UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

TITLE

Absolvent
Hodivoianu Anamaria

Coordonator științific
Titlul și numele profesorului coordonatorului

București, iunie 2025

Abstract

jdfhnjencweicn

Contents

Chapter 1

Introduction

Chapter 2

Eye Tracking

2.1 What is Eye Tracking?

2.2 Eye Tracking and AI

Chapter 3

Data Collection

3.1 The MultiplEYE Project

3.1.1 Overview

The MultiplEYE project is a research initiative that aims to bring together researchers from various fields working with eye-tracking data in multiple languages. The name MultiplEYE is a play on the words "multiple" and "eye", reflecting the project's focus on eye-tracking data in multiple languages. One of the main goals of the project is to create a large multilingual eye-tracking dataset that can be used to study human reading from a psycholinguistic perspective and to improve various natural language processing tasks.

3.1.2 Experiment Design

The MultiplEYE experiment is designed to collect eye-tracking data from participants reading a set of texts in their native language. The experiment consists of two main parts: a reading task and a comprehension task. The participant sits in a chair at a desk with their chin and forehead resting on a support platform to prevent them from moving their head. On the desk there is a monitor displaying the text the participant has to read. Below the monitor, the eye-tracking device is tracking their eye movements.

3.2 Stimuli

In total, there are 13 texts in the main experiment, 2 practice texts, and 2 back-up texts. After reading each text stimulus, participants are asked to answer some comprehension questions. At the end of the experiment, participants have to complete a questionnaire regarding basic information about themselves and their reading habits.

The texts are selected from a variety of genres: popular science (PopSci), literary (Lit), argumentative (Arg), institutional (Ins), and encyclopedic (Enc). The main

13 texts are: 1 - PopSci_MultiplEYE, 2 - Ins_HumanRights, 3 - Ins_LearningMobility, 4 - Lit_Alchemist, 6 - Lit_MagicMountain, 8 - Lit_Solaris, 9 - Lit_BrokenApril, 10 - Arg_PISACowsMilk, 12 - Arg_PISARapaNui, and 13 - PopSci_Caveman. The 2 practice texts are: 7 - Lit_NorthWind, and 11 - Enc_WikiMoon. The 2 back-up texts are: 14 - Lit_HarryPotter, and 5 - Lit_EmperorClothes.

The texts used in the MultiplEYE experiment are originally in English, and so if someone wants to run the experiment and collect eye-tracking data in another language, they have to translate the texts. The project provides a set of guidelines for the translation process.

For the Romanian translation, some of the texts were replaced with an already existing Romanian translation, for example the texts that were taken from books, while others were translated by Sergiu Nisioi using machine translation. I then edited the translations to make them more fluent and natural. The comprehension questions were also initially translated using machine translation, but needed significant editing.

After translating the stimuli, in order to run the experiment, images with the texts and questions have to be generated. I used the *MultiplEYE Image Generation Software* [2] to generate the images and other necessary files for the experiment.

3.3 Running the Experiment

In order to run the experiment, the MultiplEYE project provides a software package *MultiplEYE Experiment Implementation* [4].

An eye-tracking device is required to run the experiment. The University of Bucharest has an EyeLink 1000 Plus eye-tracker at the Faculty of Psychology and Educational Sciences, which I used to run the experiment. The eye-tracker is connected to a computer that runs the MultiplEYE Experiment Implementation software. The participant sits in front of the eye-tracker and reads the texts displayed on the monitor. The eye-tracker records their eye movements while they read.

The experiment was run with 4 participants, all of them native Romanian speakers. The experiment took about 1-2 hours for each participant to complete. All participants were volunteers and were informed about the purpose of the experiment and how their data would be used.

3.4 Pymovements

The MultiplEYE project uses the *Pymovements* library [3] for extracting eye-tracking data from the raw data collected by the eye-tracking device. First, I convert the raw data file from the EyeLink 1000 Plus eye-tracker from the EDF format to ASCII using the

EDF2ASC tool provided by the eye-tracker manufacturer. The resulting ASCII file contains the raw eye-tracking data, which is then processed using the *Pymovements* library, which calculates the fixations, saccades, and other eye movement events from the raw data. Then, the fixations are mapped to the text stimuli in order to obtain the reading times for each word in the text.

The eye-tracking experiment records the eye movements both during the reading task and during the comprehension task. Since I want to focus on ppredicting the total reading time for words during normal reading, I only keep the data from the reading task.

3.5 Mapping Words to Sentences

Since for some of the features I want to extract I need to know the sentence each word belongs to, I need to map the words to the sentences in the text. For this, I created some auxiliary functions and files and split the texts from each screen into sentences using a regex:

```
(?<=[.!?:])\s+|(?<=\n)\s*|(?<=\.")
```

This regex splits the text into sentences by matching the following patterns:

- A period, exclamation mark, question mark, or colon followed by one or more whitespace characters.
- A newline character followed by zero or more whitespace characters.
- A period followed by a closing quotation mark.

Chapter 4

Data Analysis and Feature Extraction

4.1 TRT Data

Before extracting features or training models, I looked at the data collected from the experiment. The data consists of the total reading time (TRT) for each word in the text, which is the sum of the reading times for all fixations on that word. The TRT is calculated by summing the durations of all fixations on a word, which gives an indication of how long the participant spent reading that word.

The TRT data for each participant is stored in a CSV file with columns for the participant ID, the text ID, the word ID, the word itself, the sentence ID, the sentence itself, the word index in the sentence, and the TRT for that word. The TRT is measured in milliseconds. I averaged the TRT across all participants for each word in the text, which gives a more reliable estimate of the reading time for each word.

The figure 4.1 shows a histogram of the TRT data, which gives an overview of the distribution of reading times across the dataset. As is visible in the histogram, a lot of words have a total reading time of 0 milliseconds, which means that the participant did not fixate on those words at all. This is expected, as some words in the text may be skipped or read very quickly without any fixations. The histogram also shows that there are some words with very high TRT values, which indicates that those words were read more carefully or were more difficult to process.

4.2 Features

The next step is to extract features from the text that can be used to predict the TRT for each word. I extracted both simple and more complex features. The word features are as follows:

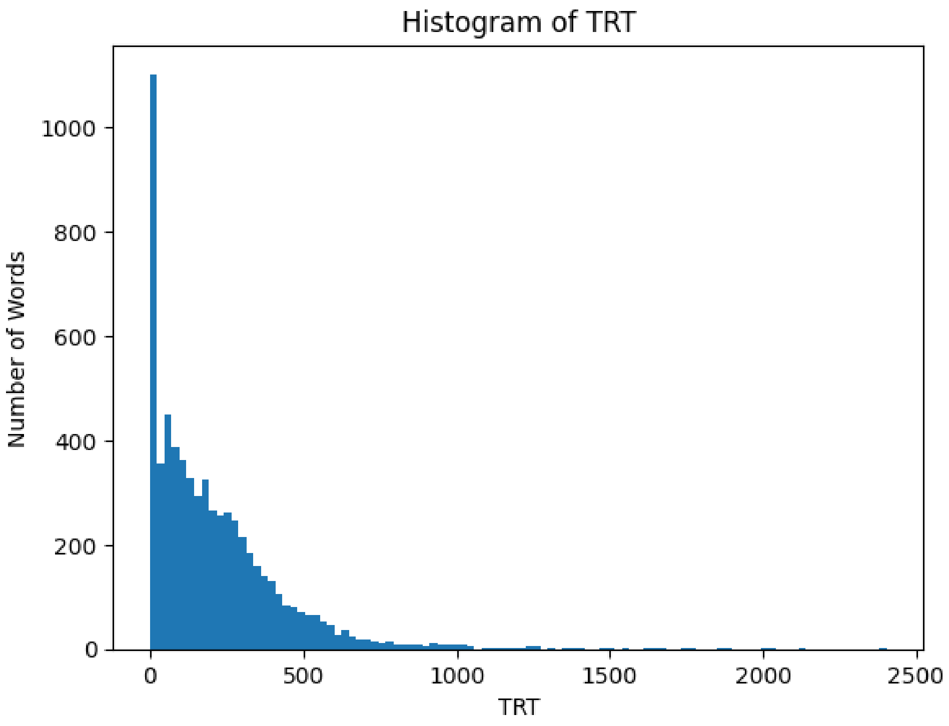

Figure 4.1: Histogram of Total Reading Time

1. **Length:** The number of characters in the word. This feature is simple but effective, as longer words tend to take more time to read.

2. **Number of tokens:** The number of tokens the word is tokenized into. This feature is useful because the more tokens the word is split into, the longer and more complex it is.

3. **Frequency:** The frequency of the word in the text. This feature is useful because more frequent words tend to be easier to read and process.

4. **Surprisal:** The surprisal of the word in the context of the sentence. Surprisal is a measure of how unexpected a word is in a given context, and it is calculated using a language model. I used a pre-trained AutoModelForMaskedLM, *dumitrescustefan/bert-base-romanian-cased-v1*, to calculate the surprisal of each word in the sentence. The surprisal is calculated as the negative logarithm of the probability of the word given the context, which is the sentence in which the word appears. The lower the surprisal, the more predictable the word is in that context.

5. **Transformer embeddings:** The contextual embedding of the word in the sentence, obtained using a pre-trained transformer model. I used a pre-trained BERT model, *dumitrescustefan/bert-base-romanian-cased-v1*, to obtain the contextual embeddings of each word in the sentence. To make this process more efficient, I first computed the embeddings for each sentence by feeding them through the BERT

Feature	Correlation with TRT
Length	0.59
Number of tokens	0.30
Frequency	-0.35
Surprisal	0.34

Table 4.1: Correlation between features and total reading time (TRT).

model and extracting the hidden states from various layers (first, middle, last, and the average across all layers).

After obtaining the token-level embeddings, I mapped them back to words. Since some words are split into multiple subword tokens by the tokenizer, I averaged the embeddings of all the tokens corresponding to a single word. This mapping was based on character-level offset information provided by the tokenizer. Subword tokens that belong to the same word have contiguous or overlapping character offsets, whereas tokens belonging to different words have non-overlapping offsets. Using this strategy, I reconstructed word-level embeddings from the token embeddings. In the end, I obtained four embeddings for each word: one from the first layer, one from the middle layer, one from the last layer, and one averaged across all layers.

After computing the features for each word, I calculated the Pearson correlation between the features and the total reading time. The results are shown in Table 4.1. As expected, the length of the word has the highest positive correlation with the TRT, followed by the surprisal and the number of tokens, while the frequency has a similar negative correlation with the TRT. The Pearson correlation is a measure of the linear relationship between two variables, and it ranges from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation. It is calculated as the covariance of the two variables divided by the product of their standard deviations by the formula:

$$r = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} \tag{4.1}$$

where r is the Pearson correlation coefficient, X and Y are the two variables, $\text{cov}(X, Y)$ is the covariance between X and Y, and σ_X and σ_Y are the standard deviations of X and Y, respectively.

Chapter 5

Models

After extracting the features, I trained several models to predict the total reading time for each word.

5.1 Metrics

In order to evaluate the performance of the models, I used the following metrics:

1. **Mean Squared Error (MSE)**: This metric measures the average of the squares of the errors, which is the average squared difference between the predicted and actual values. It is calculated as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.1}$$

where n is the number of samples, y_i is the actual value, and $\hat{y}_i$ is the predicted value. A lower MSE indicates better model performance. The MSE is sensitive to outliers, as it squares the errors, which means that larger errors have a disproportionately large impact on the MSE.

2. **R2 Score (Coefficient of Determination)**: This metric measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It is calculated as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.2}$$

where $\bar{y}$ is the mean of the actual values. The R2 score ranges from 0 to 1, where 1 indicates that the model perfectly predicts the dependent variable, and 0 indicates that the model does not explain any of the variance in the dependent variable. A negative R2 score indicates that the model is worse than simply predicting the mean of the dependent variable.

3. **Pearson Correlation Coefficient (r)**: This metric measures the linear correlation between the predicted and actual values. It is calculated as:

$$r = \frac{\text{cov}(y, \hat{y})}{\sigma_y \sigma_{\hat{y}}} \tag{5.3}$$

where $\text{cov}(y, \hat{y})$ is the covariance between the actual and predicted values, and σ_y and $\sigma_{\hat{y}}$ are the standard deviations of the actual and predicted values, respectively. The Pearson correlation coefficient ranges from -1 to 1, where 1 indicates a perfect positive linear correlation, -1 indicates a perfect negative linear correlation, and 0 indicates no linear correlation. A higher absolute value of the Pearson correlation coefficient indicates a stronger linear relationship between the predicted and actual values.

4. **Spearman Rank Correlation Coefficient (ρ):** This metric measures the strength and direction of the monotonic relationship between the predicted and actual values. It is calculated as the Pearson correlation coefficient of the ranked values. The Spearman rank correlation coefficient ranges from -1 to 1, where 1 indicates a perfect positive monotonic relationship, -1 indicates a perfect negative monotonic relationship, and 0 indicates no monotonic relationship. It is less sensitive to outliers than the Pearson correlation coefficient.

5. **Accuracy**: This metric provides an intuitive interpretation of model performance by directly reflecting the average deviation from the true values on a 0–100 scale and it is taken from the *Multilingual Language Models Predict Human Reading Behavior* [1] paper. It is defined as:

$$\text{Accuracy} = 100 - \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{5.4}$$

where n is the number of samples, y_i is the actual value, and $\hat{y}_i$ is the predicted value. This is equivalent to subtracting the Mean Absolute Error (MAE) from 100. Since the target values (reading times) are scaled to lie within the range $[0, 100]$, this metric provides a straightforward percentage-based measure of predictive accuracy, where higher values indicate better performance.

5.2 Simple Models

5.2.1 Data Preparation

First, I prepared the data for training the models. I used the features described in the previous chapter and the total reading time for each word as the target. I split the data into train and test sets, with 80% of the data used for training and 20% for testing, which means 3796 words for training and 949 words for testing. I also standardized the features using the StandardScaler from scikit-learn, which scales the features to have zero mean and unit variance, then I also standardized the total reading time.

5.2.2 Models and Results

The first models I tried were rather simple ones. I used the scikit-learn library to implement all the models except for the custom neural network, which I implemented using PyTorch. I trained the models first on the non-embedding features, then on the embeddings, and finally on both the non-embedding features and the embeddings together. The non-embedding features are the length of the word, the number of tokens, the frequency, and the surprisal. The embeddings are the contextual embeddings obtained from the BERT model. The results of the models are shown in Tables 5.1a, 5.1b, and 5.1c. All metrics are computed on the standardized reading times except the accuracy, which is calculated on the reading times in the $[0, 100]$ interval. I tried various hyperparameters for each model and selected the best ones based on the test set performance. The hyperparameters for each model are available in the annex.

5.3 BERT fine-tuning

I also fine-tuned a BERT model for the task of predicting the total reading time for each word.

5.3.1 Data Preparation

The first step was to prepare the data for fine-tuning the BERT model. Since BERT works with tokenized input, in order to use the context, I organized the data by sentences. I created a dictionary where the keys are the sentence ids and the values are the sentence text, the list of words in the sentence, and the total reading time for each word in the sentence, which I used to create a dataframe. I then split the data into train, validation, and test sets, with 80% of the data used for training, 10% for validation, and 10% for testing, and standardized the total reading time. The train set contains 250 sentences, while the validation and test sets contain 25 sentences each. The sentences were selected randomly.

Based on the dataframe, I created a dataset class built on the PyTorch Dataset class. The dataset class takes as input the dataframe, the tokenizer, and a maximum sequence length. It tokenizez the words in the sentences and creates the input tensors for the BERT model. It shortens the sentences to the maximum sequence length, and pads them if they are shorter than the maximum sequence length. The input tensors include the input ids, which are the tokenized words, the attention masks, which indicate which tokens are padding and which are not, and the targets, which are the total reading time for each word in the sentence. Because some words are split into multiple tokens, I used the word's reading time as the target for all tokens of the word.

5.3.2 Model Architecture

The first model I tried was BertForTokenClassification, using the pretrained *dumitrescustefan/bert-base-romanian-uncased-v1* model, which I modified for a regression task. BertForToken-Classification is a BERT model with a linear layer on top that outputs the probabilities for each class. In this case, I used the logits from the output, which are the raw scores for each token before being passed through a softmax function, and set the number of output classes to 1, since I want to predict a single value (the total reading time) for each token.

The second model I tried was a normal BERT model, also using the pretrained *dumitrescustefan/bert-base-romanian-uncased-v1* model. On top of the BERT model, I added a regression head, which contains a linear layer, followed by ReLU activation, layer normalization and dropout, then another linear layer that outputs a single value (the total reading time).

5.3.3 Training

I tried several different training strategies for the BERT models. The strategy that worked best was based on gradual unfreezing. I trained the first model for a total of 30 epochs, unfreezing 2 layers every 5 epochs. The strategy for the second model was the same as the first, except that I first trained only the regression head for 5 epochs, then I gradually unfreezed the BERT layers, unfreezing 2 layers every 5 epochs, for a total of 35 epochs. I used the AdamW optimizer with a learning rate of 1e-4, a weight decay of 1e-4, and a cosine scheduler with warmup with 10 warmup steps. The batch size was set to 8. The dropout rate for the first model was set to 0.3, while for the second model it was set to 0.2.

The loss function used for training was the mean squared error, which I customized to ignore the padding tokens in the input. For getting the word-level predictions, I averaged the predictions for each word based on the tokens that make up the word, so that the final prediction for each word is the average of the predictions for the tokens. The evaluation metrics were calculated on the word-level predictions.

5.3.4 Results

The results of the BERT fine-tuning models are shown in Table 5.2.

Table 5.1: Results of simple models trained on different feature sets.

Model	MSE	R2	Pearson	Spearman	Accuracy
Linear Regression	0.60	0.36	0.60	0.67	93.25
Elastic Net	0.60	0.36	0.60	0.67	93.22
SGD Regressor	0.60	0.36	0.60	0.67	93.26
Bayesian Ridge	0.60	0.36	0.60	0.67	93.24
SVR Linear Kernel	0.64	0.32	0.60	0.67	92.96
SVR RBF Kernel	0.64	0.32	0.60	0.67	87.66
K Neighbors Regressor	0.70	0.26	0.54	0.62	88.67
Random Forest Regressor	0.66	0.30	0.58	0.63	92.23
Gradient Boosting Regressor	0.57	0.40	0.63	0.69	92.64
Hist Gradient Boosting Regressor	0.59	0.38	0.62	0.68	87.81
Kernel Ridge Linear Kernel	0.60	0.36	0.60	0.67	93.25
Kernel Ridge RBF Kernel	0.60	0.37	0.61	0.67	86.55
MLP Regressor	0.60	0.36	0.61	0.67	93.58
Custom Neural Network	0.60	0.37	0.61	0.68	93.14

(a) Non-embedding features.

Model	MSE	R2	Pearson	Spearman	Accuracy
Linear Regression	0.66	0.31	0.59	0.63	74.11
Elastic Net	0.59	0.36	0.61	0.67	76.30
SGD Regressor	0.73	0.23	0.57	0.61	72.69
Bayesian Ridge	0.59	0.38	0.62	0.68	75.24
SVR Linear Kernel	0.66	0.30	0.57	0.64	70.27
SVR RBF Kernel	0.58	0.38	0.63	0.69	73.95
K Neighbors Regressor	0.66	0.29	0.57	0.64	87.40
Random Forest Regressor	0.67	0.28	0.53	0.61	76.00
Gradient Boosting Regressor	0.71	0.25	0.53	0.62	74.58
Hist Gradient Boosting Regressor	0.63	0.33	0.58	0.65	73.99
Kernel Ridge Linear Kernel	0.66	0.31	0.59	0.63	74.13
Kernel Ridge RBF Kernel	0.55	0.42	0.65	0.69	78.21
MLP Regressor	0.68	0.28	0.58	0.64	87.48
Custom Neural Network	0.52	0.39	0.62	0.66	82.32

(b) Embedding features.

Model	MSE	R2	Pearson	Spearman	Accuracy
Linear Regression	0.63	0.33	0.61	0.64	88.02
Elastic Net	0.55	0.41	0.65	0.70	92.99
SGD Regressor	0.68	0.29	0.58	0.60	85.97
Bayesian Ridge	0.55	0.41	0.65	0.69	85.96
SVR (RBF Kernel)	0.56	0.40	0.65	0.69	82.31
K Neighbors Regressor	0.70	0.26	0.56	0.62	89.48
Random Forest Regressor	0.56	0.41	0.64	0.69	91.36
Gradient Boosting Regressor	0.58	0.39	0.63	0.69	92.98
Hist Gradient Boosting Regressor	0.53	0.44	0.67	0.70	83.79
Kernel Ridge (Linear)	0.63	0.33	0.61	0.64	88.01
Kernel Ridge (RBF)	0.53	0.44	0.66	0.70	87.20
MLP Regressor	0.74	0.22	0.56	0.60	87.81
Custom Neural Network	0.50	0.43	0.66	0.67	87.93

(c) All features.

Model	MSE	R2	Pearson	Spearman	Accuracy
BertForTokenClassification	1.06	0.43	0.66	0.66	85.53
Bert with Regression Head	1.06	0.42	0.64	0.68	88.47

Table 5.2: Results of BERT fine-tuning models.

Chapter 6

Lexical Simplification

Lexical simplification refers to the process of simplifying texts by replacing complex words with simpler alternatives. This is useful in making texts more accessible to a wider audience, including those with reading difficulties or non-native speakers.

Usually, the complexity of words is the metric used to determine which words to simplify. It can be measured in various ways, such as by the length of the word, its frequency of use, or its familiarity to the reader. It can also be estimated by predicting the complexity based on an annotated corpus.

Another approach to identify which words to replace is to look at the total reading time of words in a text. The more time it takes to read a word, the more complex it is likely to be. Simplifying these words can lower tha overall reading time of the text, making it easier to read and understand.

6.1 Lexical Simplification Pipeline

In my project, I implemented a lexical simplification pipeline that includes the following steps:

1. Computing the total reading time of each word in a text.
2. Identifying the words that take the longest to read.
3. Generating a list of alternatives for these complex words.
4. Computing the reading time of the alternatives and selecting the best ones.
5. Replacing the complex words with the selected alternatives in the text.

6.2 Replacement Generation

To generate candidate replacements for complex words, I used the masked language model *dumitrescustefan/bert-base-romanian-cased-v1*. I implemented three methods for generating alternatives, each leveraging a slightly different context strategy:

1. **Basic Masking in the Sentence:** The complex word is replaced with a `[MASK]` token in the original sentence, and the model predicts suitable replacements based on this context.

2. **Sentence Pair with Masked Sentence First:** The masked sentence is concatenated with the original sentence using a `[SEP]` token, allowing the model to consider both contexts, inspired by the LSBert framework for lexical simplification [5]:

   ```
   [MASKED_SENTENCE] [SEP] [ORIGINAL_SENTENCE]
   ```

3. **Sentence Pair with Original Sentence First:** The original sentence is placed before the masked sentence, exploring whether this order affects the model's predictions:

   ```
   [ORIGINAL_SENTENCE] [SEP] [MASKED_SENTENCE]
   ```

Chapter 7

Interface

7.1 Overview

To support users in identifying and simplifying complex words in a text, I developed a web-based tool called *Reading Time Estimator*. This tool enables users to visualize word-level reading difficulty (using TRT, or Total Reading Time) and interactively replace complex words with simpler alternatives. The system combines machine learning models on the backend with a responsive, interactive frontend.

The system architecture consists of a browser-based frontend communicating with a Flask backend that routes requests to two main components: the TRT predictor and the simplifier module.

7.2 Backend

The backend of the application is implemented in Python using the Flask framework. Flask was chosen for its lightweight architecture and flexibility, making it ideal for integrating machine learning services in web environments.

The backend is responsible for the following tasks:

- **Reading Time Estimation:** Upon receiving a user input, the text is passed to the `estimate_trt()` function, which uses a trained regression model to estimate reading times.

- **Word Simplification:** When a user clicks on a word, a POST request is sent to the `/simplify` route. The `simplify_word()` function retrieves top-k replacement candidates using contextual language models and re-evaluates their reading time. The replacement with the lowest TRT (and lower than the original) is selected.

- **API Communication:** All dynamic interactions between frontend and backend use JSON for data exchange. This enables asynchronous updates (via JavaScript fetch) without requiring full page reloads.

Additionally, the backend is modular: machine learning logic is separated into `trt_model` and `simplifier` modules, making it extensible and easy to maintain.

7.3 Frontend

The frontend is implemented using HTML, CSS, and JavaScript, with dynamic content rendered via Jinja2 templates.

Major features include:

- **Interactive Heatmap:** After the user submits text, each word is displayed in a color-coded heatmap. Colors range from green (easy) to red (hard), based on normalized TRT values. This is computed in JavaScript using HSL color interpolation.
- **Word Simplification Interface:** Users can click on individual words to request simpler alternatives. This triggers an AJAX request to the `/simplify` route. The replacement word is rendered inline, with updated color and tooltip.
- **Responsive Design:** The interface is mobile-friendly and uses simple, accessible styles for clarity and usability.

Chapter 8

Conclusion

Bibliography

[1] Nora Hollenstein, Federico Pirovano, Ce Zhang, Lena Jäger, and Lisa Beinborn. "Multilingual Language Models Predict Human Reading Behavior". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou. Online: Association for Computational Linguistics, June 2021, pp. 106–123. DOI: 10.18653/v1/2021.naacl-main.10. URL: https://aclanthology.org/2021.naacl-main.10/.

[2] Deborah N. Jakobi, Maroš Filip, Cui Ding, and Lena A. Jäger. *MultiplEYE Image Generation Software*. Computer software. Version 1.0. If you use this software, please cite it as below. Mar. 25, 2025. URL: https://github.com/theDebbister/wg1-image-creation/.

[3] Daniel G. Krakowczyk, David R. Reich, Jakob Chwastek, Deborah N. Jakobi, Paul Prasse, Assunta Süß, Oleksii Turuta, Paweł Kasprowski, and Lena A. Jäger. "pymovements: A Python Package for Processing Eye Movement Data". In: *Proceedings of the 2023 Symposium on Eye Tracking Research and Applications (ETRA '23)*. ETRA '23. Tübingen, Germany: Association for Computing Machinery, 2023. ISBN: 979-8-4007-0150-4. DOI: 10.1145/3588015.3590134. URL: https://doi.org/10.1145/3588015.3590134.

[4] *MultiplEYE Experiment implementation Software*. Computer software. 2025. URL: https://github.com/MultiplEYE-COST/wg1-experiment-implementation.

[5] Jipeng Qiang, Yun Li, Yi Zhu, Yun-Hao Yuan, and Xindong Wu. *LSBert: A Simple Framework for Lexical Simplification*. June 2020. DOI: 10.48550/arXiv.2006.14939.