to_cm(self, points):
        inches = points / 72.0
        cm = inches * 2.54
        return cm

```

```

def calculate_distance(self, coords1, coords2):
    _, _, x1, y1 = coords1
    x2, y2, _, _ = coords2
    distance = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
    return self.points_to_cm(distance)

def convert_color_to_hex(self, color):
    if isinstance(color, tuple):
        return "%02x%02x%02x" % tuple(int(c) for c in color)
    elif isinstance(color, int):
        return "%02x%02x%02x" % ((color >> 16) &
                                   0xFF, (color >> 8) & 0xFF, color & 0xFF)
    return "#000000"

def append_text_span(self, text, bbox,
font, size, color, bgcolor, flags, previous_coords):
    text_span = TextSpanPDF()
    text_span.set_text(text)
    text_span.set_coords(bbox[0], bbox[2], bbox[1], bbox[3])
    text_span.set_font(font)
    text_span.set_size_text(int(size))
    text_span.set_color_text(color)
    text_span.set_background_color(bgcolor)
    text_span.set_flags(flags)

    if previous_coords:
        distance = self.calculate_distance
            (previous_coords, bbox)
        self.list_spans[-1]
            .set_distance_to_next_span(distance)
    else:
        text_span.set_distance_to_next_span(0)

    self.list_spans.append(text_span)

```

Б.3 Текст файлу Utils.py

```

from openpyxl import load_workbook
from PyQt5.QtCore import QThread, pyqtSignal
import re

class FrequencyDictionary(QThread):
    finished = pyqtSignal()

    def __init__(self):
        super().__init__()
        self.workbook = None
        self.sheet = None
        self.column = None

        self.ones = [
            "zero", "one", "two", "three", "four", "five",
            "six", "seven", "eight", "nine"
        ]
        self.teens = [
            "ten", "eleven", "twelve", "thirteen",
            "fourteen", "fifteen", "sixteen",
            "seventeen", "eighteen", "nineteen"
        ]
        self.tens = [
            "", "", "twenty", "thirty", "forty", "fifty",
            "sixty", "seventy", "eighty", "ninety"
        ]
        self.thousands = ["", "thousand"]

    def run(self):
        self.load_dictionary()

```

```

def load_dictionary(self):
    self.workbook = self.get_frequency_dictionary
        ('data/wordFrequency.xlsx')
    self.sheet = self.get_sheet_workbook
        (self.workbook, "4 forms (219k)")
    self.column = self.get_column_by_sheet('B')
    self.finished.emit()

def get_frequency_dictionary(self, path):
    return load_workbook(path)

def get_sheet_workbook(self, workbook, sheet):
    list_sheets = workbook.sheetnames
    if sheet in list_sheets:
        return workbook[sheet]
    else:
        return None

def find_freq_for_word(self, sheet, column, word):
    for cell in column:
        if word == cell.value:
            return sheet.cell(row=cell.row, column=15).value
    return 0

def get_biggest_frequency(self):
    return self.sheet.cell(row=2, column=15).value

def get_column_by_sheet(self, col):
    return self.sheet[col]

def parse_hundreds(self, num):
    if num == 0:
        return ""
    if num < 10:

```

```

        return self.ones[num]
    elif num < 20:
        return self.teens[num - 10]
    else:
        result = self.tens[num // 10]
        if num % 10 != 0:
            result += " " + self.ones[num % 10]
        return result

def number_to_words(self, n):
    if n < 0:
        return "negative"

    if n <= 9999:
        words = []
        if n >= 1000:
            words.append(self.ones[n // 1000]
                          + " " + self.thousands[1])
            n %= 1000
        if n >= 100:
            words.append(self.ones[n // 100] + " hundred")
            n %= 100
        if n > 0:
            words.append(self.parse_hundreds(n))

        return " ".join(words).strip()

    else:
        words = []
        for digit in str(n):
            words.append(self.ones[int(digit)])
        return " ".join(words)

```

Б.4 Текст програми ParserWeb.py

```

import logging
import re
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.common.exceptions
import StaleElementReferenceException, JavascriptException
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service
from TextSpan import TextSpanPDF

logging.basicConfig(
    level=logging.DEBUG,
    filename='parser_logs.txt',
    filemode='a',
    format='%(asctime)s - %(levelname)s - %(message)s' )

class ParserWeb:
    def __init__(self):
        self.list_spans = []

    def parse_webpage(self, url):
        options = webdriver.ChromeOptions()
        options.add_argument("--headless")
        options.add_argument("--disable-gpu")
        options.add_argument("--no-sandbox")
        options.add_argument("--disable-dev-shm-usage")
        options.add_argument("--disable-software-rasterizer")
        logging.info(f"Opening URL: {url}")
        try:
            driver
            =
webdriver.Chrome(service=Service(ChromeDriverManager().install()),
options=options)

```



```

spanElement.textContent = part;

spanContainer.appendChild(spanElement);
                                }

    else {
        const textNode =
            document.createTextNode(part);

spanContainer.appendChild(textNode);
                                }
    });
    element.replaceChild(spanContainer,
child);
                                }    else    if    (child.nodeType    ===
Node.ELEMENT_NODE && !ignoredTags.includes(child.tagName) && !isHidden) {
        wrapTextNodesInSpan(child);
    }
    });
}
wrapTextNodesInSpan(document.body);
""")
except JavascriptException as e:
    logging.error(f"JavaScript execution error: {e}")

elements = driver.find_elements(By.CLASS_NAME, "word")
previous_coords = None

for elem in elements:
    try:
        text = elem.text.strip()

        if not text or all(char in {'.', ',', '-', ' ',
'<', '>'} for char in text):
            continue

```

```

split_text = re.split(r'[.,\-\s<>]+', text)
split_text = [part for part in split_text if part]

    for part in split_text:
        if re.match(r'^[a-zA-Za-яA-ЯіієґІІЄґ0-9@]+$',
            , part): # Тільки текст, числа та знак @
                style = self.get_style_properties(driver,
elem)

                    font_size =
self.extract_font_size(style.get("fontSize", "12px"))
                    font_family =
self.get_active_font_family(driver, elem)
                    color = style.get("color", "black")
                    x1, y1, x2, y2 =
self.get_element_coordinates(driver, elem)

                    word_span = TextSpanPDF()
                    word_span.set_text(part) # Зберігаємо
оригінальний текст, включаючи @
                    word_span.set_size_text(int(font_size))
                    word_span.set_background_color("")
                    word_span.set_color_text(color)
                    word_span.set_coords(x1, y1, x2, y2)
                    word_span.set_flags("")

                    # Вираховуємо відстань до наступного
елемента

                    if previous_coords:
                        distance =
self.get_distance_to_next_element(previous_coords, (x1, y1))
                        word_span.distance_to_next_span =
distance

                    # Додаємо до списку

```

```

        self.list_spans.append(word_span)
        previous_coords = (x2, y2)

        logging.debug(
            f"Processed text: {part}, Distance to
next: {getattr(word_span, 'distance_to_next_span', 'N/A')}")

        except StaleElementReferenceException as e:
            logging.error(f"StaleElementReferenceException
caught: {e}, skipping this element")
            continue
        except JavascriptException as e:
            logging.error(f"JavascriptException caught: {e},
skipping this element")
            continue
        except Exception as e:
            logging.error(f"Unexpected error caught: {e},
skipping this element")
            continue

    driver.quit()
    logging.info("Finished parsing webpage.")
    return self.list_spans

def clean_html_tags(self, text):
    clean = re.compile('<.*?>')
    return re.sub(clean, '', text)

def get_active_font_family(self, driver, element):
    try:
        script = r"""
        const elem = arguments[0];
        const style = window.getComputedStyle(elem);
        const fontFamilyList = style.fontFamily.split(',');

```

```

        for (const font of fontFamilyList) {
            const fontName = font.trim().replace(/['"]/g, '');
            if (document.fonts.check(`1em ${fontName}`)) {
                return fontName;
            }
        }
        return
style.fontFamily.split(',')[0].trim().replace(/['"]/g, '');
"""
        return driver.execute_script(script, element)
except JavascriptException:
    return "default"

def get_style_properties(self, driver, element):
    try:
        script = r"""
        const elem = arguments[0];
        const style = window.getComputedStyle(elem);
        return {
            fontSize: style.fontSize || "12px",
            fontWeight: style.fontWeight || "normal",
            fontStyle: style.fontStyle || "normal",
            color: style.color || "black",
            fontFamily: style.fontFamily || "default"
        };
        """
        return driver.execute_script(script, element)
    except JavascriptException:
        return {"fontSize": "12px", "fontWeight": "normal",
"fontStyle": "normal", "color": "black",
            "fontFamily": "default"}

def extract_font_size(self, font_size_str):
    match = re.match(r"(\d+(\.\d+)?)px", font_size_str)

```

```

        if match:
            return float(match.group(1))
        return 12.0

def get_element_coordinates(self, driver, element):
    try:
        script = r"""
            const rect = arguments[0].getBoundingClientRect();
            return [rect.left, rect.top, rect.right,
rect.bottom];
        """
        return driver.execute_script(script, element)
    except JavascriptException:
        return [0, 0, 0, 0]

def get_distance_to_next_element(self, prev_coords,
curr_coords):
    prev_x, prev_y = prev_coords
    curr_x, curr_y = curr_coords
    return ((curr_x - prev_x) ** 2 + (curr_y - prev_y) ** 2)
** 0.5

```

Б.5 Текст файлу FontsManager.py

```
import logging
from PIL import ImageFont
from jsonhandler import JSONHandler

# Налаштування логування
logging.basicConfig(
    level=logging.DEBUG,
    filename='fonts_manager_logs.txt',
    filemode='a',
    format='%(asctime)s - %(levelname)s - %(message)s')

class FontsManager:
    def __init__(self):
        self.dict_names_path = {}
        self.json_handler = JSONHandler()
        self.path = "FontsPath.txt"
        self.alphabet = "abcdefghijklmnopqrstuvwxyz"
        logging.info("Initializing FontsManager...")
        self.read_fonts_from_file()
        self.calculate_default_font_and_size()

    def parse_fonts(self, font_data):
        for line in font_data:
            if 'C:\\' in line:
                parts = line.split('C:\\', 1)
                font_name = parts[0].strip()
                font_path = 'C:\\' + parts[1].strip()
                self.dict_names_path[font_name] = font_path
                logging.debug(f"Parsed font: {font_name},
                    Path: {font_path}")
```

```

        else:
            logging.warning(f"Cannot parse the string:
{line}")

def read_fonts_from_file(self):
    try:
        logging.info(f"Reading fonts from file: {self.path}")
        with open(self.path, 'r', encoding='utf-8') as file:
            lines = file.readlines()
            self.parse_fonts(lines)
    except FileNotFoundError:
        logging.error(f"The file {self.path} was not found!")
    except Exception as e:
        logging.error(f"Error reading fonts file: {e}")

def calculate_default_font_and_size(self):
    self.json_handler.create_file_if_does_not_exist()
    if self.json_handler.find_combination_font
        ("Times New Roman", 14, 100):
    else:
        logging.info("Calculating default font and size...")
        w_basic, h_basic = self.
get_average_size_letters_font_and_size("Times New Roman", 14, 100)
        self.json_handler.save_new
            ("Times New Roman", 14, w_basic, h_basic, 100)

def get_coefficient_font_letter(self,
    font_name, font_size, new_dpi):
    try:
        standard_times_14
=
self.json_handler.find_combination_font("Times New Roman", 14, 100)

        standard_w = standard_times_14["size_width"]
        standard_h = standard_times_14["size_height"]

```

```

        found_combination =
self.json_handler.find_combination_font(font_name, font_size, new_dpi)

        if found_combination:
            found_w = found_combination["size_width"]
            found_h = found_combination["size_height"]
            logging.debug(f"Found font in JSON:
                        {found_combination}")

            if found_combination["font_name"] ==
standard_times_14["font_name"] and found_combination["font_size"]
== standard_times_14["font_size"]:
                if found_combination["dpi"] ==
standard_times_14["dpi"]:
                    logging.debug("Font matches default
configuration. Coefficient=1")
                    return 1
                else:
                    standard_average_size = (standard_w + standard_h) / 2
                    found_average_size = (found_w + found_h) / 2
                    cf = standard_times_14["dpi"] / found_combination["dpi"]
                    standard_average_size *= cf
                    coefficient = found_average_size / standard_average_size

            return coefficient

    def get_average_size_letters_font_and_size(self, new_font,
new_size, new_dpi):
        total_w_cm = 0
        total_h_cm = 0
        for letter in self.alphabet:
            try:

```

```

        w_letter, h_letter =
self.get_size_letter_into_cm(letter, new_font, new_size, new_dpi)
        total_w_cm += w_letter
        total_h_cm += h_letter
    except Exception as e:
        logging.error(f"Error calculating size for letter
'{letter}': {e}")
        average_w_letter = total_w_cm / len(self.alphabet)
        average_h_letter = total_h_cm / len(self.alphabet)
        logging.debug(f"Average letter size:
Width={average_w_letter}, Height={average_h_letter}")
        return average_w_letter, average_h_letter

def get_size_letter_into_cm(self, letter, font, size, dpi):
    width_px, height_px = self.calculate_size_letter(letter,
font, size)
    width_cm = (width_px / dpi) * 2.54
    height_cm = (height_px / dpi) * 2.54
    return width_cm, height_cm

def calculate_size_letter(self, letter, font, size):
    try:
        if size is None or size <= 0:
            size = 14
        if not font:
            font = "Times New Roman"
        if font not in self.dict_names_path:
            logging.warning(f"Font {font} not found,
defaulting to 'Times New Roman'.")
            font = "Times New Roman"
        complex_path = self.dict_names_path[font]

        if not letter:
            raise ValueError("The symbol was not found.")

```

ДОДАТОК В
Слайди презентації

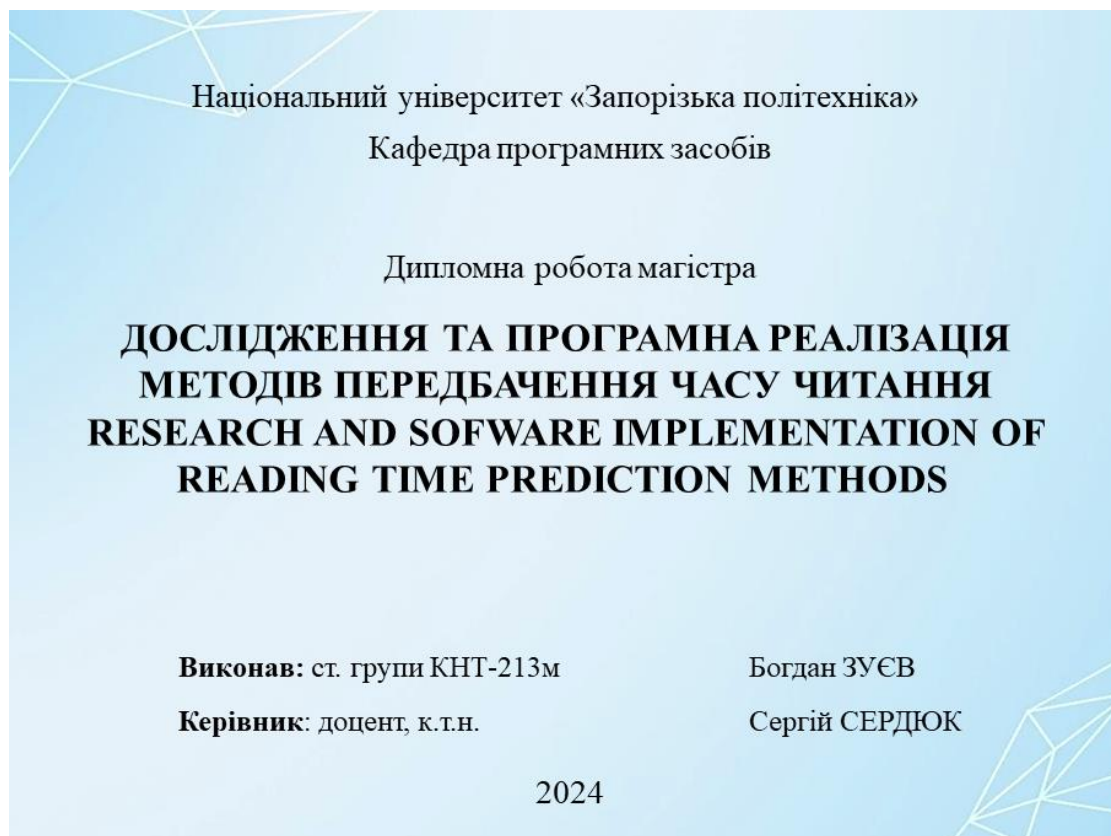


Рисунок В.1 – Титульний слайд

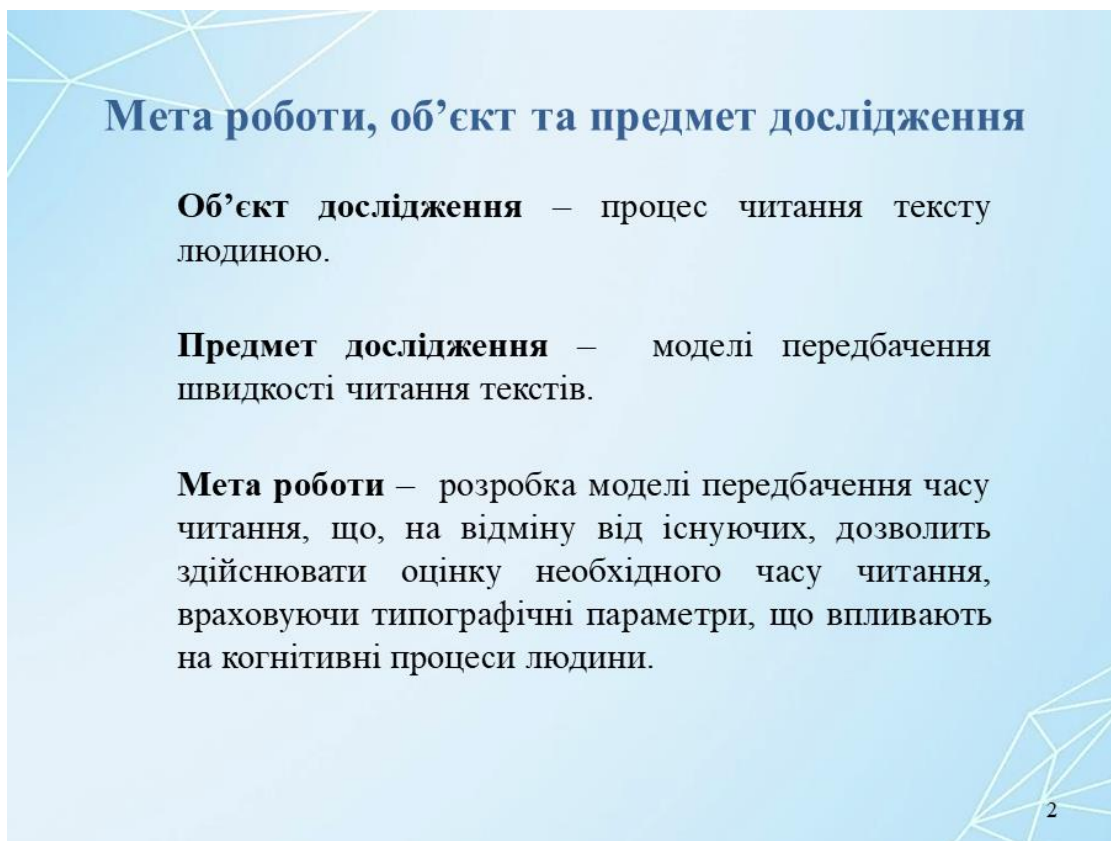


Рисунок В.2 – Мета роботи, об'єкт та предмет дослідження

Завдання до роботи

- аналіз предметної області;
- аналіз існуючих моделей передбачення часу;
- розробка технічного завдання;
- розробка вдосконаленої моделі читання;
- розробка алгоритмів опрацювання документів та вебсторінок;
- розробка способу оцінки часу виконання сакади;
- проєктування програмного забезпечення;
- реалізація програмного забезпечення;

3

Рисунок В.3 – Завдання до роботи

Аналіз програмних аналогів

Назва	Words To Time	Модель «Зміщення уваги»	Програмний застосунок
Оцінка часу на основі документів	Так	Ні	Так
Оцінка часу перегляду веб-сторінки	Ні	Ні	Так
Врахування положення тексту	Ні	Ні	Так
Обробка тексту, написаного різними мовами	Так	Так	Ні
Врахування ймовірного відхилення результатів	Ні	Так	Так
Врахування впливу шрифтів	Ні	Ні	Так

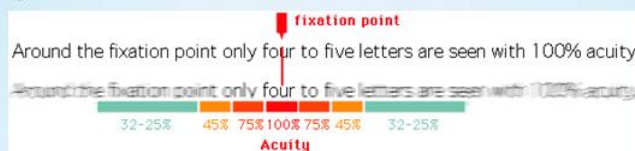
4

Рисунок В.4 – Аналіз програмних аналогів

Алгоритм опрацювання документів

Крок 1. Відкриття необхідного файлу.

Крок 2. Знаходження найближчого слова.



Крок 3. Обчислення вірогідності рефіксації на слові:

$$y = a + 0.03^2$$

Крок 4. Якщо визнано необхідність рефіксації, визначити середню затримку:

$$m = base + range \left[1 - \frac{|loc - middle|}{(length/2)} \right]$$

Крок 5. Визначити час лексичної ідентифікації та повернутися до кроку 2.

$$m' = base5 + 15(length - 5) + 40(1 - freq)$$

5

Рисунок В.5 – Алгоритм опрацювання документів

Алгоритм опрацювання вебсторінок

Крок 1. Завантаження вказаної користувачем веб-сторінки.

Крок 2. Пошук наступного існуючого вузла.

Крок 3. Формування об'єкту, що представляє текстовий вузол.

Крок 4. Використати адаптовану формулу для врахування впливу шрифту:

$$m = 3.3 + 0.49d * k$$

Крок 5. Визначення часу лексичної ідентифікації слова:

$$m' = base5 + 15(length - 5) + 40(1 - freq)$$

Крок 6. Визначення часу, необхідного для сакади та повернення до кроку 2.

$$T = 2.2 * (A - 5) + 21 (ms)$$

Крок 7. Отримання оцінки часу читання.

6

Рисунок В.6 – Алгоритм опрацювання вебсторінок

Діаграма класів програмного застосунку

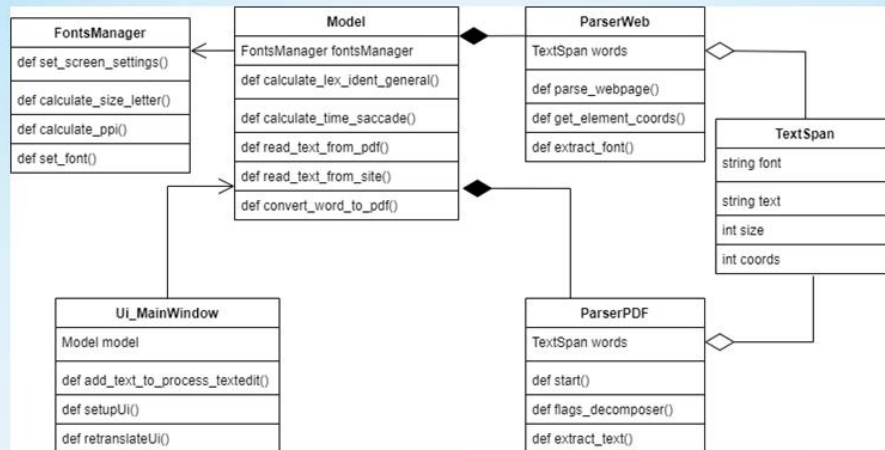


Рисунок В.7 – Діаграма класів програмного застосунку

Основні повідомлення системи

Текст	Пояснення
Fixation in word <example> on character 'x' at index 2	Фіксація в слові «example» на символі «x» з позицією в слові 2
Probability of refixation: x	Вірогідність рефіксації визначено як x
Time required for reading word <example> = x ms	Час x, необхідний для розпізнавання слова «example», виражений в мс
Pronounced: six hundred	Знайдене число аналізується як словосполучення «шість сотень»
Required: x sec(min). for reading	Оцінка часу становить x секунд або хвилин
Standard deviation is: x sec(min)	Стандартне відхилення складає x секунд або хвилин

Рисунок В.8 – Основні повідомлення системи

Результати аналізу документа

Program

Enter the file path or paste the URL link:
D:/document.docx

Choose File

User Settings

Monitor Diagonal (inches)	22
Display Width (px)	1920
Display Height (px)	1080
Distance to Screen (cm)	70
Pixel Density (DPI)	100.0

Start Analysis

Clear

Intermediate Outcomes

Fixation in word <charts> on character 'h' at index 2.
Probability of refixation = 0.154
Refixation not required...
Frequency of word <charts> per million: 0
Time required for reading word <charts> = 271
Performing saccade...

Moving to the next block... Time required 10 ms

Next word : that
Font properties : Times New Roman, 14
Calculating index of the word : <that>
Fixation in word <that> on character 't' at index 4.
Probability of refixation = 0.124
Refixation not required...
Frequency of word <that> per million: 13722
Time required for reading word <that> = 215
Performing saccade...

Moving to the next block... Time required 10 ms

Next word : indicate
Font properties : Times New Roman, 14
Calculating index of the word : <indicate>
Fixation in word <indicate> on character 'a' at index 6.
Probability of refixation = 0.184
Refixation required-----
Refixation delay time = 155.0
Frequency of word <indicate> per million: 36
Time required for reading word <indicate> = 259
Performing saccade...

Moving to the next block... Time required 10 ms

Total Reading Time

Required: 51.08 s for reading.
Standard deviation is: 9.32 s.

9

Рисунок В.9 – Результати аналізу документа

Результати аналізу вебсторінки

Program

Enter the file path or paste the URL link:
<https://www.sr-research.com>

Choose File

User Settings

Monitor Diagonal (inches)	24
Display Width (px)	2560
Display Height (px)	1440
Distance to Screen (cm)	55
Pixel Density (DPI)	122.0

Start Analysis

Clear

Intermediate Outcomes

Next word : 4866
Font not found! Times New Roman, 14 will be used
Calculating index of the word : <4866>
Fixation in word <4866> on character '6' at index 4.
Probability of refixation = 0.124
Refixation not required...
Frequency of word: <four> per million: 261
Frequency of word: <thousand> per million: 50
Frequency of word: <eight> per million: 69
Frequency of word: <hundred> per million: 58
Frequency of word: <sixty> per million: 0
Frequency of word: <six> per million: 135
Pronounced: four thousand eight hundred sixty six
Time required for reading word <4866> = 1169
Performing saccade...

Moving to the next block... Time required 157 ms

Next word : Quick
Font not found! Times New Roman, 14 will be used
Calculating index of the word : <Quick>
Fixation in word <Quick> on character 'u' at index 2.
Probability of refixation = 0.102
Refixation not required...
Frequency of word <Quick> per million: 65
Time required for reading word <Quick> = 249
Performing saccade...

Moving to the next block... Time required 179 ms

Next word : Links
Font not found! Times New Roman, 14 will be used
Calculating index of the word : <Links>

Total Reading Time

Required: 1:51 min for reading.
Standard deviation is: 16.37 s.

10

Рисунок В.10 – Результати аналізу вебсторінки

Висновки

- проведено аналіз предметної області;
- проведено аналіз існуючих моделей передбачення часу;
- розроблено технічне завдання;
- розроблено вдосконалену модель читання;
- розроблено алгоритми опрацювання документів та вебсторінок;
- розроблено спосіб оцінки часу виконання сакади;
- виконано проєктування програмного забезпечення;
- реалізовано програмного забезпечення;

11

Рисунок В.11 - Висновки