

Warsaw University of Technology

FACULTY OF  
MATHEMATICS AND INFORMATION SCIENCE



# Bachelor's diploma thesis

in the field of Data Science

Clickbait News Detection and Analysis

**Tymoteusz Urban**

student record book number 320665

**Mateusz Kubita**

student record book number 323601

**Wojciech Michaluk**

student record book number 320646

thesis supervisor

Anna Wróblewska, PhD

WARSAW 2025



## Abstract

### Clickbait News Detection and Analysis

This engineering thesis presents an analysis of clickbait articles using machine learning and artificial intelligence methods. The team conducted a review of existing research in this area, which allowed the creation of a custom metric, referred to in the thesis as the baitness measure, which quantifies the clickbait nature of articles. Natural language processing techniques were applied to find patterns that allow the classification of articles into the clickbait category based on their titles. Machine learning models were built using vectorized text representations for clickbait classification. Large language models based on the transformer architecture were used to process the text into vectors. Generative artificial intelligence was employed to summarize suspicious articles so that users could quickly determine whether the article's content is worth their attention.

Among the applied machine learning models, the best overall was XGBoost, which combined informativeness measures with embeddings generated using OpenAI, achieving the highest F1-score of 91%.

To showcase the results, a fully functional browser extension was developed for chromium-based browsers. Our tool is capable of warning users both before and after they access a clickbait article. After opening an article, the user receives a percentage score indicating the likelihood of the article being a clickbait. The prediction is explained based on the analyzed metrics, including those developed by the team. A one- to two-sentence summary of the entire article is also provided.

**Keywords:** clickbait detection, artificial intelligence, machine learning, natural language processing, generative AI, transformer models, baitness measure, browser extension, text classification



## Streszczenie

### Wykrywanie i analiza wiadomości typu clickbait

Niniejsza praca inżynierska przedstawia analizę artykułów typu *clickbait* za pomocą metod uczenia maszynowego oraz sztucznej inteligencji. Zespół przeprowadził przegląd istniejących badań w tym obszarze, co pozwoliło utworzyć własną metrykę, w pracy zwaną jako *baitness measure*, która mierzy *clickbaitowość* artykułów. Zastosowano techniki przetwarzania języka naturalnego, w celu odnalezienia wzorców pozwalających klasyfikować artykuły do klasy *clickbait* na podstawie ich tytułów. Zbudowano modele uczenia maszynowego wykorzystujące wektorowe reprezentacje tekstu do klasyfikacji clickbaitów. Do przetworzenia tekstu na wektory wykorzystano duże modele językowe oparte na architekturze *transformer*. Generatywna sztuczna inteligencja została użyta do streszczania podejrzanych artykułów, aby użytkownik mógł szybko ocenić, czy treść artykułu jest warta jego uwagi.

Pośród zastosowanych modeli uczenia maszynowego, najlepszym modelem okazał się XGBoost, który łączy miary informatywności z reprezentacjami wektorowymi wygenerowanymi przy użyciu modeli OpenAI. Uzyskał on najwyższą miarę F1 ze wszystkich modeli, wynoszącą 91%.

W celu ukazania wyników prac powstało w pełni funkcjonalne rozszerzenie do przeglądarek opartych na silniku *chromium*. Wspomniane rozszerzenie jest w stanie ostrzegać użytkowników o tytułach typu clickbait, zarówno przed, jak i po odwiedzeniu strony z artykułem. Po otwarciu artykułu użytkownik otrzymuje wskaźnik procentowy określający prawdopodobieństwo *clickbaitu*. Predykcja jest wytłumaczona w oparciu o analizowane metryki, w tym te stworzone przez zespół. Przedstawione jest również jedno- lub dwuzdaniowe streszczenie całego artykułu.

**Słowa kluczowe:** wykrywanie clickbaitów, sztuczna inteligencja, uczenie maszynowe, przetwarzanie języka naturalnego, generatywna sztuczna inteligencja, modele transformatywne, miara baitness, rozszerzenie przeglądarkowe, klasyfikacja tekstu



## Acknowledgements

We would like to thank our supervisor, PhD Anna Wróblewska, for her guidance and support throughout the work and for always being ready to help us. Working together was a pleasant and successful experience.

In addition, we are thankful to PhD Daniel Dan, PhD Robert Paluch and M.Sc. Eng. Adam Majczyk for their assistance and helpful remarks <sup>1</sup>. Their input was greatly appreciated.

---

<sup>1</sup>This work was supported by the European Union under the Horizon Europe grant "Overcoming Multilevel Information Overload" (OMINO, <https://ominoproject.eu>, grant no. 101086321) and by the Polish Ministry of Education and Science within the framework of the program titled International Projects Co-Financed. However, the views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union or the European Research Executive Agency. Neither the European Union nor the European Research Executive Agency can be held responsible for them.

Warsaw, .....

### Declaration

We hereby declare that:

1. We have not used IT tools to generate the content of the manuscript of this thesis.
2. We have not used IT tools to generate the code of the software developed for the thesis.
3. We take full responsibility for all content in this thesis, including both the manuscript and the software developed for it.

.....





# Contents

<b>1. Introduction</b>	<b>11</b>
1.1. Division of work	12
<b>2. Requirements and risk analysis</b>	<b>14</b>
2.1. Related Tools	14
2.2. Requirements	14
2.2.1 Functional requirements	14
2.2.2 Non-functional requirements	15
2.3. Risk Analysis	17
<b>3. Datasets</b>	<b>18</b>
3.1. Data sources	18
3.2. Combining and preparing the datasets	20
<b>4. Exploratory data analysis and data cleaning</b>	<b>22</b>
4.1. Granular comparison of datasets for detection	22
4.1.1 Initial look	22
4.1.2 Lexical characteristics	23
4.2. Insights and visual analysis	24
4.2.1 Merging data into final dataset for clickbait detection	24
4.2.2 Lexical characteristics	26
4.2.3 Outliers	28
4.2.4 N-gram and word analysis	29
4.2.5 Named entity recognition	30
4.2.6 Sentiment analysis	32
4.3. Clickbait spoiling dataset	33
<b>5. Informativeness measures</b>	<b>36</b>
5.1. Related work	36
5.2. Our approach	37
5.3. Analysis	38
5.3.1 Title analysis	39

5.3.2	Text body analysis . . . . .	41
5.4.	Custom measure . . . . .	42
<b>6.</b>	<b>Clickbait detection . . . . .</b>	<b>46</b>
6.1.	Related work . . . . .	46
6.2.	Our approach . . . . .	47
6.3.	Data preprocessing . . . . .	51
6.4.	Models based on informativeness measures . . . . .	52
6.4.1	Title only . . . . .	52
6.4.2	Title and body . . . . .	53
6.5.	Models based on TF-IDF . . . . .	54
6.5.1	TF-IDF Feature Extraction with Linear Discriminant Analysis (LDA) . .	54
6.5.2	TF-IDF Feature Extraction with Random Forest and XGBoost Classifiers	55
6.6.	Word embedding models . . . . .	57
6.6.1	Word2Vec . . . . .	57
6.6.2	GloVe - Global Vectors for Word Representation . . . . .	58
6.6.3	TF-IDF weighted Word2Vec . . . . .	59
6.7.	Large language models . . . . .	60
6.7.1	OpenAI . . . . .	60
6.7.2	RoBERTa . . . . .	64
6.8.	OpenAI Embeddings combined with informativeness measures . . . . .	65
6.9.	Final clickbait model selection . . . . .	67
<b>7.</b>	<b>Article spoiling . . . . .</b>	<b>68</b>
7.1.	Related work . . . . .	68
7.2.	Our approach . . . . .	70
7.3.	Data preprocessing . . . . .	70
7.4.	Large language models . . . . .	70
7.4.1	OpenAI . . . . .	70
7.4.2	BERT based models . . . . .	80
7.4.3	T5-large . . . . .	81
7.4.4	Final spoiling model selection . . . . .	82
<b>8.</b>	<b>Browser extension . . . . .</b>	<b>83</b>
8.1.	Architecture . . . . .	83
8.1.1	Overview . . . . .	83
8.1.2	Extension architecture . . . . .	83

8.1.3	Python API architecture . . . . .	84
8.2.	Modes . . . . .	86
8.2.1	Post-click detection . . . . .	86
8.2.2	Pre-click detection . . . . .	88
8.3.	GUI . . . . .	89
8.3.1	Popup . . . . .	89
8.3.2	Extension icon and badges . . . . .	89
8.3.3	In-page icons . . . . .	90
8.3.4	Options page . . . . .	91
8.4.	Testing . . . . .	93
8.4.1	Unit Tests . . . . .	93
8.4.2	Integration Tests . . . . .	93
8.4.3	Performance Tests . . . . .	94
8.4.4	User Interface Tests . . . . .	94
8.4.5	Usability tests . . . . .	95
8.4.6	Acceptance Tests . . . . .	95
<b>9.</b>	<b>Summary . . . . .</b>	<b>98</b>
9.1.	Future work and limitations . . . . .	98
	<b>Bibliography . . . . .</b>	<b>99</b>
	<b>List of Figures . . . . .</b>	<b>107</b>
	<b>List of Tables . . . . .</b>	<b>109</b>



# 1. Introduction

Clickbaits are a common problem on the Internet, as they not only waste time of users but also contribute to disinformation. Clickbait titles are often written to catch users' attention while hiding relevant information to compel them to click on the article [1]. This practice arises because content creators earn revenue from ads displayed on the website hosting the article. Since income increases with higher website traffic, article titles are frequently designed to draw attention. Our goal was to address this issue by developing machine learning models to automatically detect clickbaits and filter content on the Internet. Those models analyze articles titles and content to identify whether they have a bait character and properly classify articles. It allows filtering this type of content or warning users in advance. Our team tested classic machine learning models and more advanced deep learning models, including tests of commercial large language models such as GPT-4 or BERT [2] [3]. Thanks to the broad spectrum of our analysis, we were able to choose the most appropriate model to solve the problem of detecting clickbaits in advance.

In addition to classification, we also dealt with the task of spoiling the clickbait. Often clickbait articles contain a lot of irrelevant information, and the main answer to the title is hidden somewhere in the text. Developed methods allow us to extract relevant information from the articles and provide it to the end users.

What is more, we implemented a browser extension to analyze internet sites in real time. This tool is operational on a web browser. End users are provided with the option to perform a check of an article to find out if an article is a clickbait. Our extension is also capable of automatically identifying clickbaits. This allows users to quickly assess whether the content is worth their attention. The extension offers a user-friendly interface designed not only for developers but also for normal end-users of browsers. We named it *ClickGuard*.

In Figure 1.1, we present a visual abstract showing the key findings of our research on the detection and analysis of clickbait news.

In this document, our team guides the reader through the work completed during the project. First, requirements and risk analysis are described, including how we identified market needs for the browser extension product. Next, the data acquisition process is explained. Our team outlines the exploratory data analysis and initial data cleaning process. Following this, the analysis of

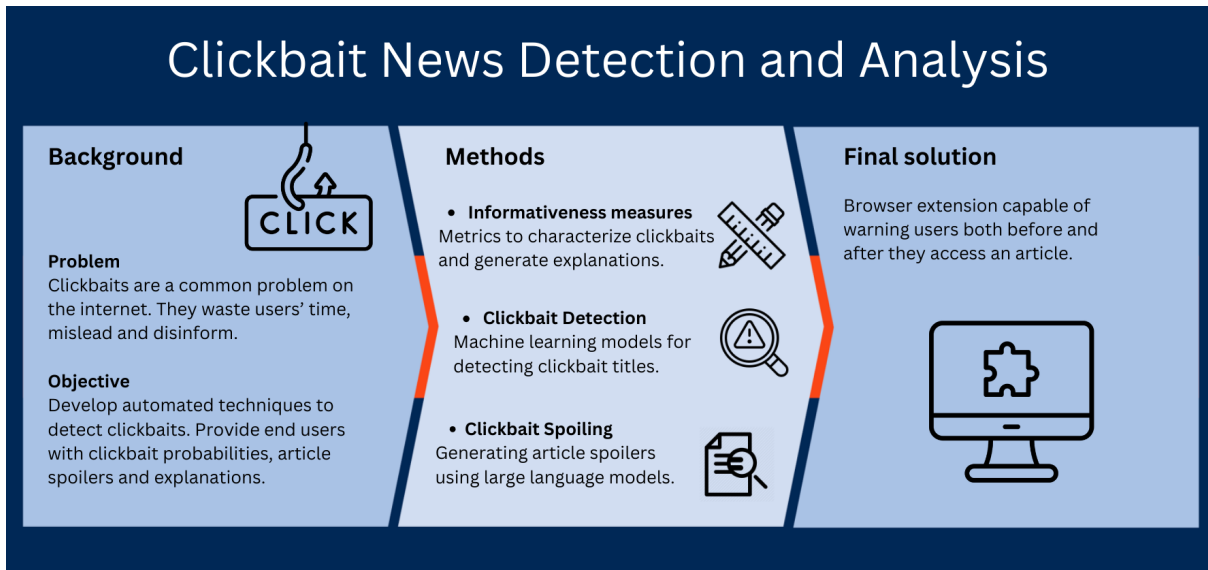


Figure 1.1: Visual abstract of the thesis

the master dataset is presented. In the Section 5, we introduce our custom baitness measure. Subsequently, the process of clickbait detection performed by our team is described. Additionally, we explain the article spoiling task and detail how our browser extension was designed. Finally, in the summary, our team outlines the limitations of *ClickGuard* and discusses future work.

### 1.1. Division of work

The contribution of the authors of the thesis is presented in the table 1.1.

Table 1.1: The contribution of the authors of the thesis

Role	Wojciech Michaluk, 320646	Tymoteusz Urban, 320665	Mateusz Kubita, 323601
Writing – original draft	1, 4, 7, 9.1	2.1, 2.2, 5, 6.4, 8, appendices	2.3, 3, 6, 9
Writing – review & editing	1, 4, 7, 9	1, 2, 5, 8, 9, appendices	1, 3, 6
Software	Clickbait spoiling models	Browser extension, python API service	Clickbait detection models
Key solutions of the work <ul style="list-style-type: none"> <li>• Conceptualization</li> <li>• Investigation</li> <li>• Methodology</li> </ul>	System requirements, related work review concerning clickbait spoiling	System requirements, system architecture, related work and tools review	Risk analysis, system design, related work review concerning machine learning models for clickbait detection
Formal Analysis (including explanatory data analysis) and Validation	Exploratory data analysis	Informativeness measures analysis	Machine learning models validation
Data curation	Yes	Yes	Yes
Project administration	Yes	Yes	Yes
Resources	No	Yes	No
Visualization	Yes	Yes	Yes
Funding acquisition	No	No	No



## 2. Requirements and risk analysis

### 2.1. Related Tools

On Chrome Web Store there can be found several browser extensions for detecting clickbaits. However, most of them are limited to only one website (e.g., Facebook or Youtube) or just do not work. Still, there are some more universal ones, for example *CliNe - Clickbait News Detector* [4], which is recommended by Google. It provides real-time clickbait warnings by displaying alerts, which automatically appear after user enters an article. The extension sends requests to external API, which uses LSTM-based deep learning model for clickbait classification. However, this tool provides little information about the prediction, as no probabilities or explanations are displayed. Also it is limited as it works only after user enters desired page.

Another type of clickbait detection tools are websites, which work as wrappers for displaying articles aggregated from multiple sources. Good example is a *ClickbaitDetector* [5] website, where user can browse articles from multiple Spanish news portals and filter them by categories or original source. Each article is presented with the probability of clickbait and key baiting characteristics of the title. Main limitation of this tool, is that users must rely on this website as their news outlet instead of the original sites.

### 2.2. Requirements

#### 2.2.1. Functional requirements

In this section, we provide the functional requirements for our browser extension, which were defined prior to the development. The system was created in line with them, but some additional features were added. More details about the app can be found in section 8. Figure 2.1 shows a simplified illustration of actors and their available actions in our system, while more detailed action descriptions can be found in Table 2.1.

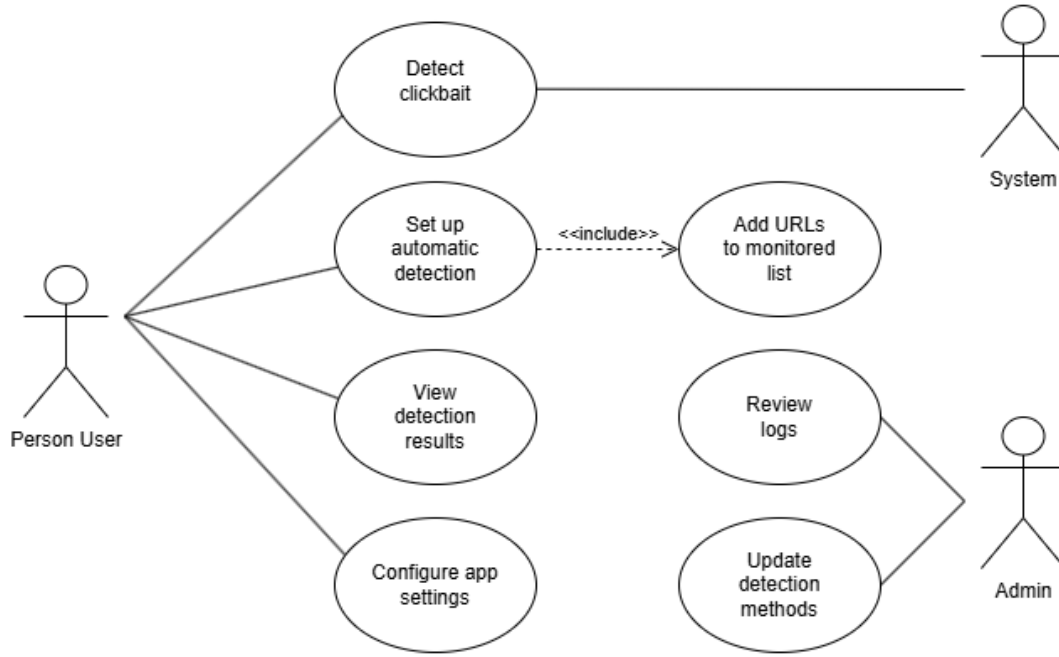


Figure 2.1: Main browser extension logic use case diagram

### 2.2.2. Non-functional requirements

Non-functional requirements focus on how well the system performs its tasks. To present non-functional requirements of our clickbait detection tool, we have created a FURPS (Functionality, Usability, Reliability, Performance, Supportability) Table 2.2. By using FURPS we ensure all important aspects of our software are considered. It covers functionality (features, capabilities), usability (human interaction, aesthetics), reliability (failures, recoverability, accuracy), performance (speed, response time), and supportability (maintainability, adaptability, and compatibility) [6]. We omit the functional requirements part, as it was presented in the previous section.

Table 2.1: Functional requirements

Actor	Use case	Description
Person User	Detect clickbait on request	The tool should allow users to manually trigger the extension and run clickbait check.
	Setting up automatic clickbait detection	The user can enable automatic detection. The extension should automatically detect clickbaits for sites that are articles (based on meta properties). Additionally, the user should be allowed to add URL patterns for which automatic scans will be performed.
	View detection results	Users can view detection results in the extension popup or by looking at a badge, which would change based on the detection result. Predictions are saved in browser history, so processing the same article multiple times is avoided. Also, an explanation and spoiler are provided.
	Configure app settings	Users can customize the app's behavior to suit their preferences. They can modify detection modes, add monitored sites, and enable or disable certain features.
System	Handle API failure	If the API fails or is unavailable, the end user should be notified.
	Automatically check for clickbaits	If set up, the extension should automatically detect if an article contains a clickbait title.
Admin	Review logs	Admin should have the ability to view logs and detection history on the server side.
	Update models	Admin can easily change the models or methods for clickbait detection.

Table 2.2: FURPS Analysis

Requirements area	Description
Functional	See Table 2.1
Usability	Extension popup should be simple, intuitive and easy to use.
	Colored badge on extension icon showing prediction.
Reliability	Error handling for API failures or unavailability.
	Accurate predictions by ML models so the tool is reliable.
Performance	Classification should take no longer than 3 seconds.
	The extension should not negatively impact site loading.
Supportability	Cross-browser compatibility or special versions for other browsers.
	Extension can be upgraded independently from Python API and vice versa.

## 2.3. RISK ANALYSIS

All requirements were met, apart from the cross-browser compatibility. Our tool is available only for chromium-based browsers.

### 2.3. Risk Analysis

At the beginning of the project, we believed that the SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis (Table 2.3) technique would be helpful at that point. It is a decision-making tool that enables us to make informed decisions and evaluate internal and external factors that have an impact on our project [7].

Table 2.3: SWOT Analysis

Strengths	Weaknesses
Advancements in Machine learning: There is significant progress in ML, especially in natural language processing algorithms (NLP) and large language models (LLM) that make it easier to build our own solution for predicting clickbaits.	Data Quality Issues: We are aware that very often the available data might be of low quality, which can lead to worse model performance.
Robust solution: We aim to develop a simple, user-friendly browser extension that alerts users of potential clickbait content. In a time, when clickbait headlines dominate online media, this solution addresses a problem by promoting informed browsing and reducing frustration caused by misleading content.	Interpretability: Since we also use deep learning models, they are often perceived as black boxes, making it hard to explain how they actually work.
Opportunities	Threats
Expansion beyond the English language: Our solution may detect clickbaits in multiple languages, such as English, Polish, Slovenian.	Ethical concerns: The algorithm may be biased, leading to mistrust among users of this extension.
Growing awareness around misinformation: With growing awareness, this tool might be an opportunity to position clickbait detection as part of a larger Internet content verification system.	Low prediction accuracy: It may negatively impact the willingness to use the extension on a daily basis.

### 3. Datasets

#### 3.1. Data sources

We introduce four datasets used in this project. Two of them come from Kaggle - a common data source among data scientists, one from Clickbait Challenge, and another provided by SemEval, an organization hosting scientific events. These datasets are in English and we build our models only for this language. However, there is potential to extend it to more languages in the future. Mentioned datasets are presented below.

1. Kaggle-1: [8] [kaggle.com/datasets/amananandrai/clickbait-dataset](https://kaggle.com/datasets/amananandrai/clickbait-dataset). It consists of 32 000 articles with titles and labels identifying clickbaits from websites such as ‘BuzzFeed’, ‘Upworthy’, ‘ViralNova’, ‘Thatscoop’, ‘Scoopwhoop’ and ‘ViralStorie’.
2. Kaggle-2: [9] [kaggle.com/datasets/vikassingh1996/news-clickbait-dataset](https://kaggle.com/datasets/vikassingh1996/news-clickbait-dataset) (only train2.csv file, as train1.csv corresponds to the Kaggle-1 dataset). This dataset comes from Kaggle InClass Prediction Competition and contains 21,029 articles, each with title, text and label indicating whether it is a clickbait or not.
3. Clickbait Challenge 2017: [10] [webis.de/events/clickbait-challenge/shared-task.html](http://webis.de/events/clickbait-challenge/shared-task.html). For the purpose of Clickbait Challenge that was held in 2017 organizers compiled this dataset. It contains a total of 38 830 Twitter posts from 27 major US news publishers, each one annotated on a 4-point scale [not click baiting (0.0), slightly click baiting (0.33), considerably click baiting (0.66), heavily click baiting (1.0)] by five annotators from Amazon. They also mention that "it is authentic, representative and rich in terms of potential features, unbiased, and large scale" [11]. It contains many columns, from which we only use *title* of the tweet and *truthClass* that is the median clickbait score by annotators.
4. SemEval Spoiling Dataset: [12] [13] [pan.webis.de/semeval23/pan23-web/clickbait-challenge.html](http://pan.webis.de/semeval23/pan23-web/clickbait-challenge.html). This dataset is designed specifically for clickbait spoiling task. The corpus was manually collected from five social media accounts (on Twitter, Reddit, and Facebook) which deal with manual clickbait spoiling: r/savedyouaclick, @HuffPoSpoilers,

### 3.1. DATA SOURCES

@SavedYouAClick, @UpworthySpoiler, and @StopClickBaitOfficial. Additionally, some articles were taken from the Clickbait Challenge 2017 with manual spoiling. Apart from standard columns, it also has a clickbait spoiler extracted from the article body and a spoiler generated by a human. Thanks to those fields, we are able to train our own clickbait spoiling models. Columns of this dataset are listed below.

- **targetTitle**: The title of the linked web page.
- **targetParagraphs**: The main content of the linked web page, consisting of manually extracted paragraphs from the main content.
- **humanSpoiler**: The human-generated spoiler (abstractive) for the clickbait post from the linked web page. There may be missing values.
- **spoiler**: The human-extracted spoiler for the clickbait post from the linked web page.
- **spoilerPositions**: The position of the human-extracted spoiler for the clickbait post from the linked web page.
- **tags**: The spoiler type, which may be classified as "phrase," "passage," or "multi" for task 1 (spoiler type classification).

To sum it up, the key details about the datasets are presented in Table 3.1.

Table 3.1: Summary of clickbait detection and spoiling datasets

Name	Kaggle-1	Kaggle-2	Clickbait Chal- lenge 2017	SemEval Spoiling Dataset
Number of labelled ti- tles	32,000	21,029	38,830	4,000
Usage	Detection	Detection	Detection	Spoiling
Columns	title (headline), bi- nary label indicat- ing whether it is clickbait or not	title (headline), bi- nary label indicat- ing whether it is clickbait or not	postTimestamp, postText, post- Media, targetTitle, targetDescription, targetKeywords, targetParagraphs, targetCaptions, truthJudgments, truthMean, truth- Median, truth- Mode, truthClass	targetTitle, tar- getParagraphs, humanSpoiler, spoiler, spoilerPo- sitions
Collecting method	Gathered from popular news websites. The an- notation method is unknown	Collected for a Kaggle competi- tion	Sourced from Twit- ter and annotated by five experts	Extracted from so- cial media accounts known for exposing clickbait

### 3.2. Combining and preparing the datasets

We prepared 3 different datasets for different tasks and models. In order to have a greater variability of sources for clickbait detection, we concatenate original datasets so that our models may be more robust and generalizable across diverse styles and formats of clickbait content.

- **Dataset for title-based classification**

This is our main dataset. It consists of 3 datasets: Kaggle-1, Kaggle-2 and Clickbait Challenge 2017 dataset. The two Kaggle datasets, referred to as Kaggle-1 and Kaggle-2, each contain two columns: the title (headline) of an article and a binary label indicating whether the article is clickbait or not. In contrast, the Clickbait Challenge 2017 dataset offers a

more comprehensive set of features, including 'postTimestamp', 'postText', 'postMedia', 'targetTitle', 'targetDescription', 'targetKeywords', 'targetParagraphs', 'targetCaptions', 'truthJudgments', 'truthMean', 'truthMedian', 'truthMode', and 'truthClass'. To maintain consistency across datasets and simplify processing, we reduced the Clickbait Challenge 2017 dataset to include only the 'title' and 'truthClass' columns. Here, the 'truthClass' label represents the median of the 'truthJudgments' column, which is annotated on a 4-point scale. This 'truthClass' median label serves as a comparable clickbait indicator to the binary labels provided in the Kaggle datasets. Finally, we combine Kaggle-1, Kaggle-2, and the streamlined Clickbait Challenge 2017 dataset into a single, unified dataset for further analysis of clickbait detection. Merging these datasets into a single dataset is an important step because it provides a variety of sources and strengthens the dataset's reliability. More details about concatenation are described in Section 4.2.

- **Dataset for models utilizing body of the article**

For models utilizing both titles and article bodies, we additionally created a balanced dataset based only on the Clickbait Challenge 2017 dataset. It's the only one that contains article content, so there was no need for concatenation with other datasets.

- **Clickbait Spoiling dataset**

For the spoiling task the SemEval Spoiling Dataset dataset is used.



## 4. Exploratory data analysis and data cleaning

Our approach was to first analyze the original datasets separately. In Section 4.1, we checked the quality of the data and basic statistics to find out if the used datasets are similar. We plotted distributions of title text lengths in Figure 4.2. After that, we merged them and created a balanced dataset, on which we performed further, more detailed analysis (see Section 4.2).

### 4.1. Granular comparison of datasets for detection

In this section, we analyze each original dataset separately to find out if the datasets have similar distribution. We describe how the process of getting familiar with the data in the project was held.

#### 4.1.1. Initial look

First, we are performing data preprocessing. This operation refers to checking if data in datasets are valid. Then data are cleaned and prepared for further, detailed analysis. While preprocessing, the data was displayed for a visual check. At first glance, there were no inconsistencies in text formatting. All data was in 'string' format, and there were no 'null' or empty values in the datasets. Additionally, the 'langid' library was used to check the language of the titles. All of them are in English. Then clickbait distribution in the datasets was checked; see Figure 4.1.

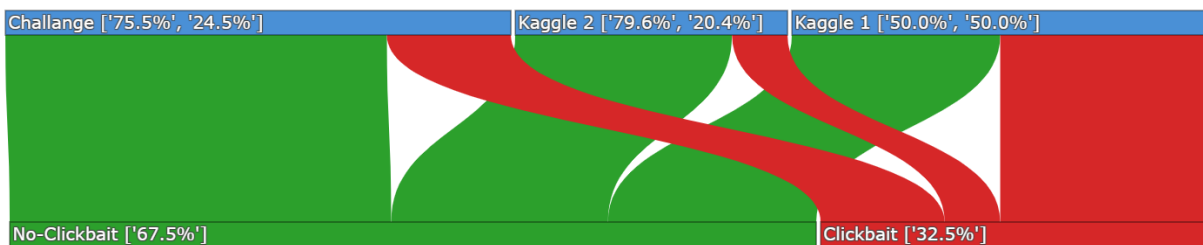


Figure 4.1: Sankey diagram of clickbait distribution by dataset

The significance of clickbait data comes from the 1st Kaggle dataset where clickbaits are 50% of the data. The number of clickbaits in Kaggle-2 is only 20.4% and in 'Clickbait Challenge' is

#### 4.1. GRANULAR COMPARISON OF DATASETS FOR DETECTION

24.5%. Thus, in the later merged dataset, there are 67,5% non-clickbait and 32.5% clickbait titles (Figure 4.1). A balanced dataset is needed to ensure that the machine learning models used can capture patterns in both types of article titles: clickbait and non-clickbait. Moreover, accuracy can be appropriately used to assess model performance on a balanced dataset (see Section 4.2).

##### 4.1.2. Lexical characteristics

The analysis of substantial statistics for each dataset for detection is presented in Table 4.1. It was discovered that titles in SemEval Spoiling Dataset and Kaggle-2 datasets are longer than in Kaggle-1 dataset. Titles in Kaggle-1 have fewer special characters than titles in other datasets. Clickbait titles in Kaggle-1 dataset are characterized by greater number of capital letters than clickbaits in other datasets.

Table 4.1: Character composition analysis of clickbait detection datasets

Dataset	Clickbait Status	capital letters		small letters		special chars		blank chars		numerical		letters	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Kaggle 1	Non-Clickbait	4.6	2.5	38.8	12.3	0.7	1.0	7.2	2.2	0.5	1.2	43.4	12.3
	Clickbait	9.7	2.9	35.3	10.3	0.8	1.1	8.9	2.7	1.0	1.4	45.0	12.5
Kaggle 2	Non-Clickbait	7.0	7.6	54.3	77.6	1.8	2.5	11.3	15.9	0.6	1.5	61.3	83.7
	Clickbait	7.5	8.1	52.3	88.1	1.7	2.7	11.5	18.4	0.8	1.7	59.8	94.4
Clickbait Challenge	Non-Clickbait	7.4	9.9	59.4	116.7	1.9	3.2	12.3	24.1	0.7	1.8	66.8	125.4
	Clickbait	7.7	9.6	54.5	113.1	1.8	3.4	12.0	23.7	0.8	1.9	62.2	121.0

The number of various characters in each dataset is presented in Figure 4.2. There is a similar distribution in 'Kaggle 2' and 'Clickbait Challenge'. The Duplicates check showed that there were duplicate titles in the mentioned datasets. The 'Kaggle 1' dataset has fewer outliers than other datasets by IQR criteria (see Section 4.2.3).

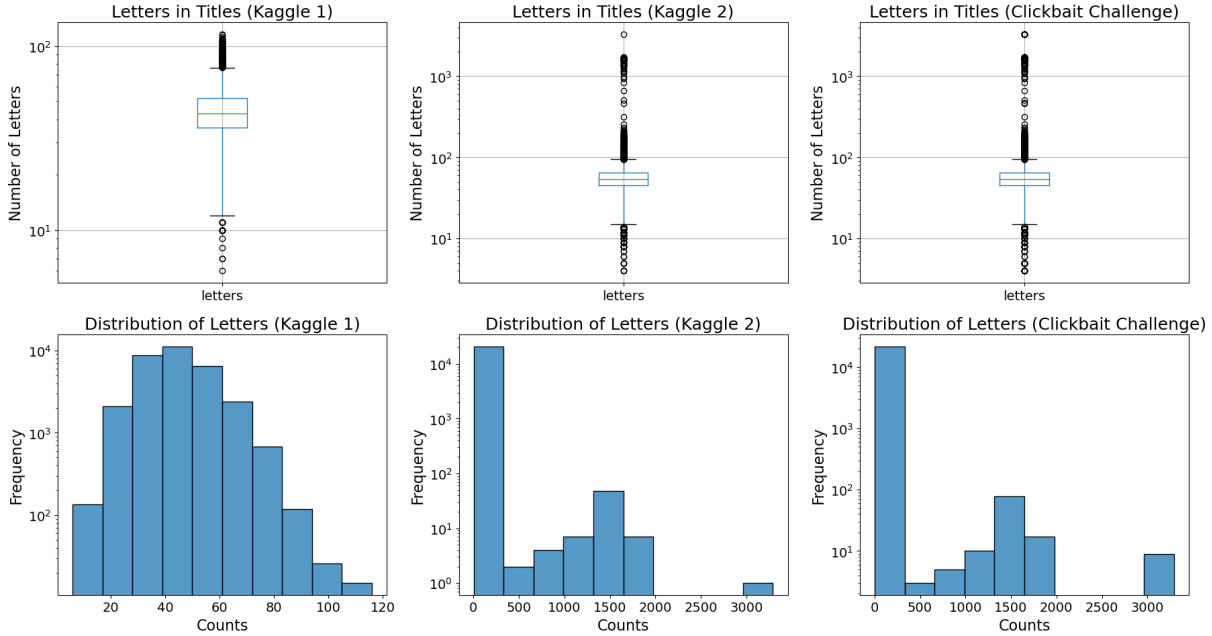


Figure 4.2: Length of titles in detection datasets

The average token length of article titles was checked as well. In this analysis we assumed that the token is a lemmatized word. It is similar in all datasets. The results are displayed in Table 4.2.

Table 4.2: Average token length for titles in clickbait detection datasets

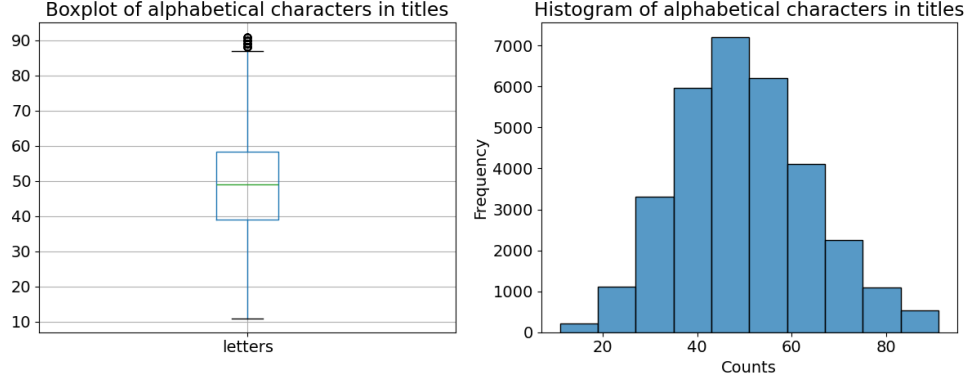
Dataset	Average Value
Kaggle-1	5.73
Kaggle-2	5.74
Clickbait Challenge	5.74

## 4.2. Insights and visual analysis

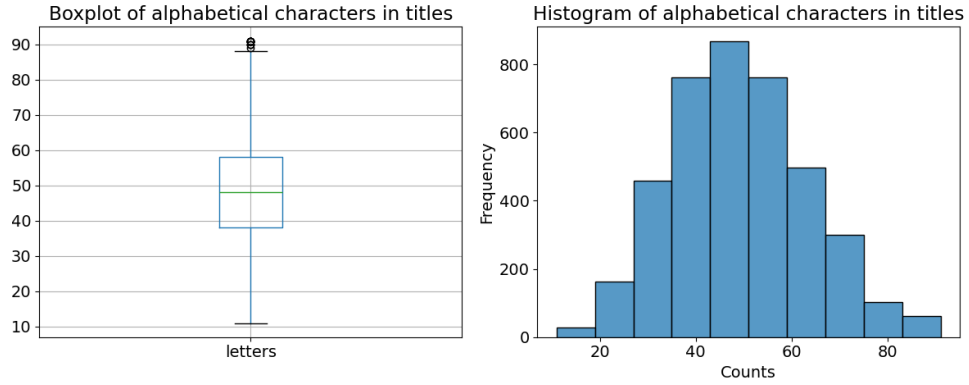
### 4.2.1. Merging data into final dataset for clickbait detection

In the next step, datasets were merged. It's important to note that the resulting dataset is imbalanced, with over 65% non-clickbait articles. Class balance is crucial for classification

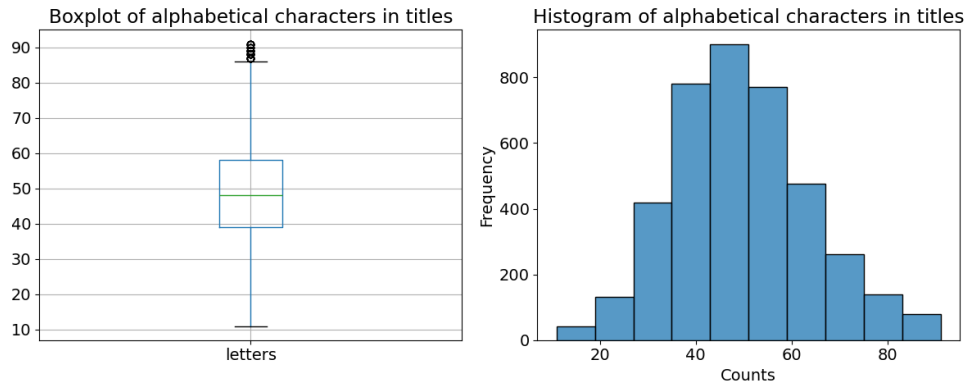
tasks, and most classifiers suffer from the curse of learning from an extremely imbalanced training dataset. Thus, we created a balanced sample by randomly selecting 20,000 clickbait titles and 20,000 non-clickbait titles. This helped address the imbalance [14]. We call this dataset **dataset\_title\_detection\_balanced\_raw**. The preprocessed version of this dataset is called **dataset\_title\_detection\_balanced\_refined**. Preprocessing is described in Section 4.



(a) Train dataset (80% of data)



(b) Test dataset (10% of data)



(c) Validation dataset (10% of data)

Figure 4.3: Distribution of alphabetical characters in titles after train/test/val split in clickbait detection dataset

After merging, the train-test-val split was done in proportions 80%/10%/10%. From this moment, all analyses are performed only on the train set. This is because the analysis of the merged dataset affects the rest of the project and we use found insights in the model building. Thus, we split the data to avoid data leakage [15].

Our team checked the distribution of the clickbait and non-clickbait articles in the 'train', 'test' and 'val' datasets. They are similar, which is shown in Figure 4.3. Similar distribution allows for training and testing models without any doubts that the results are interfered by the dissimilar distribution of clickbaits in sets.

#### 4.2.2. Lexical characteristics

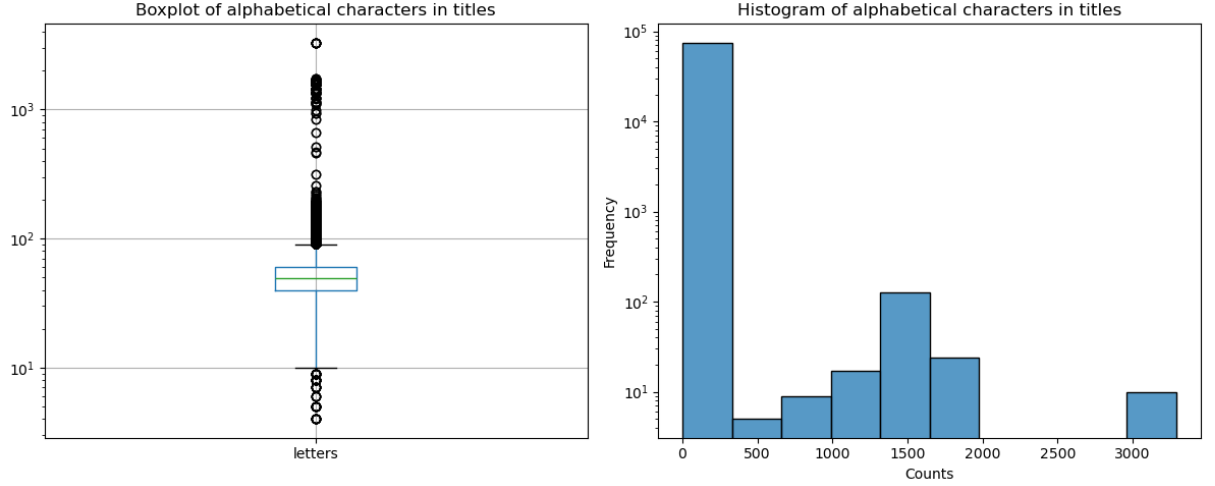
Firstly, the length of the text data was checked. For better understanding, some measurements were applied, such as capital and small letters, special characters, blank signs, and numerical characters. The results are presented in Table 4.3.

Table 4.3: Merged dataset: descriptive statistics by clickbait status

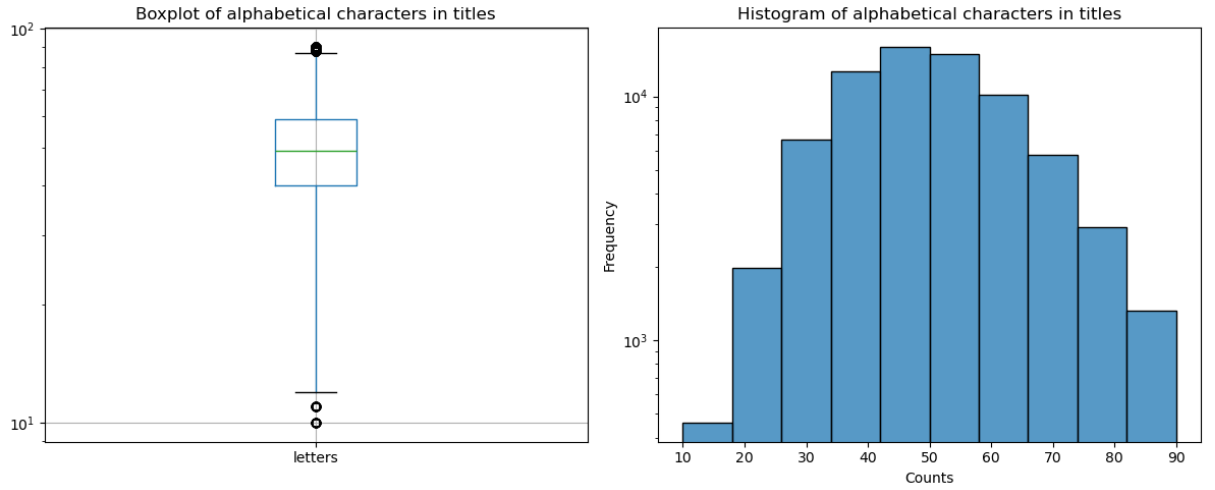
	<b>capital letters</b>		<b>small letters</b>		<b>special chars</b>		<b>blank chars</b>		<b>numerical</b>		<b>letters</b>	
<b>title</b>	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
<b>With Outliers</b>												
non-clickbait	7.2	8.8	56.8	99.0	1.8	2.9	11.8	20.4	0.6	1.6	64.0	106.5
clickbait	8.2	6.9	46.0	73.3	1.4	2.4	10.5	15.3	0.9	1.6	54.2	78.5
<b>Without Outliers</b>												
non-clickbait	6.7	4.3	48.2	13.5	1.6	1.5	10.0	3.0	0.5	1.3	54.9	14.0
clickbait	8.0	4.4	41.9	13.8	1.3	1.4	9.7	3.1	0.9	1.5	49.9	14.8

## 4.2. INSIGHTS AND VISUAL ANALYSIS

Character analysis revealed that there are more capital letters in clickbaits. There are also more numerical signs, but titles are shorter. The number of blank characters and special characters is the same. For better understanding, the number of alphabetical characters in titles was visualized by boxplots and histograms in Figure 4.4.



(a) Length of titles with outliers in a 'log' scale



(b) Length of titles without outliers in a 'log' scale

Figure 4.4: Length of titles with texts classified as outliers by the IQR criterion (see Section 4.2.3)

During the further capitalization analysis it turned out that there are 14 titles with more capital than small letters in merged dataset (the total number of titles is 40 000). Among them only 9 were clickbaits so we excluded the hypothesis that more capital than lowercase characters in a title indicates a clickbait title.

### 4.2.3. Outliers

To identify extreme data points, two techniques described below were considered: Z-score and Interquartile Range (IQR). Z-score measures the distance in terms of how many standard deviations a value is above or below the mean value. As an outlier for Z-score method, data point which distance is more than 3 standard deviations from the mean is assumed. The Z-score assumes a normal distribution since the calculation relies on the mean and standard deviation. For skewed data, IQR is more effective [16]. It is ranged between the first quartile and the third quartile in a dataset capturing 50% of values. For skewed distributions, the mean and standard deviations are not reliable indicators. Here is the overview of the two methods:

- **Z-score Criterion:**

- Defines an outlier as any data point that lies more than a specified number of standard deviations (often 3) away from the mean.
- Suitable for data that are normally distributed, as it assumes a symmetric distribution around the mean.
- Formula: A data point  $x$  is an outlier if  $|z| > 3$ , where  $z = \frac{x-\mu}{\sigma}$ , and  $\mu$  and  $\sigma$  represent the mean and standard deviation of the data, respectively.
- **Pros:** Simple to calculate and interpret, especially useful for normally distributed data.
- **Cons:** Sensitive to extreme values and not suitable for skewed or non-normal distributions.

- **IQR Criterion:**

- Defines an outlier as any data point that falls below the lower bound  $Q_1 - 1.5 \times \text{IQR}$  or above the upper bound  $Q_3 + 1.5 \times \text{IQR}$ .
- Suitable for data with a skewed distribution, as it relies on the median and quartiles rather than the mean.
- Formula: A data point is an outlier if it lies outside the interval  $[Q_1 - 1.5 \times \text{IQR}, Q_3 + 1.5 \times \text{IQR}]$ , where  $Q_1$  and  $Q_3$  are the first and third quartiles, and  $\text{IQR} = Q_3 - Q_1$ .
- **Pros:** Robust to non-normal data and less sensitive to extreme values.
- **Cons:** May miss some outliers in normally distributed data, as it does not account for the distribution shape.

The IQR was used for outlier detection since the distribution of the length of titles in our case is skewed. The data point was classified as an outlier when it lies below lower bound ( $Q_1 - 1.5 \times \text{IQR}$ ) or above upper bound ( $Q_3 + 1.5 \times \text{IQR}$ ). Titles classified as outlier, in detecting dataset, were removed. This ensured that the title lengths were closer to a normal distribution. The length of titles now ranges from 11 up to 124 characters (Figure 4.4b).

#### 4.2.4. N-gram and word analysis

While conducting EDA the N-gram analysis was held. An N-gram is a sequence of  $N$  adjacent symbols or the words found in a language dataset in a particular order, where  $N$  is some natural number [17]. In our case, the lemmatization of the words was applied. Reducing words to their core form, such as 'walking'  $\rightarrow$  'walk', allows us to avoid counting similar N-grams separately. Both clickbait articles and no-clickbait articles frequently refer to politics (see Figure 4.5). This is indicated by phrases like "donald trump" or "white house". Clickbaits try to capture reader's attention by including personal topics such as "zodiac sign" or "based zodiac". Clickbaits tend to engage users' emotions with phrases like: "make laugh" or "need to know."

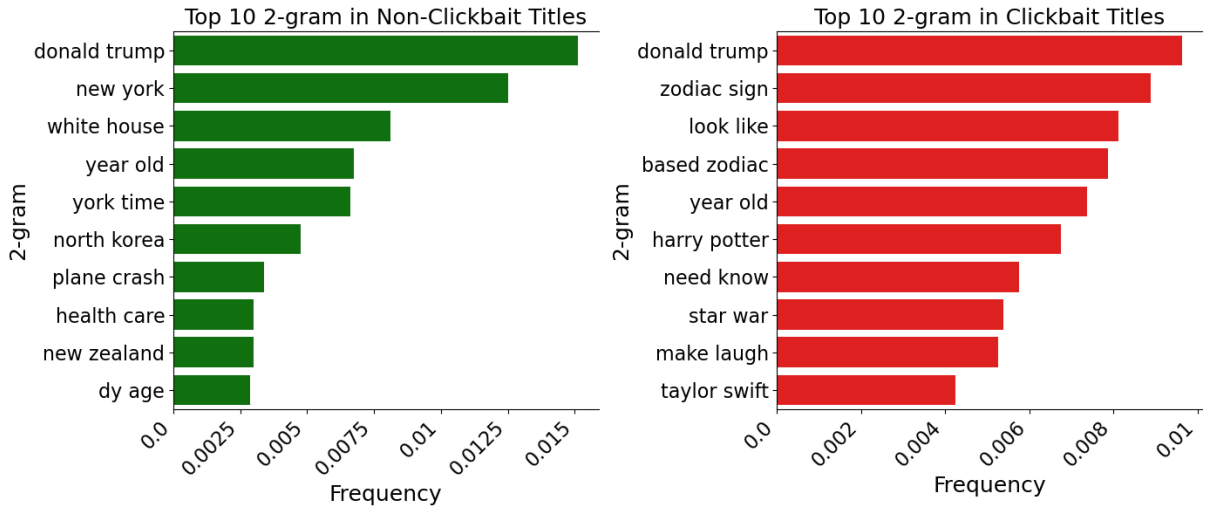


Figure 4.5: Most frequent N-grams in titles

During the text analysis, the high occurrence of stop words in clickbait articles was noted. The mean of stop words in clickbait articles is 4.042, while in valid articles, it is 2.612. The most common stop words are shown below in Figure 4.6.



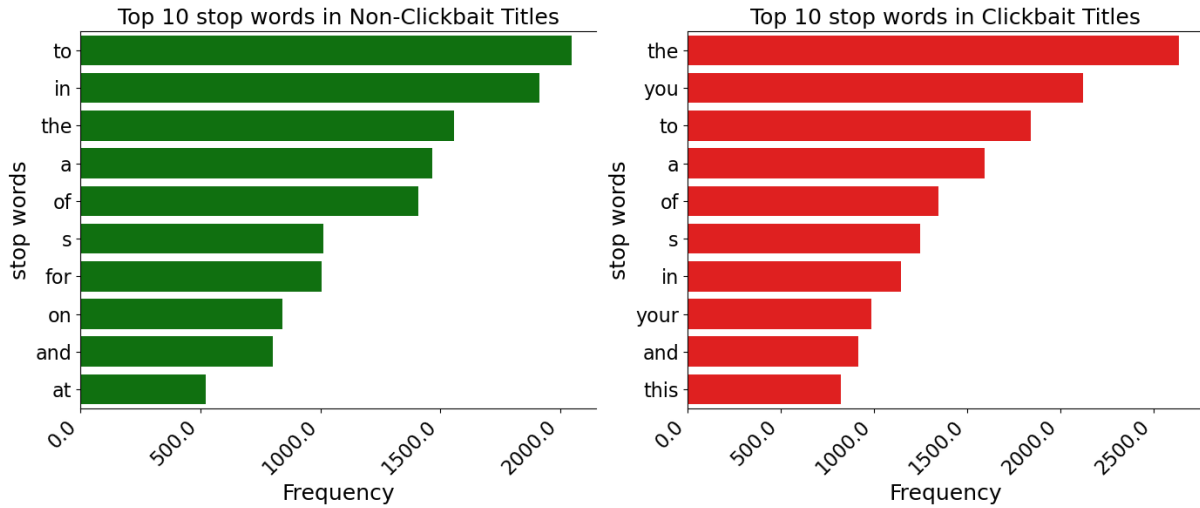


Figure 4.6: Most frequent stop words in titles

On average, there are 1.4 more stop words per article in clickbait titles than in non-clickbait titles. In clickbaits, stop words like "you/yours" occur more often, which may indicate persuasive language.

#### 4.2.5. Named entity recognition

Another applied technique was Named Entity Recognition [18]. This method aims to localize and classify named entities in text. For calculating this metric, we used spaCy library [19]. Certain categories, such as person, organization, location, etc., were defined prior to the algorithm application. This analysis showed that some organizations, places or people are more frequently used by clickbaits or non-clickbaits.

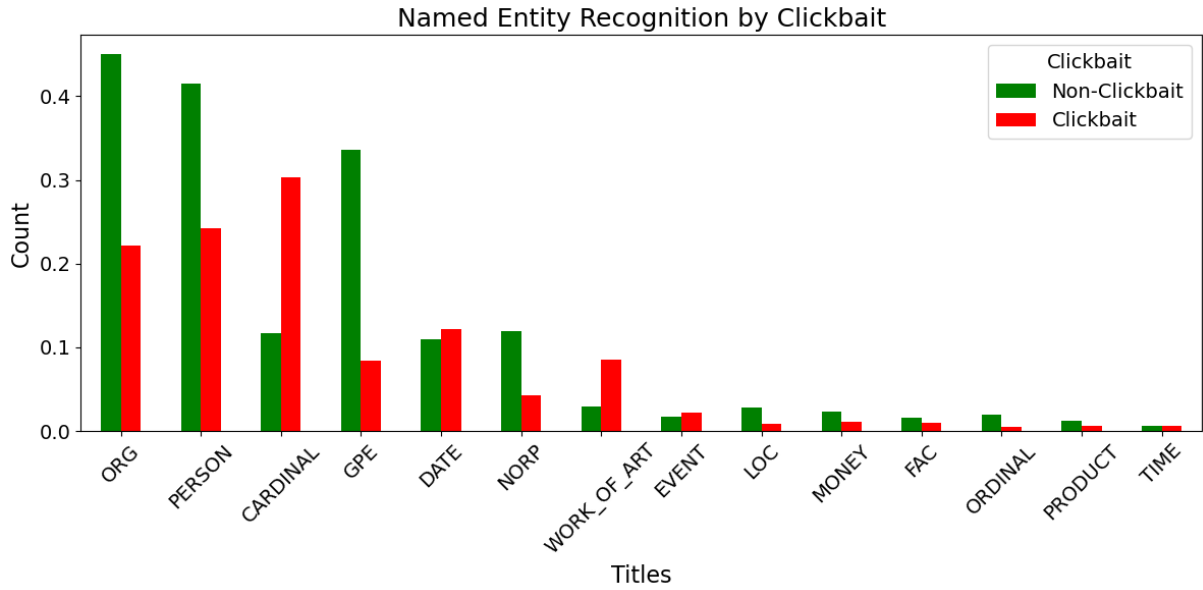


Figure 4.7: Named Entity Recognition in titles

In Figure 4.7, it is noticeable that the occurrence of CARDINAL, DATE, WORK\_OF\_ART supports clickbaits. ORGANIZATION, PERSON, GPE, NORP, and LOCATION are more frequently used in valid articles. In Table 4.4, you can find more detailed descriptions of the mentioned categories.

Table 4.4: Named entity recognition categories

No.	Entity	Description	Example
1	ORG	Organizations — organizations such as companies, institutions, or agencies.	Apple, UN
2	PERSON	Person — individual people or characters.	Barack Obama, Elon Musk
3	CARDINAL	Numerals that do not fall under other numeric types.	three, 1,000
4	GPE	Geopolitical Entities — geographical or political regions such as countries, cities, or states.	USA, Berlin
5	DATE	Absolute or relative dates or periods.	July 4th, next week
6	NORP	Nationalities, religious or political groups.	American, Christian, Democrat
Continued on next page			

No.	Entity	Description	Example
7	WORK_OF_ART	Titles of books, songs, etc.	The Mona Lisa
8	EVENT	Named events.	Olympics, World War II
9	LOC	Locations that are not geopolitical entities, like mountain ranges, bodies of water, etc.	Mount Everest, Pacific Ocean
10	MONEY	Monetary values, including unit.	\$500, €20 million
11	FAC	Facilities — buildings, airports, highways, bridges, etc.	Eiffel Tower, JFK Airport
12	ORDINAL	First, second, etc.	first, third
13	PRODUCT	Objects, vehicles, foods, etc., that are products.	iPhone, Boeing 737
14	TIME	Times smaller than a day.	2 PM, morning

#### 4.2.6. Sentiment analysis

The Sentiment Analysis [20] technique was applied to determine if the emotional tone of the message is positive, negative, or neutral. This was shown by polarity measurement which is explained below. Apart from that, the subjectivity of the text was calculated. Both polarity and subjectivity are calculated by averaging the respective scores of each word in the text [21].

- **Polarity** is given by:

$$\text{polarity} = \frac{\sum_{i=1}^n w_i \cdot p_i}{\sum_{i=1}^n w_i}$$

Where:

- $w_i$  is the weight of word  $i$  in the text.
- $p_i$  is the predefined polarity score of word  $i$ , ranged between  $[-1.0, 1.0]$ . Values closer to  $-1.0$  are negative, and closer to  $1.0$  are positive.
- $n$  is the total number of words in the text.

- **Subjectivity** is calculated similarly:

$$\text{subjectivity} = \frac{\sum_{i=1}^n w_i \cdot s_i}{\sum_{i=1}^n w_i}$$

Where:

- $w_i$  is the weight of word  $i$ .

### 4.3. CLICKBAIT SPOILING DATASET

- $s_i$  is the predefined subjectivity score of word  $i$ , where a value closer to 0 represents objectivity, and closer to 1 represents subjectivity.
- $n$  is the total number of words in the text.

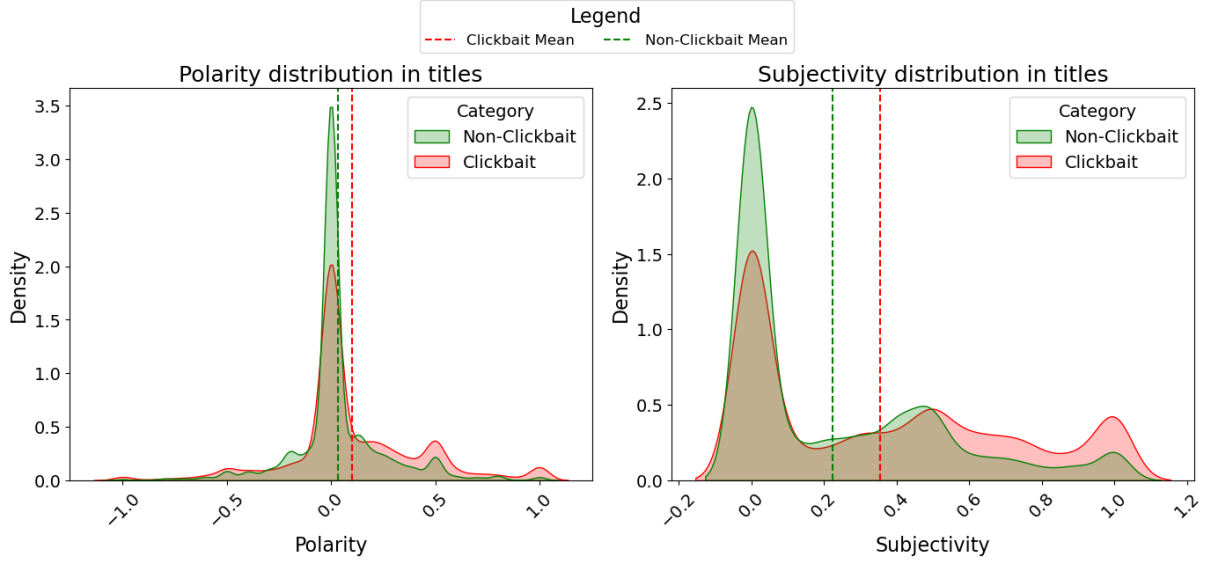


Figure 4.8: Sentiment analysis in titles

In Figure 4.8, it is shown that both clickbait and non-clickbait articles have polarity centered around 0. However, clickbait articles distribution has heavier tails. Clickbait articles have greater values for subjectivity than non-clickbait. Therefore the mean of subjectivity for clickbait is closer to 1 than for non-clickbait. It shows that the semantics of clickbait titles is extreme more often than in credible titles. A text might be either strongly negative or positive. Clickbait titles are also more subjective.

### 4.3. Clickbait spoiling dataset

For the SemEval Spoiling dataset preprocessing was required. At the first glance, there were special characters in the beginning and ending of each string in columns: 'targetTitle', 'humanSpoyer', 'spoiler'. Special characters were removed.

Then the 'NaN' and empty values were detected (see Table 4.5), mainly in the column 'humanSpoyer'. Rows that contained meaningless values were removed. After checking the language of the data with the 'langid' library [22], it turned out that we have only English articles. There were no duplicates.

Table 4.5: Null counts for the SemEval Spoiling Dataset dataset

Column Name	Number of Nulls
targetTitle	2
targetParagraphs	0
humanSpoiler	640
spoiler	0
spoilerPositions	0
tags	0

The analysis of the distribution of spoiler types showed that the most common spoiler tag is 'phrase' (see Table 4.9). The least common spoiler type is 'multi'. Examples of such spoilers are presented in Table 4.6.

Table 4.6: Spoiler examples by Tag in 'tags' column

tags	targetTitle	humanSpoiler	spoiler
passage	Wes Welker Wanted Dinner With Tom Brady, But Patriots QB Had A Better Idea	They Threw A Football	how about that morning we go throw?
phrase	Just how safe are NYC's water fountains?	They are safe and meet guidelines for water quality	The Post independently tested eight water fountains in New York City's most frequented parks, and found that all met or exceeded the state's guidelines for water quality
multi	Taste test: 29 hot sauce bottles, ranked	Its Sriracha, its always sriracha	Sriracha Hot Chili Sauce, Franks RedHot Original Sauce, Cholula Chili Garlic, Louisiana Hot Sauce, CaJohns Bourbon Infused Chipotle-Habañero Sauce

4.3. CLICKBAIT SPOILING DATASET

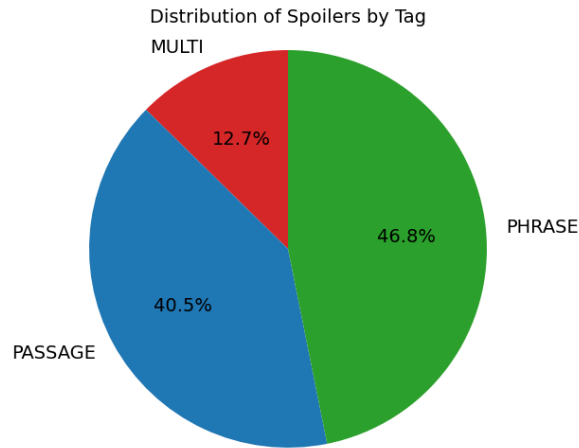


Figure 4.9: Distribution of Spoiler type by Tag

The text analysis showed in table 4.7 of spoilers by used tag showed that the longest spoilers are tagged as 'multi' while the shortest are 'phrase'. There are no other surprising observations coming from the mentioned analysis.

Table 4.7: Spoiling dataset: descriptive statistics by spoiler tag

	capital letters		small letters		special chars		blank		numerical		letters	
spoiler tag	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
multi	4.0	5.0	53.5	50.6	2.9	3.6	10.7	12.1	0.7	2.0	57.5	53.0
passage	2.9	3.1	47.4	38.5	1.8	2.0	9.6	8.4	0.4	1.5	50.2	39.5
phrase	2.8	2.4	26.4	30.2	1.4	1.8	4.9	6.9	0.5	1.9	29.2	31.1

## 5. Informativeness measures

In this section, we provide information about what informativeness measures we used and how we utilized them.

Informativeness can be described as the degree to which the text contains relevant and useful information. Because informativeness is subjective, as it varies across different domains and depends on the target audience, it is tough to measure directly [23]. Still, we can estimate it by using other linguistic characteristics such as readability, lexical richness, difficulty of the text, and usage of specific words and structures, which can help to calculate perceived informativeness.

### 5.1. Related work

This approach can be found in work by Yu et al. [24], where they perform multimodal clickbait detection. Alongside visual extractor for thumbnails and statistical extractor for user profiles, they use textual and linguistic extractors on headlines and main content. Linguistic extractor aims to capture patterns and writing styles using various features, including the disparity between text and title, title sentiment, and lexical analysis of title, within what count-based vector is designed to quantify features such as usage of numbers, capital letters, punctuation marks, different types of verbs, hedges, and typical clickbait vocabulary.

Another work incorporating informativeness measures was done by Reisenbichler et al. [25] and is about natural language generation for content marketing. They generate SEO content using a fine-tuned GPT-2 model. Each generated text is assessed by a quality score based on five measures: overall topic treated in the content (mean similarity between generated text and top T results), keyword integration (similarity between top 10 most frequent words), content uniqueness (fraction of unduplicated n-grams), text naturality (similarity between generated text and top search results based on naturalness measures) and readability (47 different readability metrics).

## 5.2. Our approach

In Table 5.1, we present metrics that were tested. Some of those measures were already discussed and initially inspected in EDA. However, in this section, they are analyzed in detail and used for building models. Every measure was compared against the target variable. We analyzed box plots and calculated means and other statistics, each for clickbait and non-clickbait titles. After finding insights and differences, we used the measures as features to build machine learning models.

Table 5.1: Informativeness measures used

No.	Name	Description
1	Character count	Number of characters, including whitespaces
2	Word count	Number of words
3	Mean word length	Average length of words
4	Common words ratio	Percentage of stop words
5	Capital letters ratio	Percentage of uppercase characters
6	Capital words count	Number of fully capitalized words, excluding one-letter words
7	Punctuation ratio	Percentage of characters which are punctuation or special characters
8	Bait punctuation count	Punctuation related to bait titles: !" (?#
9	Non-bait punctuation count	Punctuation related to non-bait titles: \$%&,,:- /
10	Numbers count	Calculates the number of numbers (not digits)
11	Pronouns	Number of pronouns e.g., "she", "his", "you"
12	2nd person pronouns	Number of words "you", "your", "yours", "yourself", "yourselves"
13	Superlatives ratio	Percentage of adjectives and adverbs which are in superlative form
14	Speculatives usage	Number of speculative words e.g. "may", "might", "possibly"
15	Bait phrases usage	Usage of words and phrases strictly connected to clickbaits e.g., 'tweets' or 'hilarious'
16	Similarity score	Similarity of title and article body, based on cosine similarity between embeddings generated by Word2Vec model
17	Polarity score	Polarity of the text, obtained with TextBlob library
18	Subjectivity score	Subjectivity of the text, obtained with TextBlob library
19	TTR	Type-token ratio, ratio of unique words (here we call them "types") to total number of words (tokens), lexical diversity measure



20	CTTR	Corrected type-token ratio, $CTTR = \frac{t}{\sqrt{2n}}$ where $t$ - types, $n$ - tokens
21	Maas Index	Lexical diversity index, $a^2 = \frac{\log n - \log t}{\log^2 n}$ where $t$ - types, $n$ - tokens
22	HD-D	Hypergeometric distribution diversity, measures lexical diversity, average TTR on random samples of 42 words from the text
23	FRES	Flesch reading-ease score, measures difficulty of the text, $FRES = 206.835 - 1.015 \frac{words}{sentences} - 84.6 \frac{syllables}{words}$ , the higher the easier
24	FKGL	Flesch-Kincaid grade level, text difficulty measure, reflects US grade level needed to understand the text quite easily, $FKGL = 0.39 * \frac{words}{sentences} + 11.8 \frac{syllables}{words} - 15.59$ , the higher the harder
25	ARI	Automated readability index, text difficulty measure, US grade level necessary to comprehend the text, $ARI = 4.71 * \frac{characters}{words} + 0.5 \frac{words}{sentences} - 21.43$

We start off with basic measures such as word count, character count, average word length, and common words ratio. Following the approach of Yu et al. [24] we include clickbait related features such as usage of punctuation marks, numbers, hedges (speculatives) and special baiting vocabulary. The baiting vocabulary includes obscene words (e.g. "sexy", "nudes"), slang words (e.g. "OMG") and media indicators (e.g. "TWEETS", "VIDEO"). Based on our own experience and EDA, we also propose other measures - the percentage of superlative adjectives, pronoun usage, and sentiment score. What's more, we added the similarity between the headline and the article body using cosine similarity and vectors generated using Word2Vec model. It is important to note that measures 8,9,12,15 (indices in Table 5.1) were partially designed based on EDA.

We also incorporate lexical diversity measures TTR and CTTR. Those metrics calculate the ratio of unique words to the total number of words. Because those two metrics are text-length sensitive, we tested also more stable ones [26]: Mass index and HD-D. HD-D is calculated based on random samples of 42 words from the text. It calculates the probability of the token appearing in the sample for each unique token in the text [27]. Because of that, HD-D requires a minimum of 42 words in the text, so it can be applied only to the article body. The last metrics we used are FRES, FKGL, and ARI, which are text difficulty measures.

### 5.3. Analysis

Analysis was conducted on the train set. For word tokenization, we used the `word_tokenize` function from the NLTK library. For measures 3-5, 19-22 (indices in Table 5.1) we removed

### 5.3. ANALYSIS

punctuation before tokenizing.

#### 5.3.1. Title analysis

Each measure was applied to all titles from the dataset. For each measure, we have analyzed distributions using violin plots (examples can be seen in Figure 5.1) and checked statistics (mean, median, standard deviation, quantiles) against the clickbait variable. In Table 5.2, we provide only means, both for clickbait headlines and non-clickbait headlines. The *ratio* column represents the average measure value for clickbait titles divided by the average measure value for non-clickbait titles.

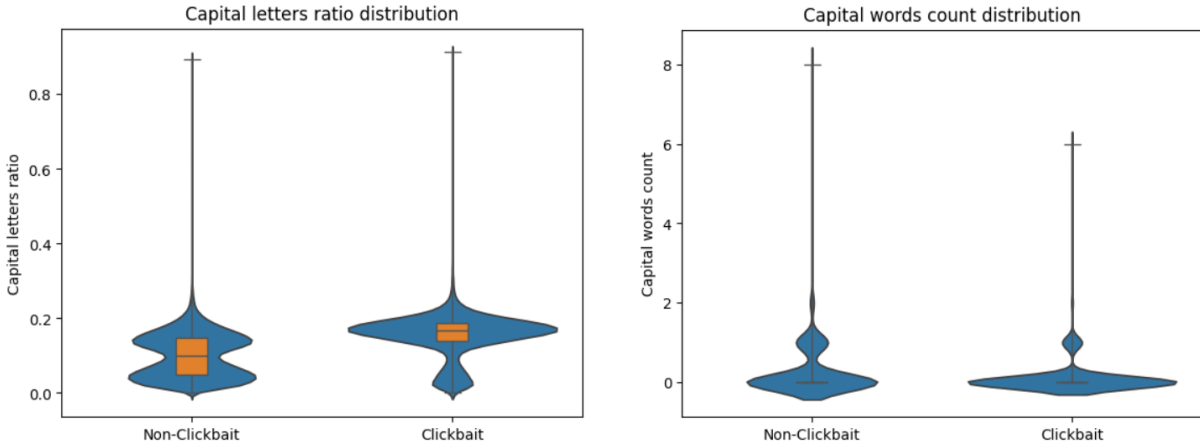
Table 5.2: Measures averages for clickbait and non-clickbait titles

No.	Measure	Mean (non-clickbait)	Mean (clickbait)	Ratio
1	Character Count	61.77	58.85	0.95
2	Word Count	9.97	10.3	1.03
3	Mean Word Length	5.2	4.72	0.91
4	Common Words Ratio	0.23	0.35	1.51
5	Capital Letters Ratio	0.1	0.15	1.54
6	Capital Words Count	0.24	0.11	0.43
7	Punctuation Ratio	0.02	0.02	0.86
8	Bait Punctuation Count	0.07	0.25	3.78
9	Non-Bait Punctuation Count	0.71	0.35	0.49
10	Numbers Count	0.17	0.4	2.39
11	Pronouns	0.23	0.77	3.40
12	2nd Person Pronouns	0.03	0.4	14.62
13	Superlatives Ratio	0.02	0.05	2.51
14	Speculatives Usage	0.05	0.07	1.63
15	Bait phrases Usage	0.23	0.89	3.81
16	Polarity Score	0.03	0.1	2.99
17	Subjectivity Score	0.22	0.36	1.60
18	TTR (Type-Token Ratio)	0.99	0.98	1.00
19	CTTR (Corrected TTR)	2.19	2.22	1.01
20	Maas Index	<0.01	<0.01	1.28
21	FRES (Reading Ease)	65.29	76.33	1.17
22	FKGL (Grade Level)	6.56	5.13	0.78
23	ARI (Readability Index)	8.93	6.61	0.74

There are no substantial differences in character and word count. Clickbaits have higher

common words percentage and smaller average word length. It makes sense as stop words are often short.

Measures prepared specifically for analyzing clickbaits are all higher for clickbait headlines (capitals ratio, pronouns, superlatives, speculatives, bait phrases). What's more, the usage of numbers is also higher. The unexpected observation is that the average number of fully capitalized words is higher for non-clickbait titles; the distribution can be seen in Figure 5.1. There is no difference in punctuation, but after splitting it (based on literature and EDA) into bait and non-bait punctuation, the differences are clearly visible.



(a) Distributions of capital letters ratio

(b) Distributions of fully capitalized words count

Figure 5.1: Distribution of metrics related to capital letters

Sentiment analysis showed that clickbait headlines are more subjective. Also, the polarity is more varied in clickbaits than in non-clickbait headlines and, interestingly, more positive than negative on average.

Clickbaits are also easier to read, which is proofed by values of FRES, FKGL and ARI. It is hard to calculate lexical diversity for titles, as they are super short. However, lexical diversity analysis of all titles merged together showed that clickbaits are less lexically varied (see Table 5.3).

Table 5.3: Lexical diversity of merged clickbait titles and merged non-clickbait titles

Measure	Non-clickbait	Clickbait
TTR	0.133	0.097
CTTR	37.954	28.085

### 5.3.2. Text body analysis

For body analysis, all metrics were changed to ratio instead of "usage" by dividing counts by either the number of characters (for metrics concerning individual characters, e.g., bait punctuation) or the number of words (for metrics analyzing whole words, e.g., fully capitalized words count). All titles are relatively short, and we are curious if some phrase was used or not. However, in article bodies we need to check the ratio, as the articles are long and differ much in length.

Table 5.4: Measures averages for clickbait and non-clickbait articles

No.	Measure	Mean (Non-Clickbait)	Mean (Clickbait)	Ratio
1	Number of Words	554.36	598.37	1.08
2	Number of Characters	3323.43	3555.14	1.07
3	Mean Word Length	4.75	4.67	0.98
4	Common Words Ratio	0.402	0.406	1.01
5	Capital Characters Ratio	0.039	0.045	1.15
6	Capital Words Ratio	0.013	0.016	1.27
7	Punctuation Ratio	0.028	0.029	1.05
8	Bait Punctuation Ratio	0.0031	0.0028	0.92
9	Non-Bait Punctuation Ratio	0.022	0.0234	1.06
10	Numbers Ratio	0.016	0.02	1.22
11	Pronouns Ratio	0.051	0.06	1.17
12	2nd Person Pronouns Ratio	0.004	0.012	2.94
13	Superlatives Ratio	0.00012	0.00017	1.41
14	Speculatives Ratio	0.0051	0.0064	1.25
15	Bait Words Ratio	0.01	0.02	1.18
16	Similarity	0.75	0.72	0.96
17	Polarity	0.09	0.12	1.34
18	Subjectivity	0.42	0.45	1.08
19	Type-Token Ratio (TTR)	0.56	0.56	1.00
20	Corrected TTR (CTTR)	8.01	8.17	1.02
21	Maas Index	0.016	0.016	1.00
22	HDD	0.85	0.84	0.99
23	FRES (Reading Ease)	60.65	63.36	1.04
24	FKGL (Grade Level)	9.76	9.19	0.94
25	ARI (Readability Index)	12.37	11.81	0.95

Based on Table 5.4, we could argue that articles with clickbait headlines are longer on average, but it is probably not a rule. Additionally, the average word length in clickbait articles is slightly

shorter, and they tend to use a marginally higher number of common words.

When it comes to capitalization, the difference is visible: clickbait articles contain more capital letters and more fully capitalized words. However, in terms of punctuation, there is almost no difference at all. The "bait punctuation" feature was specifically designed for analyzing titles, and in the body of the article, the relationship is slightly reversed. Similarly, for "non-bait punctuation" there are no significant differences in the means or medians in opposition to what we observe in headlines. Clickbait articles tend to use more numbers, though the difference is less visible than in headlines.

Pronouns are used more frequently in articles with clickbait headlines, especially second-person pronouns, as they appear almost three times as often in clickbait articles as in non-clickbait ones. Superlatives are so rare that they are not worth deeper analysis, while speculative language appears just slightly more often in clickbait articles. Since this feature is also title-specific, it does not show substantial differences in the article bodies.

The similarity score between text and title is bigger for non-clickbait articles, but the difference is really small. Yet again, the polarity is more varied and a little bit more positive for clickbait headlines. Clickbait articles are also more subjective.

Lexical diversity is nearly identical between the two types of articles, with a nod to non-clickbait articles. Clickbait articles are also slightly easier to read, though some outliers exist because some articles are very short.

To summarise, most relationships between the measures and clickbaits are similar as in the headlines, but the differences are generally smaller and weaker in the article bodies. The only exception is punctuation, where the relationship is reversed.

#### 5.4. Custom measure

In our work we also created our own *baitness* measure, similar to what was done by Reisenbichler et al. [25]. This score measures eye-catchingness, usage of curiosity-generating techniques, sentiment and text difficulty. It is a combination of other metrics, which selection depended on the earlier-mentioned analysis. In the equations we use custom function  $f$  which cuts value to  $[0,1]$  range:

$$f(x) = \min(1, \max(0, x))$$

The **eye catchingness** is defined as:

$$\text{EyeCatch} = f\left(\frac{\text{BaitPunctCount} + 3 \cdot \text{CapitalsRatio} + \text{NumbersCount}}{3}\right)$$

**Content curiosity** is measured as:

$$\text{Curiosity} = \sqrt{f\left(\frac{2\text{ndPronouns} + 2 \cdot \text{SuperlativesRatio} + \text{SpeculativesCount} + \text{BaitWordsCount}}{4}\right)}$$

To measure **sentiment** we multiply polarity and subjectivity:

$$\text{Sentiment} = \sqrt{|\text{PolarityScore}| \cdot \text{SubjectivityScore}}$$

**Difficulty of text** is represented by FRES and stop words ratio.

$$\text{TextEase} = \frac{f(\text{FleschScore}/100) + f(\text{CommonWordsRatio} \cdot 1.5)}{2}$$

And the whole metric is an average of all of them:

$$\text{Baitness} = \frac{\text{EyeCatch} + \text{Curiosity} + \text{Sentiment} + \text{EaseOfText}}{4}$$

The eye-catchingness consists of text elements that naturally draw attention, such as the use of capital words, numbers, and special punctuation (e.g., exclamation marks). The capital letters ratio is multiplied by 3, because in 99% of the titles, this metric is below 0.3 (see Figure 5.1). Scaling ensures that values are more evenly distributed between 0 and 1.

In content curiosity, we measure usage of phrases and techniques, which aim is to create a curiosity gap [24] and lure user into reading the article. We also include usage of second-person pronouns, superlatives ratio and count of speculatives, as we think they fit into this curiosity-generating category. Superlatives ratio is multiplied by 2, as it is the only measure not based on direct count, thus is smaller than other metrics. The overall value is scaled up using the square root to guarantee distribution is more balanced between 0 and 1.

In sentiment, we calculate geometric mean of polarity and subjectivity.

Difficulty of text is represented by FRES divided by 100, due to its values being originally between 0 and 100. Common words ratio is also added and multiplied by 1.5 for the same reason as capital letters ratio in eye-catchingness.

Initial inspection on the test set showed that this metric is good at highlighting the *baitness* of the headline. In Figure 5.2 and Table 5.5, we can see that there is a significant difference in distribution between clickbait and non-clickbait titles.

Table 5.5: *Baitness* measure statistics compared against clickbaits on the test set

Titles type	Mean	Std	Median
Non-clickbait	0.24	0.12	0.22
Clickbait	0.43	0.16	0.44

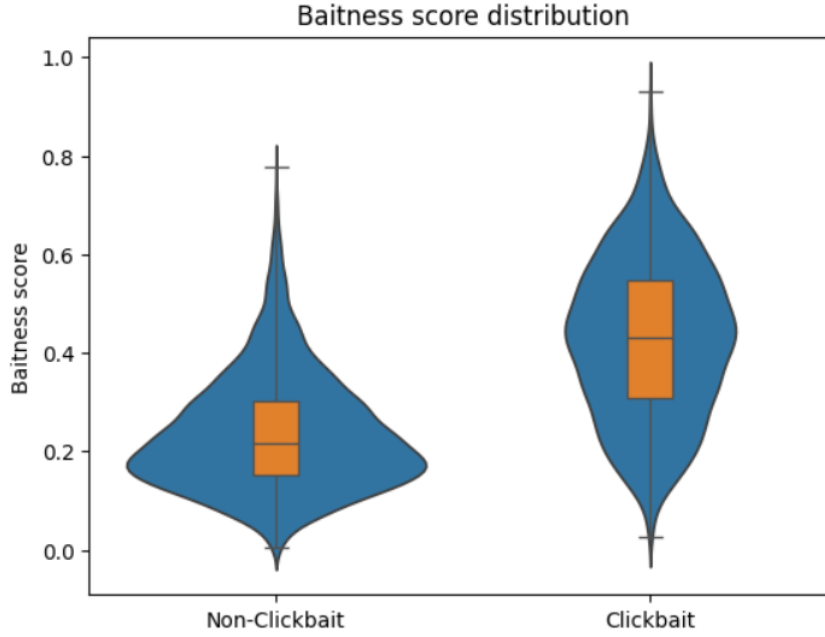


Figure 5.2: Distribution of *baitness* score on the test set

What's more, it can easily be used for clickbait classification task. When setting the threshold at 0.32, we get an F1-score and accuracy at a level of 0.76 and 0.83 AUC score (see ROC curve in Figure 5.3) on the test set. This is a great result, as we can achieve almost as good results as machine learning models by simple rules and measures.

#### 5.4. CUSTOM MEASURE

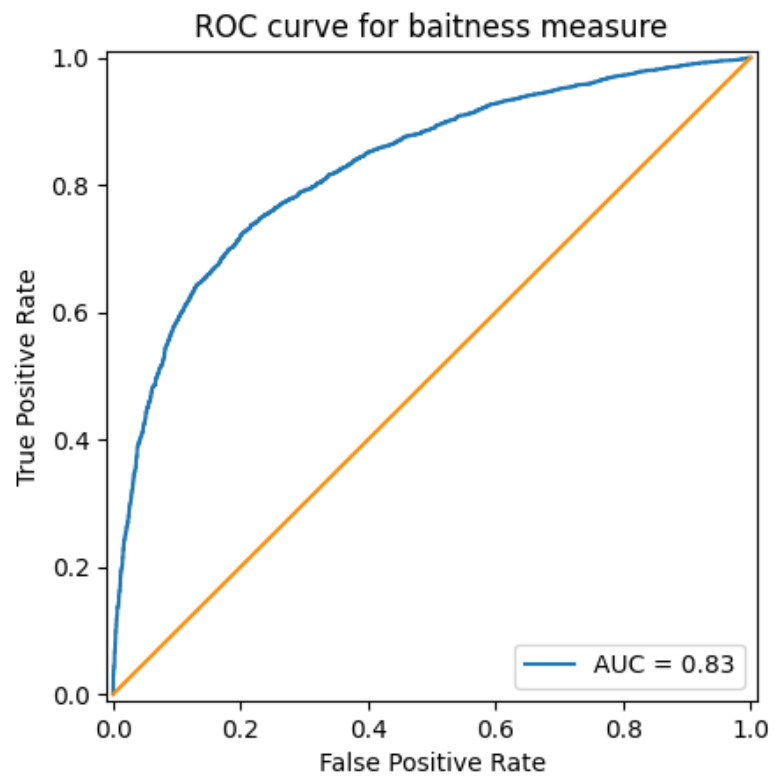


Figure 5.3: ROC curve for *baitness* score used for classification



## 6. Clickbait detection

In this section, we explore the development of machine learning models to detect and classify clickbait headlines. The goal is to distinguish clickbait from non-clickbait headlines based on linguistic, structural, and semantic features. A variety of machine learning methods are tested and evaluated, including traditional methods (e.g., TF-IDF, GloVe) and modern deep learning approaches (e.g. LLM). Before starting the work on the clickbait detection, existing solutions were analyzed.

### 6.1. Related work

*The Clickbait Challenge 2017* was a shared task inviting the submission of clickbait detectors for a comparative evaluation [28]. The best-performing solution, measured by F1-score, employed a neural network architecture with bidirectional gated recurrent units (biGRU). For input of the network, they used a sequence of word embeddings that they got from pre-trained Glove on Wikipedia. Many participants also used LSTM architecture. Interestingly, one team computed a set of 175 linguistic features (informativeness measures) from the teaser message and then applied a random forest algorithm, achieving high performance. Some also used XGBoost, TF-IDF, or logistic regression.

In another study, *Clickbait Detection Using Deep Recurrent Neural Network* [1], the authors describe a browser extension designed to detect clickbaits by analyzing article links. The approach involves creating a whitelist of valid domains and a blacklist of clickbait domains. For this purpose RNN, particularly Gated Recurrent Units (GRUs) were trained. It was compared with Classical Ensemble Methods (CEM), Lure and Similarity for Adaptive Clickbait (LSAC), Convolutional Long Short-Term Memory (C-LSTM). The GRU-based approach outperformed these methods due to its comprehensive analysis of source ratings and multilayered clickbait identification.

A separate study, titled *Clickbait Detection with Style-aware Title Modeling and Co-attention* [29] introduced the SATC (Style-Aware Title Modeling with Co-Attention) method for clickbait detection. This approach leverages stylistic and contextual patterns in titles to improve

performance. SATC is build with multiple components:

- **Content Modeling** - encodes the title and the body of the article as a vector to capture core meaning. Text is broken down into word sequences. Each word is then converted into word embedding. Transformer model is applied to understand the meaning of the surrounding words. Not all words are equally important for content interpretation, so an attention mechanism is applied to prioritize the most relevant information.
- **Style Modeling** - transformers extract the core content representations from the title and body.
- **Interaction Modeling** - captures to what extent the title is related to the body of the article. The model detects instances where specific words in the title have corresponding words in the body, for instance - title: 'restaurants'  $\rightarrow$  body: 'businesses', 'cafes'. For this purpose, the 'multi-head co-attention network' was used.

The final prediction is a weighted sum of scores calculated in dense layers. They come from each module described above. The authors tested many deep learning architectures. CNN, LSTM, ARU-Att, and SATC outperformed DSSM, a method that used handcrafted features. This shows that handcrafted features are not optimal for clickbait detection. GRU-Att, LSDA outperformed CNN and LSTM - methods without attention. The best method overall was SATC, which outperformed all other baseline methods. Authors claim that it was because SATC can capture stylistic patterns in the title and interactions between contexts in the title and body. For training and testing, 2 datasets were used: Clickbait Challenge [30] and FNC (Fake News Challenge) [31].

## 6.2. Our approach

To develop our custom solution, we began by testing multiple approaches, starting with classical methods before transitioning to more modern techniques. A crucial aspect of building machine learning models for NLP is the creation of word and text embeddings.

A word embedding is a numerical representation of words in a vector (can be continuous or binary) space, where words with similar meanings are positioned closer to each other. By mapping each word to a fixed-length vector of real numbers, embeddings capture the context, meaning, and relationships between words in a way that reflects their semantic and syntactic properties. Beyond word embeddings, text embeddings represent larger units, such as sentences or documents (groups of words). These embeddings serve as inputs for machine learning models, enabling them to process language more effectively.

In our thesis, we used the following models for creating embeddings and detecting clickbaits, starting from very simple ones and gradually moving to more complex ones.

- **TF-IDF**, which stands for Term Frequency - Inverse Document Frequency. It is a simple yet effective statistical method that transforms text into numerical representations. It measures the importance of different words in a document by assigning a weight, which is calculated using the following formulas.

$$TF(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

$$IDF(t, D) = \log \left( \frac{N}{|\{d \in D : t \in d\}|} \right)$$

Each word in a document is assigned a weight that is a product of its term frequency and inverse document frequency value.

$$TF\text{-}IDF(t, d, D) = TF(t, d) \cdot IDF(t, D)$$

Where:

- $t$ : The term (word) for which we are calculating the TF-IDF.
- $d$ : A single document (article) in the corpus  $D$ .
- $D$ : The entire collection of documents
- $N$ : The total number of documents in the corpus  $D$ .

The main idea behind this concept is that the weight takes into account how frequently does a word occur in a given document, but if some words occur very often in the entire corpus (e.g. "and", "the", "a", etc...) their score is minimized through IDF term. Then, for a document, we get a vector that represents the importance of each word in the document relative to both the document itself and the entire corpus.

Although somewhat outdated, this algorithm is still recommended in Natural Language Processing in Action, which describes it as a *shallow* NLP machine powerful and useful for many practical applications, such as spam filtering and sentiment analysis [23]. Its simplicity makes it easy to implement, and it remains a valuable baseline for our thesis.

- **Word-embedding models.** These models generate word embeddings that capture the semantic meaning of words. There are two very popular ones, Glove and Word2vec, which were used by us.
  - **Word2Vec.** It is a shallow, two-layer neural network that aims to capture semantic relationships between words and represent them in a continuous vector space (as

embeddings). This algorithm was developed by researchers at Google [32] in 2013 and can be trained by using either two model architectures: continuous bag-of-words (CBOW) or continuous skip-gram architecture. In the first one, the model predicts the current word based on surrounding words, while in the latter, it is the vice versa, so the model uses the current word to predict the surrounding words.

- **TF-IDF weighted Word2Vec.** It is a method to combine word-embedding models with TF-IDF method. It can be done by multiplying word-embedding by its TF-IDF value. Then the value must be normalized to ensure that they are comparable across different words or documents [33]. It is possible to combine these algorithms in different ways, which is mentioned in the following paper [34]. Additionally, the authors also mention that the combination of these two algorithms outperforms each method individually for various tasks.
- **Glove (Global Vector for Word Representation).** Training of Glove is performed on aggregated global word-word co-occurrence statistics from a corpus. It uses matrix factorization. It was launched at Stanford University in 2014, and according to the authors of this algorithm, it outperforms Word2vec, achieving better results faster [35].

Even though both of these word-embedding models (Word2Vec, GloVe) are, as of 2025, outdated and the Transformer-based models are the current state of the art in NLP, they still serve as valuable tools for tasks with limited computational resources or simpler applications [36]. In the paper detailing models [11] submitted for the Clickbait Challenge 2017, two of the top-performing teams used a pre-trained GloVe model (on Wikipedia and further dataset) to create embeddings.

- **Large Language Models (LLMs)/Transformer-based models.** In recent years, LLMs have gained significant popularity, especially following the public release of ChatGPT, which quickly became a viral topic of global discussion. Large Language Models are machine learning algorithms designed to comprehend and generate human language text. In this work, we use the following LLMs: ChatGPT and BERT. The ChatGPT API can not only generate embeddings for a given text but also directly respond to human questions, such as evaluating whether a headline might be clickbait.
- **Tree-based models.** For detection, we mainly use tree-based models, such as Random Forest and XGBoost. Random Forest is a commonly used machine learning algorithm developed by Leo Breiman in the 2000s. It creates a group of decision trees to make predictions.

Each tree is grown using a random selection of data features. At each split within a tree, a small, random set of features is chosen, and the best split is determined based on these features. [37]

XGBoost constructs trees sequentially with a boosting technique that corrects errors from previous trees to enhance performance. It is worth mentioning that out of 29 winning solutions in machine learning competitions featured on Kaggle’s blog in 2015, 17 of them used XGBoost [38]. Also, Random Forest is mentioned as one of the few algorithms handling a very large number of variables without overfitting [39], which would be a good fit for our problem of the curse of dimensionality that we are dealing with in NLP.

### Evaluation metrics

The obvious thing is that to compare the performance of the models mentioned above, one needs to decide on evaluation metrics. In order to assess their effectiveness, we want to use multiple metrics, as each provides unique insights into the model’s performance. For the detection problem, we want to use the following metrics:

- **Accuracy** is the ratio of correctly predicted instances (both positive and negative) to the total number of instances. It is defined mathematically as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

where:

- TP = True Positives
- TN = True Negatives
- FP = False Positives
- FN = False Negatives

- **Precision**, also known as the positive predictive value, measures the proportion of correctly predicted positive instances out of all instances predicted as positive. It is calculated as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall**, also known as sensitivity or true positive rate, measures the proportion of actual positive instances that the model correctly identified. It is calculated as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1-score** is the harmonic mean of precision and recall, providing a balance between the two metrics. It is calculated as:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **ROC-AUC Score** is the area under the ROC curve, which represents the TP rate vs FP rate for every threshold.

These metrics were also used in *Clickbait Challenge 2017* [10]. The primary classification metric that we use is the F1-score because it takes into account both false positives and false negatives.

Apart from quantitative metrics, we evaluate models also subjectively by taking into account their complexity and interpretability. Both of these attributes are hard to measure or define mathematically using simple formulas. However, to assess complexity, one may consider the size of the model (number of parameters), while interpretability can be evaluated by analyzing how effectively the models highlight the features that influence their predictions.

### 6.3. Data preprocessing

Before training the models for clickbait detection problem it is important to note that text preprocessing might improve the accuracy results of text classification. Common preprocessing steps include converting text to lowercase, removing punctuation, or stopwords removal. This aims to reduce the noise and standardize the text while reducing the number of dimensions (the number of unique words occurring in the corpus of a document). This data cleaning process should improve the TF-IDF's, Word2Vec and Glove performance by improving consistency and reducing dimensionality. However, the effectiveness of text preprocessing may vary depending on the particular dataset and task [40].

In the article *The influence of preprocessing on text classification using a bag-of-words representation* [40], the authors tested various combinations of basic preprocessing methods (spelling correction, converting uppercase letters into lowercase letters, HTML tag removal, punctuation mark removal, stopwords removal and reduction of repeated characters) and their impact on ML' model's performance. They conclude that *it is always recommended to perform an extensive and systymatic variety of preprocessing methods*. Unfortunately, some preprocessing methods may even harm the performance. They mention that stopwords removal may sometimes harm performance because they can have discriminative value in specific domains. Moreover, in the paper *The impact of preprocessing on text classification* [41], the main conclusion is that no unique

combination of preprocessing steps provides successful classification results for every domain and language.

Taking everything into account, we applied basic text preprocessing for these three models: TF-IDF, Word2Vec and GloVe and verified if it improved their performance. In the case of large language models (OpenAI, BERT), there is no need to use preprocessed text because the syntax and structure are essential and provide additional information for them.

The process of data cleaning for title-based classification is described below:

1. **Removing non-ASCII characters**
2. **Lowercasing:** Converts all text to lowercase to ensure uniformity across the dataset and reduce the number of unique words in the corpus.
3. **Removing stop words and punctuation:** Removes common stop words (e.g., **a**, **and**, **the**) and punctuation marks. We use the list of stopwords from NLTK package [42].
4. **Lemmatization:** Reduces words to their base or root form (e.g., **running** becomes **run**).

The cleaned version of `dataset_title_detection_balanced_refined` is called `dataset_title_detection_balanced_cleaned`.

To achieve an 80%, 10%, 10% split for the cleaned dataset, we also perform the following steps using the `train_test_split` function from scikit-learn library:

Firstly, we splitted the data into a training set (80%) and a temporary set (20%) by setting `test_size=0.2`. We used the `stratify` parameter to maintain the class distribution.

Nextly, we splitted the temporary set (20%) into validation (10%) and test sets (10%) by using `train_test_split` again, this time with `test_size=0.5`, dividing it evenly. Again, we used the `stratify` parameter to preserve class proportions.

## 6.4. Models based on informativeness measures

Firstly, we used the measures as features to build machine learning models for clickbait detection.

### 6.4.1. Title only

Initially, we tested models using title-based features only. Firstly, we checked the correlation between the features and removed strongly correlated columns (correlation over 0.5), i.e., *number of characters*, *mean word length*, *punctuation*, *pronouns*, *TTR*, *CTTR*, *FKGL*, and *ARI*. All other

variables were standardized, and a logistic regression model was fitted. After initial tests, two more variables were dropped (*speculatives*, *Maas index*) as they were insignificant ( $p > 0.05$ ). The final version of logistic regression model utilized 12 variables: *the number of words*, *common words ratio*, *capitals ratio*, *capital words count*, *bait punctuation*, *non-bait punctuation*, *use of numbers*, *second-person pronouns*, *superlatives*, *bait phrases*, *polarity*, *subjectivity*, and *FRES*. This model achieved pretty good results on the test set, as presented in Table 6.1. Also, all model coefficients were in line with the previous analysis (e.g., the more capital letters, the higher the probability of the headline being a clickbait).

After logistic regression, we tested the random forest classifier. For this model, we used all variables, excluding the correlated ones mentioned in the previous paragraph. It helped us achieve better results than using logistic regression, as F1-score increased from 0.805 to 0.821. Additionally, we checked feature importance, which is available directly in tree-based classifiers. The top 5 features with the biggest importance were *capitals ratio*, *common words ratio*, *usage of second-person pronouns*, *FRES* and *bait phrases*, while the bottom 5 were *bait punctuation*, *capital words count*, *Maas index*, *superlatives* and *speculatives usage*.

Table 6.1: Performance metrics for models with informativeness measures as features

Model	Accuracy	Precision	Recall	F1-score	AUC
Logistic Regression	0.811	0.806	0.805	0.806	0.873
Random Forest	0.823	0.822	0.821	0.822	0.897

#### 6.4.2. Title and body

In order to check whether the article body helps with the predictions, we had to train two models: one with title-based features only and one with additional features based on the article text. However, the only dataset that contains article text is the Clickbait Challenge dataset. That said, firstly, we trained the random forest model only on titles from this dataset and achieved an F1-score of 0.66. After that, we added additional features achieved from article texts, and we got an F1-score of 0.72. It shows that information about the article itself is a value added in a clickbait classification task. It gave us 9% better accuracy and F1-score (see Table 6.2).

There is also one more insight from this experiment. As we can see, model trained and tested only on Clickbait Challenge datasets got a lot worse results than models trained and tested on the whole dataset. This may indicate that the relationships and dependencies within the data in the other datasets are much simpler and more predictable, while in the CC dataset, they are



more complex.

Table 6.2: Performance metrics for Random Forest models on Clickbait Challenge dataset

Features	Accuracy	Precision	Recall	F1-score	AUC
Title	0.657	0.656	0.655	0.656	0.724
Title and body	0.716	0.716	0.716	0.716	0.789

## 6.5. Models based on TF-IDF

This subsection introduces the models, which are built upon TF-IDF vectorization. The first model employs straightforward mathematical operations based on Linear Discriminant Analysis on the vectorized text, while the second model combines TF-IDF-based text vectorization with machine learning classifiers, namely Random Forest and XGBoost.

For TF-IDF Vectorization we use `TfidfVectorizer` from Scikit-Learn package [43]. Firstly we fit and transform this vectorizer on training part of our dataset, train the ML models and then use this already pretrained model on validation dataset to quantify its performance and choose the best model.

### 6.5.1. TF-IDF Feature Extraction with Linear Discriminant Analysis (LDA)

The very first model that we want to evaluate is very straightforward. This method is mentioned in the book *Natural Language Processing in Action* [23] as very simple yet effective. They say that *it has much better accuracy than the fancier state-of-the-art algorithms published in the latest papers*. For that reason, we wanted to verify how such a simple algorithm performs.

The training steps for this model are described below:

- Compute the average position (centroid) of all the TF-IDF vectors within clickbait and within non-clickbait titles.
- Compute the vector difference between the centroids.

To make the prediction we only need to find out if a new TF-IDF vector is closer to the clickbait or non-clickbait centroid.

The metrics for this model for both raw and cleaned datasets are presented in Table 6.3 below.

This straightforward algorithm based on TF-IDF with some basic math operations achieved relatively high-performance metrics. On the raw dataset it produced an F1-score of 0.755, while

Table 6.3: Performance metrics for TF-IDF with LDA

Dataset	Accuracy	Precision	Recall	F1-score
Raw	0.786	0.884	0.659	0.755
Cleaned	0.782	0.747	0.852	0.797

for the cleaned version it improved to 0.797. It shows that the preprocessing steps improved model performance in this case. To visualize the results, we present a confusion matrix for these models.

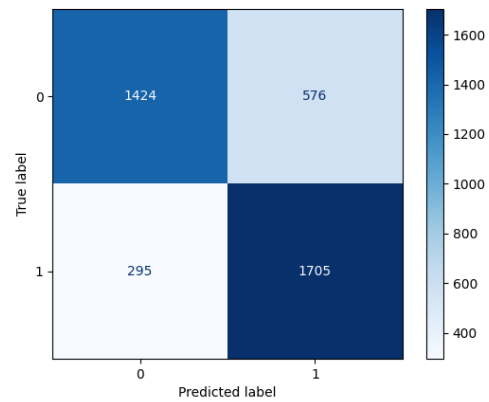
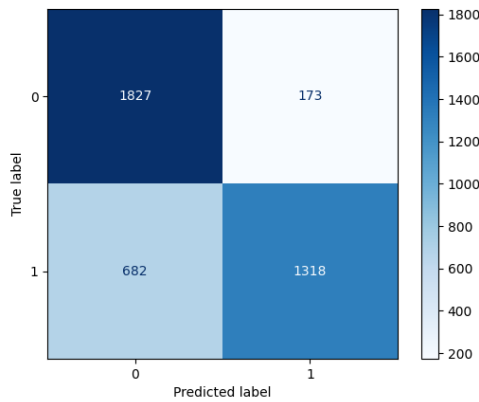


Figure 6.1: Confusion matrix for raw dataset    Figure 6.2: Confusion matrix for cleaned dataset

After cleaning the dataset, the model's ability to predict true positives improved significantly, and the number of false negatives was reduced.

### 6.5.2. TF-IDF Feature Extraction with Random Forest and XGBoost Classifiers

After vectorizing the text through `TfidfVectorizer` we apply traditional machine learning classifiers, namely Random Forest and XGBoost.

The model performance and corresponding metrics are evaluated using the validation dataset, and the results for RF and XGBoost are presented in Table 6.4 below.

Surprisingly, text preprocessing not only failed to improve the model's performance but actually caused a decline. For Random Forest, the F1-score dropped from 0.829 to 0.799, while for XGBoost, it dropped from 0.803 to 0.751. This might be due to the stopword removal step, as the average number of stopwords significantly differs across clickbait and non-clickbait titles (see Section EDA). Thus, removing them could lead to the abandonment of such crucial information. To support this claim, in one article mentioned previously [40], they also noticed that for the

Table 6.4: Performance metrics for raw and cleaned dataset

Dataset	Model	Accuracy	Precision	Recall	F1-score	ROC-AUC
Raw	Random Forest	0.832	0.842	0.816	0.829	0.906
Cleaned	Random Forest	0.804	0.818	0.781	0.799	0.882
Raw	XGBoost	0.816	0.862	0.752	0.803	0.894
Cleaned	XGBoost	0.778	0.849	0.673	0.751	0.858

Sentiment Analysis problem, stopwords removal worsened the model accuracy. This outcome emphasizes that preprocessing methods, while generally beneficial, sometimes may negatively impact performance. Another observation is that the RF model generally delivered slightly better results than XGBoost in this case. One can also notice remarkable improvement in metrics compared to the previous method of TF-IDF with LDA.

We also present the confusion matrix for the Random Forest model trained on Raw dataset that achieved F1-score of 0.829.

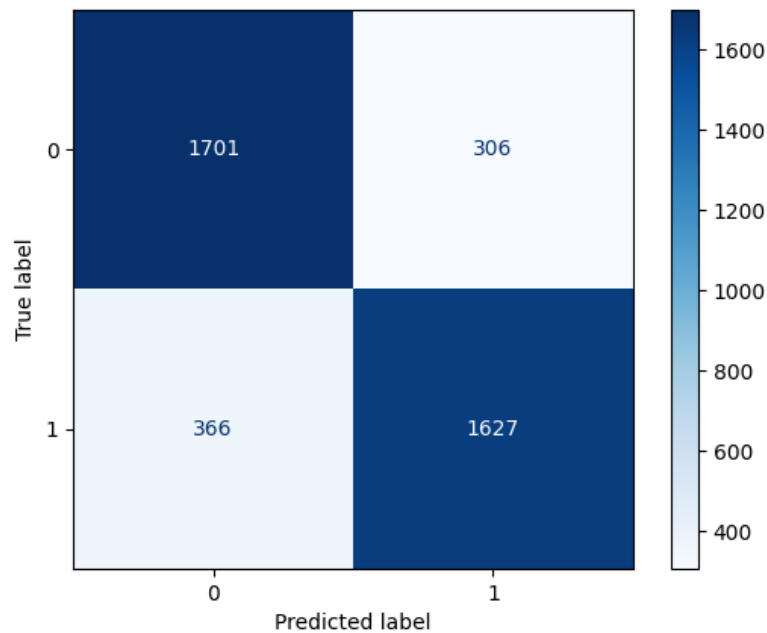


Figure 6.3: Confusion matrix for Random Forest model trained on raw dataset

The models we analyzed so far were based on a simple method of TF-IDF Vectorization. Unfortunately, it does not consider the order of words and the impact of the neighbors of a word on its meaning. The following word-embedding models that we analyze take into account the

neighborhood of a word and can identify synonyms, antonyms, or words belonging to the same category.

## 6.6. Word embedding models

Word embedding models are designed to generate vector representations of words that encapsulate their semantic meanings. In this thesis, we use two widely recognized models: GloVe and Word2Vec.

### 6.6.1. Word2Vec

We want to test only CBOW architecture in our thesis but with two different approaches. The first idea is to train our own Word2Vec model by using the Gensim library [44], while the second one is to use an already pre-trained model called *word2vec-google-news*. It is a model trained on the Google News dataset, which contains around 100 billion words [32], in which each word is represented as a 300-dimensional vector.

To calculate the embedding of a title, we take the average of the embeddings of all words in the title with embeddings available. If a word does not have an embedding in the Word2Vec model, it is omitted. Formally, for a title  $T = [w_1, w_2, \dots, w_n]$ , the embedding  $\mathbf{e}_T$  is calculated as:

$$\mathbf{e}_T = \frac{1}{|W|} \sum_{w \in W} \mathbf{e}_w,$$

where  $W = [w \in T \mid \mathbf{e}_w \text{ exists}]$  is the sequence of words in the title for which embeddings are available, and  $\mathbf{e}_w$  represents the embedding of word  $w$ .

After calculating the embeddings for a given title we fit classifiers on train and use validation dataset to quantify the performance. In Table 6.5, we present the resulting metrics of these models.

Table 6.5: Performance metrics for Random Forest and XGBoost models using Word2Vec embeddings

Model	Classifier	Accuracy	Precision	Recall	F1-score	ROC-AUC
Trained on our data	XGBoost	0.83	0.858	0.789	0.822	0.901
word2vec-google-news	XGBoost	0.832	0.84	0.818	0.829	0.908
Trained on our data	RF	0.829	0.865	0.779	0.82	0.9
word2vec-google-news	RF	0.832	0.868	0.783	0.823	0.903

It can be observed that the performance of all these models is similar. Also, training Word2vec on our own dataset does not provide any significant improvement. This is expected, as the pre-trained Word2Vec model was trained on the Google News dataset, which covers a wide range of general topics that match well with the general topics in our dataset of titles.

### 6.6.2. GloVe - Global Vectors for Word Representation

GloVe is the next word-embedding model that we test. This model is more efficient and maintains similar accuracy as Word2vec [23]. Training of Glove is performed on aggregated global word-word co-occurrence statistics from a corpus. It uses matrix factorization, while Word2Vec is based on backpropagation, which is usually less efficient. To build embeddings for the titles, we follow the same procedure as for the Word2Vec model (see Section 6.6.1). There are several pre-trained GloVe models available, each providing word embeddings with a different number of dimensions. We use a model returning 100-dimensional vectors from Kaggle [45]. The performance metrics for leveraging GloVe method are presented in Table 6.7.

Table 6.6: Performance metrics for Random Forest and XGBoost models using GloVe embeddings

Classifier	Accuracy	Precision	Recall	F1-score	ROC-AUC
Random Forest	0.804	0.817	0.782	0.799	0.865
XGBoost	0.832	0.84	0.818	0.829	0.908

In this case the XGBoost performed better than Random Forest. Also, the performance is very similar to the models trained using Word2Vec embeddings. Even though this approach did not significantly improve, we used only 100-dimensional vectors, while for Word2Vec we used 300-dimensional vectors.

### 6.6.3. TF-IDF weighted Word2Vec

The Word2Vec model primarily captures the meaning of words but does not account for their importance in text. In contrast, TF-IDF is an algorithm that evaluates word importance but lacks the ability to capture word meaning. The TF-IDF weighted Word2Vec model combines these two approaches, overcoming their limitations by leveraging both word meaning and importance.

This approach involves first generating vector representations using the Word2Vec model and then applying TF-IDF weights to the words. Although, it is not widely used, this method has been referenced in a few articles [33], [34]. We chose to evaluate whether it truly enhances model performance. The formula for calculating vector representations for a given title is given below:

For a title  $T = [w_1, w_2, \dots, w_n]$ , the embedding  $\mathbf{e}_T$  is calculated as:

$$\mathbf{e}_T = \frac{\sum_{w \in W} e_w(tfidf)_w}{\sum_{w \in W} (tfidf)_w},$$

where:

- $tfidf_w$  is the TF-IDF value for a given word,
- $W = \{w \in T \mid \mathbf{e}_w \text{ exists} \wedge tfidf_w \text{ exists}\}$  is the sequence of words in the title for which embeddings and TF-IDF values are available, and  $\mathbf{e}_w$  represents the embedding of word  $w$ .

The performance metrics for this model is presented below in Table 6.7.

Table 6.7: Performance metrics for Random Forest and XGBoost models for TF-IDF weighted Word2Vec

Classifier	Accuracy	Precision	Recall	F1-score	ROC-AUC
Random Forest	0.772	0.75	0.815	0.781	0.803
XGBoost	0.774	0.739	0.845	0.788	0.805

Unfortunately, this approach declined performance metrics compared to using single Word2Vec model.

To summarize the performance of word-embedding models, the best results were achieved using Word2Vec pre-trained on Google News with XGBoost and GloVe with XGBoost, both reaching an F1-score of 0.829.

## 6.7. Large language models

In this subsection, we test two popular Large Language models, namely those provided by OpenAI (ChatGPT) and BERT. For building the models we use

`dataset_title_detection_balanced_raw`, as there is no need to use preprocessed text.

### 6.7.1. OpenAI

Several endpoints and models are available through the OpenAI Platform. We use two endpoints for our clickbait detection problem, namely Text Generation and Embeddings. Below, we provide an overview of how these endpoints are applied.

**Embeddings.** To generate embeddings, the user needs to define the number of dimensions and the model. By default, the length of the embedding vector is 3072 for *text-embedding-3-large* model. We used *text-embedding-3-large* and tried different number of dimensions, namely 3072, 1000, 100 and 30 for generating embeddings.

When it comes to calling the API, we were facing some challenges. At the very beginning we tried synchronous API calls by sending requests row by row, which then turned out to be inefficient and prone to failure, since our train dataset contains 32000 rows and validation 4000 rows. Also, it very often happened that we got rate limit exception by using such a method. Then, we experimented with asynchronous API calls by using *asyncio* library in Python. Even though this method was much faster, we still got rate limit exceptions. Surprisingly, we discovered that the official OpenAI API provides batch API to process calls asynchronously. It is ideal for our usage since the costs are reduced by 50% and each batch completes within 24 hours.

After getting the embeddings for each title through this batch API we trained Random Forest and XGBoost. The performance metrics are presented below in the Table 6.8.

Table 6.8: Performance metrics for Random Forest and XGBoost models using OpenAI embeddings

Classifier	Dimensions	Accuracy	Precision	Recall	F1-score	ROC-AUC
Random Forest	3072	0.86	0.881	0.831	0.855	0.932
XGBoost	3072	0.866	0.877	0.851	0.864	0.938
Random Forest	1000	0.858	0.876	0.832	0.853	0.931
XGBoost	1000	0.86	0.866	0.849	0.858	0.934
Random Forest	100	0.822	0.827	0.813	0.82	0.901
XGBoost	100	0.819	0.82	0.816	0.818	0.901
Random Forest	30	0.788	0.788	0.787	0.787	0.862
XGBoost	30	0.783	0.783	0.781	0.782	0.857

It can be noticed that these models outperformed all of the approaches before. The best performing model using this approach achieved F1-score of 0.864, while for previous methods (TF-IDF, Word2Vec, GloVe) the highest F1-score was 0.829. Furthermore, the performance remained similar with dimensionalities of 3072 and 1000, while there was a noticeable decline in metrics when the dimensionality of embedding vectors was reduced to 100 or even 30. It is presented on the graph 6.4 below.

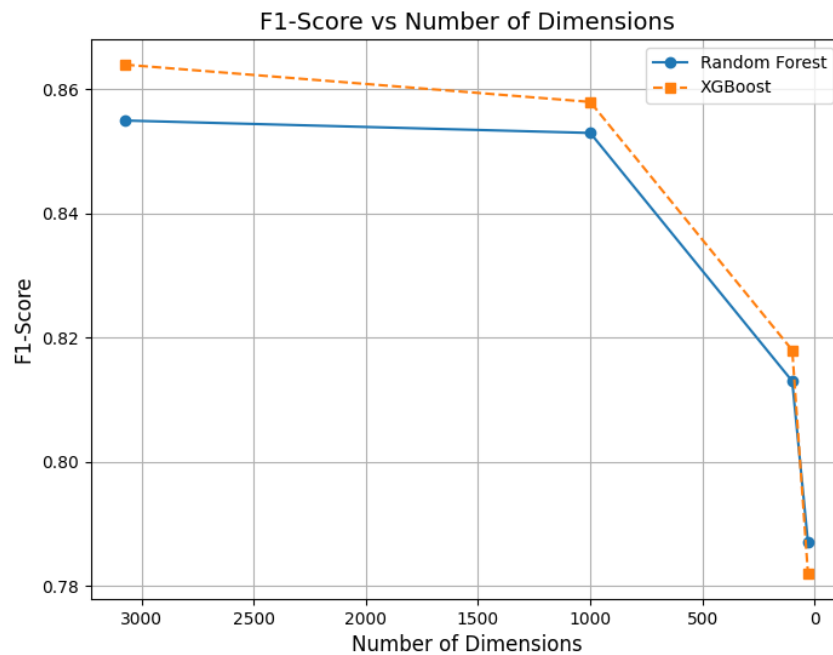


Figure 6.4: Effect of number of dimensions of embeddings on F1-score



**Text Generation.** OpenAI platform also offers simple LLM APIs to generate text based on a given input text from a prompt.

The API provides several models for text generation. We tested *GPT-3.5-turbo* and *GPT-4o-mini*. According to the documentation *GPT-4o-mini* has higher intelligence than *GPT-3.5-turbo* but is just as fast. It is worth mentioning that we also tried to test *GPT-4* model, but unfortunately, there is a batch limit of 100000 tokens for this model for our tier [46], which is too low quota for our purposes. Also, there are multiple hyperparameters that can vary (e.g. temperature, frequency\_penalty, presence\_penalty, etc.). We can't test every variation. Instead, we'll focus on prompt engineering and aligning it with the best practices.

The prompt plays a vital role in determining the quality of the output. In the documentation, they suggest several best practices methods for prompt engineering to improve the quality of an output (e.g. specifying the steps required to complete a task, including details in your query to get more relevant answers, providing examples) [47]. The first system prompt we test does not contain any examples, whereas the second one includes examples.

#### System Prompt 1

You are an assistant tasked with classifying article titles as either clickbait or legitimate. Your response should be precise and straightforward: respond with only "1" if the title appears to be clickbait, and only "0" if it seems legitimate.

Do not include any additional text, explanations, or reasoning in your response.

In the second system prompt we provide examples, as the documentation suggests. The prompt is presented below:

#### System Prompt 2

You are an assistant tasked with classifying article titles as either clickbait or legitimate. Your response should be precise and straightforward: respond with only "1" if the news title appears to be clickbait, and only "0" if it seems legitimate. Do not include any additional text, explanations, or reasoning in your response.

Here are some examples:

Title: "23 Pictures That Will Make You Say "Huh."" Response: 1

Title: "The Most Moving Personal Essays You Needed To Read In 2015" Response: 1

Title: "10 Life Hacks That Will Change the Way You Live Forever" Response: 1

Title: "Government Announces New Infrastructure Plan for 2024" Response: 0

Title: "The Shocking Truth Behind Your Morning Coffee!" Response: 1

Title: "Local School Wins National Award for Academic Excellence" Response: 0

For both of the prompts presented above we use following user prompt:

#### User Prompt

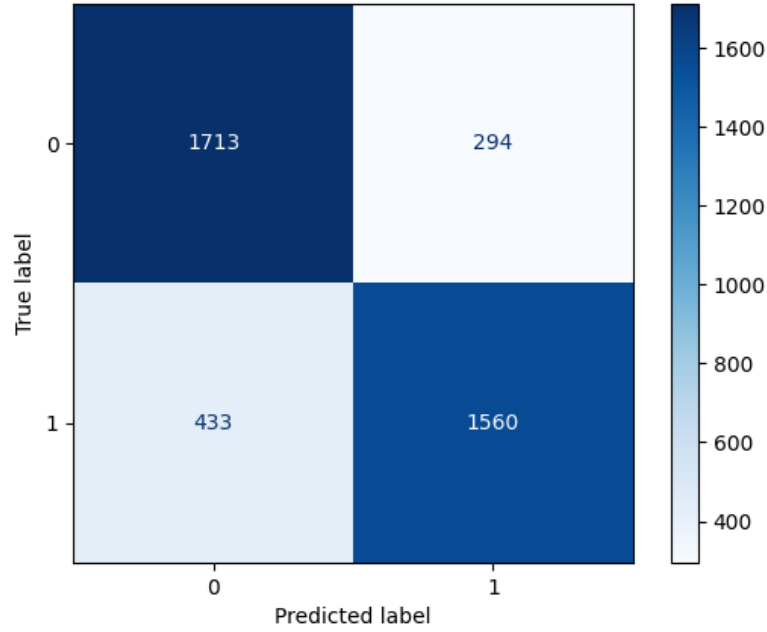
Here is the title that you need to classify: [title]

In addition to experimenting with prompts, we also adjusted the temperature parameter. Higher temperature values are expected to make the output more random, while lower values are expected to result in more deterministic responses. The range of possible values for temperature is  $[0, 2]$ . The performance metrics for these prompts and different models and temperature are presented in Table 6.9

Table 6.9: Performance metrics for direct prompting with OpenAI

Prompt	Model	Temperature	Accuracy	Precision	Recall	F1-score
Prompt 1	GPT-4o-mini	1	0.772	0.849	0.659	0.742
Prompt 2	GPT-4o-mini	1	0.819	0.841	0.786	0.813
Prompt 2	GPT-4o-mini	0	0.818	0.841	0.783	0.811
Prompt 2	GPT-4o-mini	2	0.798*	0.841*	0.735*	0.784*
Prompt 2	GPT-3.5-turbo	1	0.742	0.679	0.914	0.779

We observed that setting the temperature to its maximum value (2) caused the model to generate strange outputs, such as random combinations of characters, which were impossible to parse, and we labeled such outputs as non-clickbait. This case only happened when we set temperature to 2 (it is presented in the table above with '\*'), otherwise we did not get any parsing error. Setting the temperature to the minimum value (0) did not improve value of F1-score. On the other hand, there was a difference in metrics between *GPT-3.5-turbo* and *GPT-4o-mini* in terms of F1-score, which confirms that *GPT-4o-mini* has higher intelligence. On the diagram below we present a confusion matrix for the model that leveraged *GPT-4o-mini* model with second prompt and temperature parameter set to 0.

Figure 6.5: Confusion matrix for *GPT-4o-mini* model with second prompt

Additionally, comparing these models to those built using the Embedding endpoint shows a decline in F1-score from 0.864 to 0.813.

### 6.7.2. RoBERTa

One also tested RoBERTa, which is a LLM model introduced in the paper *RoBERTa: A Robustly Optimized BERT Pretraining Approach* by Yinhan Liu and colleagues [3]. It is an improved version of Google’s BERT model, which was first released in 2018. We tested already fine-tuned version of RoBERTa on clickbait detection dataset taken from *huggingface* [48].

The performance metrics are presented below in Table 6.10.

Table 6.10: Performance metrics for fine-tuned version of RoBERTa model

Classifier	Accuracy	Precision	Recall	F1-score	ROC-AUC
RoBERTa	0.832	0.795	0.894	0.842	0.91

## 6.8. OpenAI Embeddings combined with informativeness measures

We have tested several models and approaches to build a clickbait detection classifier. The best-performing model so far achieved an F1-score of 0.864. It used embeddings from OpenAI combined with an XGBoost model. We also created a model using informativeness measures, including our custom baitness measure, which delivered impressive results.

Our goal for the final clickbait detector is to combine these two approaches - using embeddings (3072 and 1000 dimensional) alongside with 15 informativeness measures. These measures are presented in the table 6.11.

Table 6.11: Measures used for clickbait detection

No.	Measure
1	Common Words Ratio
2	Capital Letters Ratio
3	Capital Words Count
4	Bait Punctuation Count
5	Non-Bait Punctuation Count
6	Numbers Count
7	2nd Person Pronouns
8	Superlatives Ratio
9	Speculatives Usage
10	Bait Phrases Usage
11	Polarity Score
12	Subjectivity Score
13	FRES (Flesch Reading Ease Score)
14	FKGL (Grade Level)
15	Baitness Measures (custom metric derived from all of the above)

Refer to Section 5 for a detailed explanation of these metrics.

The performance metrics for these models are summarized below in Table 6.12:

Table 6.12: Performance metrics for OpenAI embeddings combined with Informativeness measures

Dmensions	Model	Accuracy	Precision	Recall	F1-score	ROC-AUC
1000	Random Forest	0.898	0.916	0.876	0.895	0.961
1000	XGBoost	0.91	0.922	0.896	0.909	0.969
3072	Random Forest	0.893	0.914	0.867	0.89	0.963
3072	XGBoost	0.908	0.92	0.893	0.906	0.969

It is visible that all of the models that were trained on a dataset consisting of both embeddings from OpenAI and informativeness measures outperformed all of the previous ones. The best achieved a F1-score of 0.909, which is a satisfactory result. It also shows that by adding our custom measures, we enhanced the performance of the model built with embeddings alone. We also present the confusion matrix for the best performing model.

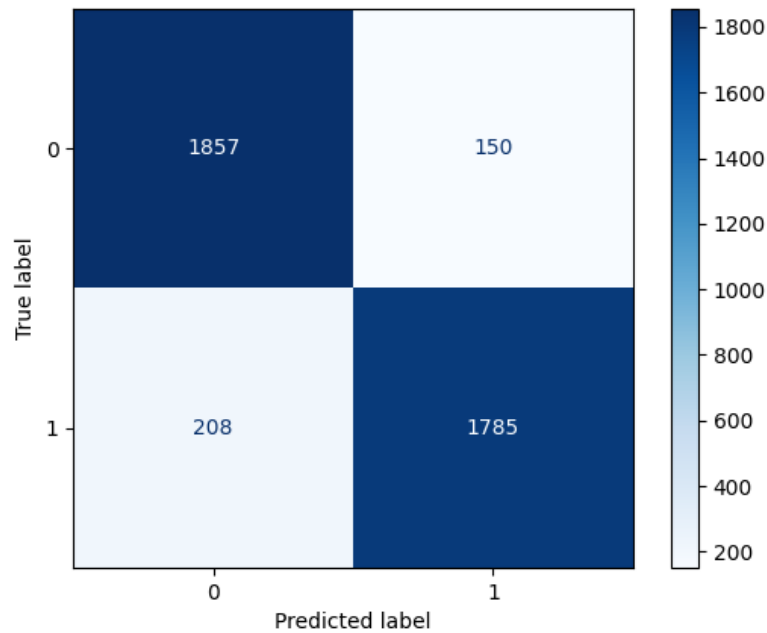


Figure 6.6: Confusion matrix for the XGBoost model using 1000-dimensional embeddings combined with informativeness measures

This model outperformed all others, delivering the best performance.

### 6.9. Final clickbait model selection

The model that combines both informativeness measures and 1000-dimensional OpenAI embeddings, using XGBoost, achieved the highest F1-score (0.909) and will serve as the primary model for clickbait detection in our browser extension.

## 7. Article spoiling

One of the features our team is implementing in the project is a clickbait spoiling solution. The goal of the article spoiling is to inform the user about the content of the article. It is done by generating the short text for the user to satisfy the curiosity caused by the title of the article. Spoiler tries to answer the question or address the topic mentioned there. Other times, it is the reference to the title topic from the article body. It serves as the first line of defense to prevent users from falling for clickbait. After reading a displayed word or sentence from the article, the user might decide not to enter the potentially clickbait source.

### 7.1. Related work

The paper [49] states that the best results for the spoiling problem among the participants of the SemEval 2023 Challenge were obtained by using variations of the BERT model. However, BERT-based models have the limitation of processing input text with a maximum length of 512 tokens. This posed a challenge, as the SemEval Spoiling Dataset mentioned in the paper contains examples which exceed the allowed text length. Consequently, contest participants utilized the *Best Matching 25* (BM25) algorithm 7.1. Another approach involved chunking the input text. The output spoiler of each chunk was merged to produce the final spoiler.

#### Equation: Best Match 25

$$\text{Score}(D, Q) = \sum_{q_i \in Q} \text{IDF}(q_i) \cdot \frac{\text{TF}(q_i, D) \cdot (k_1 + 1)}{\text{TF}(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)} \quad (7.1)$$

- $\text{Score}(D, Q)$ : Relevance score of document  $D$  for query  $Q$ .
- $\text{IDF}(q_i)$ : Inverse document frequency of query term  $q_i$ .
- $\text{TF}(q_i, D)$ : Term frequency of  $q_i$  in document  $D$ .
- $|D|$ : Length of document  $D$ .
- $\text{avgdl}$ : Average document length.

## 7.1. RELATED WORK

- $k_1$ : Saturation parameter.
- $b$ : Length normalization parameter.

Another model mentioned in SemEval 2023 Challenge solutions was *T5-large*. It is a text-to-text transformer which has 770 million of parameters. T5-large was mostly trained using texts of 512 input tokens long. However, thanks to the use of relative attention it can use much longer input sequences. This solves the problem of limited input for spoiling task.

The correctness between generated spoilers and human-generated spoilers was measured by using several metrics: BLEU, BERT-Score (BSc), METEOR (MET), and cosine similarity (see Table 7.1).

Table 7.1: Clickbait spoiling metrics with descriptions

Metric	Description
Cosine Similarity	Measures similarity between two texts by calculating the cosine of the angle between their vectorized representations. Useful in tasks where semantic similarity is key, such as sentence matching.
BLEU (Bilingual Evaluation Understudy)	Measures n-gram precision by comparing the generated text with a reference text. It is useful for evaluating fluency and coherence in translation tasks.
BERT-Score (BSc)	Uses contextual embeddings from models like BERT to assess semantic similarity between the candidate and reference texts. It calculates cosine similarity between token embeddings, capturing meaning.
METEOR (MET)	Combines precision and recall. Allows partial matches and synonyms to improve correlation with human judgments. METEOR includes stemming and word order penalties to assess fluency and relevance more flexibly.



## 7.2. Our approach

Our team approached the article spoiling task by testing various transformer-based large language models (RoBERTa, T5-large, GPT-4o-mini). Then evaluation metrics (see Table 7.1) were calculated to measure the similarity between generated text and ground truth text. The best spoiling model was used in the browser extension to spoil suspected clickbait articles.

## 7.3. Data preprocessing

The preprocessing held for the spoiling problem was similar to the one described in Section 6.3. While large language models were used for spoiling, it was important not to damage the structure or syntax of the text. That is why a brief text cleaning and manipulation process was conducted. Data cleaning covered:

1. **Removing special characters:** There were special characters in the text left after the extraction of the articles. It might have lowered the similarity of the generated results to the human-created spoilers.
2. **Changing contractions to full form:** Contractions in the text (e.g., "can't" to "cannot") were expanded to their full form to ensure better text consistency and facilitate model understanding.

## 7.4. Large language models

### 7.4.1. OpenAI

**Technical details** In this section, the usage of the *GPT-4o-mini* model from OpenAI is described. We chose this model for text generation since the pricing was pretty low: 0.15\$/1M input tokens and 0.6\$/1M output tokens [50]. The token definition by OpenAI is 0.75 of word, so for English texts it is approximately 4 characters. Furthermore, our team grouped the calls into batches, so it gave us a discount 50% while generating the texts. The batch limit was the size of 2M tokens. Therefore, the spoiling dataset was divided into 3 batches of approximately 1130 prompts each batch. The operation of spoilers generations cost about 0.75\$. The text generated by the model had a *JSON* format for easier text extraction. The model temperature was set to 1 based on the experience described in Section Text Generation 6.7.1.

**System Prompt 3**

You are a model designed to **generate** concise **spoilers** from articles as "humanSpoiler" and **extract article body paragraphs** that provide information for the spoiler as "spoiler". The "spoiler" paragraphs should form a consistent text. Your task is to analyze the main question or topic posed by the article's title ("targetTitle") and generate a spoiler based on the content provided in the article's paragraphs ("targetParagraphs"). Your response must include a **JSON** object with the keys "humanSpoiler" and "spoiler".

Guidelines:

1. Use "targetTitle" to determine the central question or topic the spoiler should address. 2. Review "targetParagraphs" to find the most relevant details that directly answer the question or topic posed in the title. 3. Consider the provided tag to adjust your response:

- "phrase": Provide a short, direct answer or single phrase.
- "passage": Provide a more detailed, informative spoiler.
- "multi": Provide multiple sentences or a structured, longer spoiler.

Answer:

```
{
  "humanSpoiler": "<A model-generated spoiler>",
  "spoiler": "<A concise, extracted spoiler from the article's content>"
}
```

Examples:

Example 1:

Input:

```
{
  "idx": 1,
  "tag": "phrase",
  "targetTitle": "Teen Mom 2 Star Jenelle Evans Reveals Sex Of Her Second Child",
  "targetParagraphs": "Teen Mom 2 star Jenelle Evans took to Twitter and Instagram Monday (February 3) to announce that she and boyfriend Nathan Griffin..."
}
```

Output:

```
{
  "idx": 1,
  "humanSpoiler": "Boy",
  "spoiler": "boy"
}
...
```

Pay attention to return valid **JSON** format!

There are two main columns in the spoiling dataset: humanSpoiler, spoiler (Table 3.1). Based on the article's title, the tag of the spoiler, and the article paragraphs, two columns were generated: humanSpoiler\_ext, spoiler\_ext. humanSpoiler\_ext is a generated spoiler based on the content provided in the article (targetParagraphs). This can be the answer to the question stated in the title (targetTitle) or just a short description of the topic announced by the title. spoiler\_ext is a content extracted from the article's paragraphs, which allowed for text generation as humanSpoiler\_ext.

During the text extraction process, 45 responses from the OpenAI model turned out to be difficult to convert for the needed form. Our team decided to remove those data points. This resulted in the final dataset of 3310 generated spoilers and extracted paragraphs.

**Spoiling examples** In Table 7.2 spoiling examples were shown for each tag. There are model generated and human written spoilers. The table contains also model and human extracted paragraphs. Model texts are more detailed and visibly longer. Model generated spoilers do not contain hashtags (eg. #StopClickbait, @thesheertruth).

Table 7.2: Article and Spoiler Data with Extracted Paragraphs

Field	Data
<b>Article Title</b>	Sasheer Zamata Joins SNL Cast Amid Controversy Over Black Women
<b>Tag</b>	phrase
<b>Generated Spoiler</b>	Sasheer Zamata joins SNL cast.
<b>Human Spoiler</b>	Sasheer Zamata (@thesheertruth)
<b>Human Paragraph</b>	Sasheer Zamata

*Continued on next page*

Field	Data
<b>Model Paragraph</b>	Sasheer Zamata is joining the SNL cast after criticism over the lack of black women on the show.
<b>Article Title</b>	Halal supermarket in Paris told to sell pork and alcohol or face closure
<b>Tag</b>	passage
<b>Generated Spoiler</b>	Local authorities are requiring a halal supermarket in Paris to sell pork and alcohol to avoid closure due to lease violations.
<b>Human Spoiler</b>	Its because the store did not go through the proper permit process
<b>Human Paragraph</b>	It argues that members of the local community are not being served properly if the shop does not sell pork or alcohol products.
<b>Model Paragraph</b>	A halal supermarket in Colombes, Paris, has been ordered by local authorities to sell pork and alcohol or face closure because it does not meet the conditions of its lease as a general food store. The mayor argues that the shop's limited offerings do not serve the local community's needs, prompting legal action to revoke the lease.
<b>Article Title</b>	The 10 Professions With the Most Psychopaths
<b>Tag</b>	multi
<b>Generated Spoiler</b>	CEOs, lawyers, and media professionals top the list of professions with the most psychopaths.
<b>Human Spoiler</b>	CEO, Lawyer, Media (TV and radio), Salesperson, Surgeon, Journalist, Police officer, Clergy person, Chef, Civil servant
<b>Human Paragraph</b>	CEO, Lawyer, Media, Salesperson, Surgeon
<i>Continued on next page</i>	

Field	Data
<b>Model Paragraph</b>	According to research by psychologist Kevin Dutton, the professions that attract the highest number of psychopaths include CEO, lawyer, media (TV and radio), salesperson, surgeon, journalist, police officer, clergyperson, chef, and civil servant. The other end of the scale features jobs least likely to be populated by psychopaths, such as care aide, nurse, therapist, beautician, charity worker, teacher, creative artist, doctor, and accountant.

**GPT-4o-mini spoilers evaluation** The selected model had 2 tasks. First, *GPT-4o-mini* extracted a spoiler from the given original body text paragraphs. This extracted spoiler is referred to in the graphs as **paragraphs**. The second task was to generate a spoiler from scratch using the article title and its body text. Such a generated spoiler is referenced in the graphs as **spoilers**. Both outputs of the model were compared with ground truth texts. Thus, by the **spoilers** score, the similarity between the generated spoiler and the human-extracted spoiler is marked. By the **paragraphs** similarity score, we marked the similarity between the model-extracted spoiler from the article body and the ground truth extracted text for the spoiler. Both similarity scores were calculated in the range [0, 1] by the following metrics:

**Cosine Similarity** For cosine similarity measurement, the Scikit-Learn library [43] was used. It creates the vectors from text with TF-IDF method using `TfidfVectorizer`. As the corpus for the TF-IDF, all of the human generated spoilers and human extracted paragraphs were taken. Then the cosine of the angle between vectorized representations is calculated with formula:

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

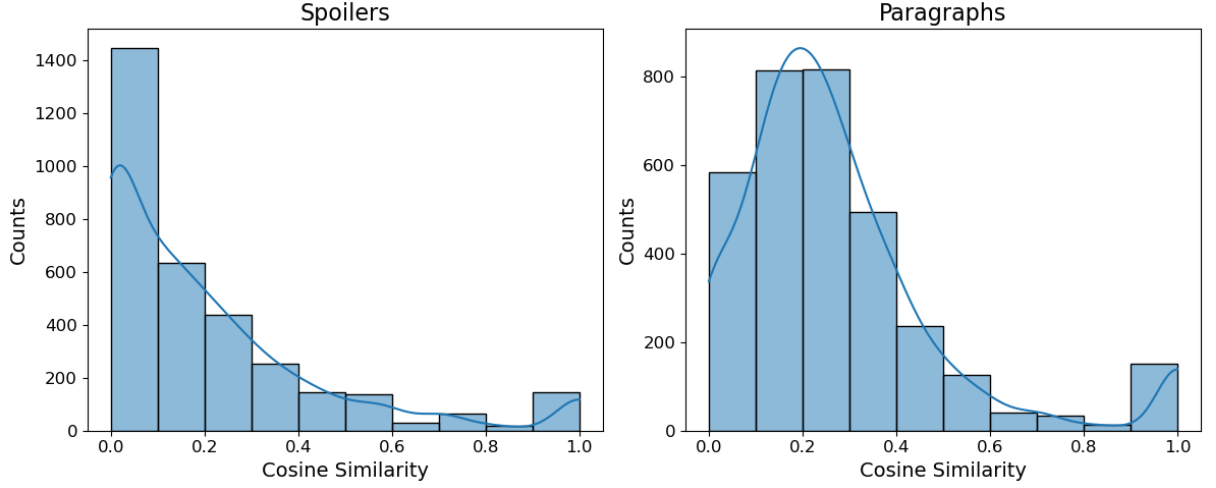


Figure 7.1: Cosine Similarity between generated and original texts

Figure 7.1 shows that the similarity between the extracted paragraphs and the original paragraphs in the dataset is greater than between spoilers. Most of the paragraphs have similarity between 20%-30% ( $\simeq 800$  samples) while the most common similarity among spoilers is 0%-10% ( $\simeq 1400$  samples). The measurement is relatively low, but this is expected since Cosine Similarity counts only similarity in the vocabulary used.

**BLEU** For BLEU measurement, the NLTK library [42] was used. The precision of matching n-grams calculates the metric.

$$\text{Precision}_n = \frac{\text{Number of matching n-grams}}{\text{Total n-grams in candidate}}$$

As a default, BLEU includes unigrams up to 4 grams. In our case the highest results were obtained for weights: [1.0, 0.0, 0.0, 0.0] (Table 7.3), starting with 1-gram and ending with 4-gram. However, the metric uses only 1-gram with such weights. In case of checking the whole range up to 4-grams, weights were set to [0.40, 0.30, 0.20, 0.10]. This choice is based on some reasonable manual tests we have taken.

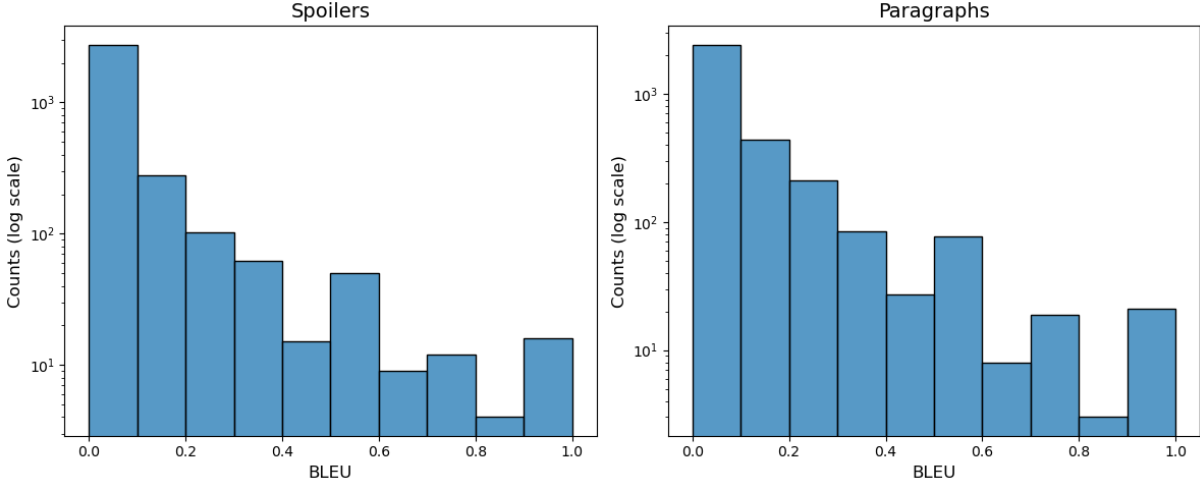


Figure 7.2: BLEU similarity between generated and original texts, weights [0.40, 0.30, 0.20, 0.10] (log scale)

Figure 7.2 shows that the most frequent score among Spoilers and Paragraphs in BLEU similarity falls between 0%–10% with  $\simeq 2800$  samples for spoilers and  $\simeq 2400$  for paragraphs. This is expected since the generated text might be completely different in terms of vocabulary used but still conveys the same meaning, which BLEU does not capture.

**METEOR** METEOR [51] calculation was performed with the NLTK library [42]. It is a complex metric which combines the characteristics of BLEU and Cosine Similarity.

$$\text{METEOR} = (1 - \alpha) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(1 - \alpha) \cdot \text{Recall} + \alpha \cdot \text{Precision}} + (\beta \cdot \text{Fragmentation})$$

- Precision: Ratio of matched unigrams to total unigrams in the generated text.
- Recall: Ratio of matched unigrams to total unigrams in the ground truth text.
- Fragmentation: A penalty term based on how fragmented the generated parts are within the ground truth parts.
- $\alpha$ : Balances Precision and Recall (default 0.9 in NLTK).
- $\beta$ : Controls the penalty for fragmentation (default 3.0 in NLTK).

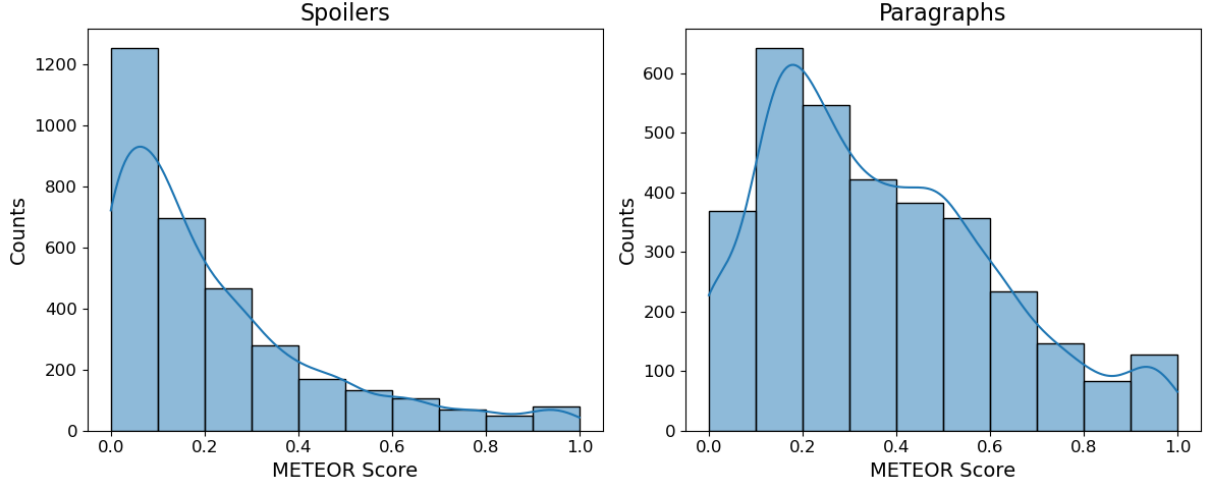


Figure 7.3: METEOR similarity between generated and original texts

Figure 7.3 shows that the similarity of the generated spoilers is positively skewed. More than  $\simeq 1200$  samples have similarity between 0%-10%, which is the largest group. The paragraphs distribution is closer to the normal distribution, although it is also positively skewed. The largest group has similarity between 10%-20% and represents  $\simeq 600$  samples.

**BERT-Score** The BERT-Score was calculated with the Hugging Face Transformers [52] and BERT-Score [53] libraries. This metric is the most complex so far used for the spoiling problem. It involves many steps before calculating the final value:

- Step 1: Generated and ground truth texts are represented with contextual embeddings created by pretrained model. In our case, the *BERT base uncased* model was used for this purpose. The model captured surrounding words while creating embeddings.
- Step 2: The cosine similarity of generated and ground truth texts is calculated.
- Step 3: Each token in the generated text is matched with the most similar token in the ground truth text and vice versa. The Recall, Precision are calculated which results with F1-score.

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Step 4: TF-IDF algorithm is used to give the importance weights for words. The weights are then incorporated into the BERT-Score formulas.
- Step 5: Final indicators are linearly rescaled to improve human readability.



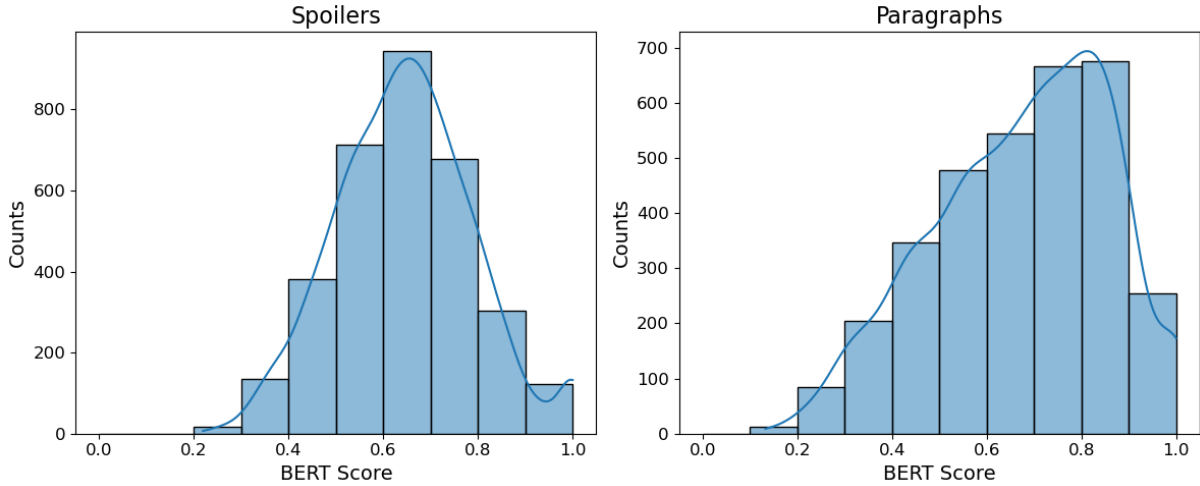


Figure 7.4: BERT-Score between generated and original texts

Figure 7.4 shows the normal distribution of BERT-Score for spoilers. It is centered with the range of 60%-70% similarity ( $\simeq 600$  points). The similarity distribution of paragraphs is negatively skewed. Its peak is the group of similarity between 80%-90% ( $\simeq 700$  samples).

Table 7.3: Mean values of performance metrics for generated spoilers

Metric	Spoiler	Paragraph
BLEU (1-4 grams)	0.068	0.094
BLEU (1 grams)	0.167	0.203
Cosine Similarity	0.207	0.274
METEOR	0.226	0.365
BERT-Score	0.644	0.674

**Spoiling Metrics summarization** The comparison of all mean values calculated by metrics is shown in Table 7.3. In every metric, the extracted paragraph similarity was higher than the spoiler generation. It comes from the fact that, in general, the text extraction problem gives more consistent results with ground truth texts than the text generation problem.

The complexity of the metrics, ranked from least to most complex, is as follows: Cosine Similarity, BLEU, METEOR, and BERT-Score. It turned out that the BLEU measurement shows the lowest similarity score. This result comes from the fact that the generated text is unlikely to use the same words as the ground truth text. Additionally, the generated texts and the model-extracted paragraphs were longer than the human texts. It lowers the score of all metrics but the most negative impact was put on BLEU metric, since compared n-grams are

unmaching. This also causes the higher 1-gram BLEU metric score as it does not take into account the order of words. The BERT-Score showed the highest score. This is because this metric takes into account not only the linguistic similarity but also the meaning of the generated text.

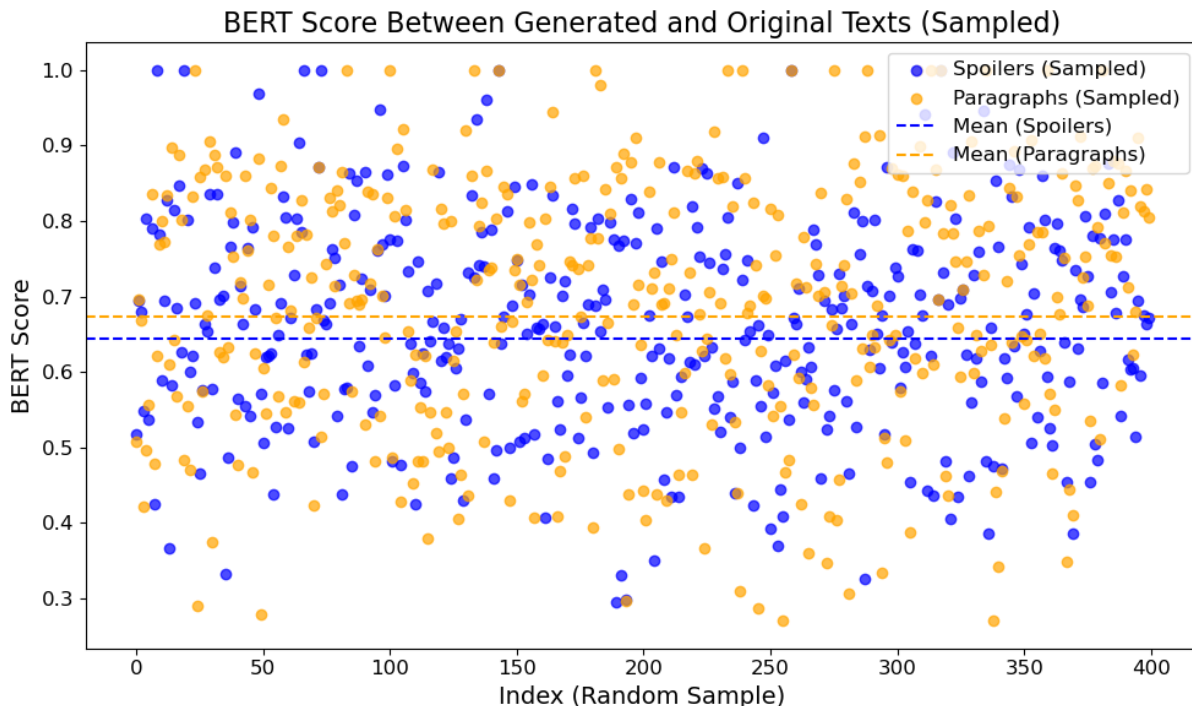


Figure 7.5: Data points of BERT-Score among Spoilers and Paragraphs

Scatter plot for BERT-Score shown as Figure 7.5 visualizes how data points are located in terms of similarity score. There were randomly sampled 400 data points for this plot (around 13% of all data). The cloud of points is compact with no deviations upward or downward. This indicates that, in terms of text meaning, the model is generalizing well. In comparison to Figure 7.6 (random sampled BLEU similarity), it is visible that the cloud of points representing BLEU spoilers has a lower variance and is located at the bottom of the similarity scale. This indicates that the n-grams in generated texts are mostly incompatible with ground truth text but the meaning of the text is still quite accurate (high BERT-Score).

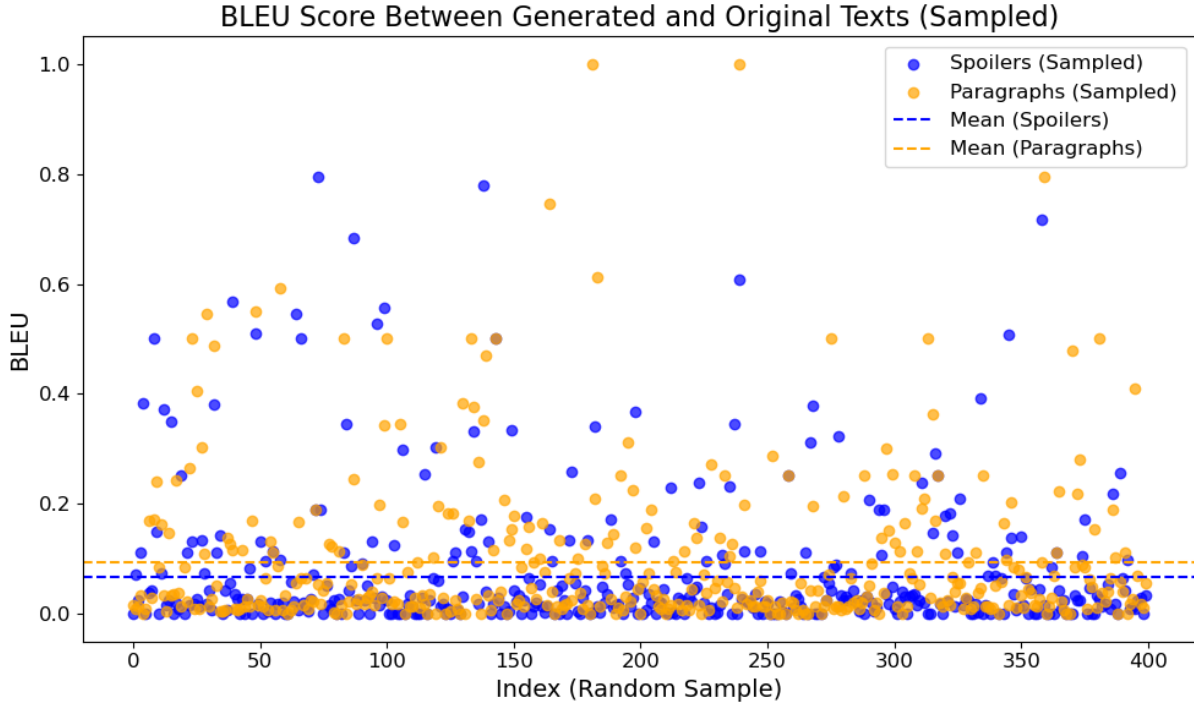


Figure 7.6: Data points of BLEU score among Spoilers and Paragraphs

#### 7.4.2. BERT based models

The RoBERTa model was mentioned in paper [49] as the most effective solution used in the Clickbait contest. Therefore our team decided to test RoBERTa in our setting.

**RoBERTa** The RoBERTa model was trained on the Wikipedia articles, CommonCrawl News, OpenWebText (160 GB of text data). The provided texts were of 512 length tokens maximum. The tokens were created by using the WordPiece algorithm - subword tokenization technique. The maximum length of 512 tokens poses a challenge in the extraction of clickbait spoilers. The articles of the *SemEval Spoiling Dataset* [12] are mostly longer than that. During tests, the article title was given to the model, the amount of tokens left were distributed for the article content. Our team decided to deal with long paragraphs by splitting them into sentences. Then *Best Match 25* (Equation 7.1) algorithm was used to grant the importance score for each sentence. About 15 sentences with highest score were then used to serve as the article content from which the spoiler was extracted. If the text given to the model reached the tokens limit, it was truncated.

**Spoilers Evaluation with RoBERTa** To evaluate the quality of the spoilers, the BERT-Score was calculated. For this purpose, the library [53] from HuggingFace was used. The pre-trained *bert-base-uncased* [54][55] model was employed. The resulting similarity score was 0.6

(with 1.0 being the maximum). However, despite the relatively high BERT-Score, the spoilers generated by RoBERTa were not human-readable, as shown in Table 7.4. The generated spoilers lacked context, and the sentences were completely out of mixed, making them nonsensical for users.

To obtain satisfactory, human-readable spoilers, the model needs to be fine-tuned. Since RoBERTa, like other models in the BERT family, is primarily a masked language model, it is not inherently designed for text generation tasks. Given the need for a text generation model for the browser extension, our team decided to experiment with *T5-large* or *GPT-4o-mini* models which are dedicated for text generation.

Table 7.4: Spoiler example generated by *RoBERTa* with *BestMatch25*

Field	Data
<b>Article Title</b>	CNN caught ADDING key word to Trump’s comments; Twitter ERUPTS - Alle.
<b>Human Spoiler</b>	They added the word racial to his comments about profiling.
<b>Generated Spoiler</b>	Suggestion for the network: Here is another one: perhaps it is time for CNN to update its slogan to The Most MIS-Trusted Name in News - now that it is surpassed even MSNBC in this regard. CNN has been running headlines throughout the day that specifically quotes Trump as saying racial profiling and the Fox News excerpts the network sent out Monday evening also framed his quotes as responses to questions about racial.

#### 7.4.3. T5-large

Research papers (see Section 7.1) have identified *T5-large* as a state-of-the-art solution for the clickbait article spoiling problem. Our team decided to test this model with an input length of 1024 tokens and an output length of 150 tokens. Initially, the pre-trained version of the model was used; however, since *T5* is a text-to-text transformer, it does not inherently address the spoiler extraction problem. Fine-tuning the model was necessary.

We attempted to fine-tune the model, but given our access to a machine with 32 GB of RAM and CPU-only processing, this was not feasible. Loading the training data and the entire model into memory caused the Python kernel to crash. To address this issue, we optimized the code by reducing the input text length to 800 tokens and the output length to 128 tokens. Additionally,

we lowered the batch size of the training data from 12 to 4 inputs. These changes improved the situation, preventing the Python kernel from crashing initially. However, the training process still consumed all 32 GB of memory, causing the machine to crash before completing the training.

Given the satisfactory results achieved with *GPT-4o-mini*, we decided to focus on developing a browser extension rather than fine-tuning our own model architectures without adequate computational resources.

#### 7.4.4. Final spoiling model selection

Our team encountered numerous issues according to the RoBERTa input text limit (Section 7.4.2) and computational resources for T5-large fine-tuning (Section 7.4.3). This prompted us to select *GPT-4o-mini* as the model responsible for the generation of spoilers in the browser extension. The metrics described in Section 7.4.1 indicate good results for this model.

## 8. Browser extension

In our thesis, we focus on developing methods and models for clickbait detection. To demonstrate our accomplishments, we created a browser extension.

### 8.1. Architecture

#### 8.1.1. Overview

Our app consists of two main modules: the extension itself, which works locally in the browser, and a separate Python microservice hosted on GCP.

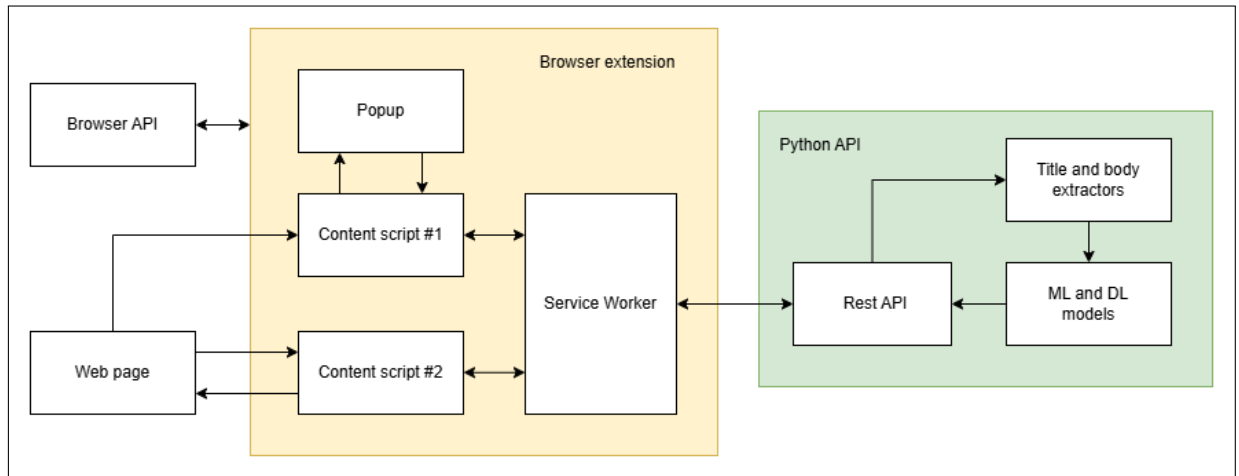


Figure 8.1: System architecture

#### 8.1.2. Extension architecture

Many components make up the extension. All components are JavaScript scripts and communicate with each other via messaging. Each of them can access the browser API, however content scripts only in a limited scope. Browser API provides interaction with the browser, i.e. with tabs, icons, bookmarks, or storage.

- **Popup**

Apart from the JS script, the popup also consists of an HTML file. It is the main UI element

in the browser extension. It's a small window that appears after clicking the extension icon. It does not have direct access to the web content. Instead, it can call content scripts or inject other scripts into web pages.

- **Content scripts**

This type of script interacts directly with the web content. Content scripts can be programmatically injected or configured to run automatically on specified URLs. They can access browser storage, but not icons or bookmarks. In order to do it, they need to message the service worker.

- **Service worker**

This is a script that runs in the background. It's active only when it is receiving events. Does not have access to the web content, but has full browser API access.

- **Options page**

There is also an options page, similar to the popup. It can be used to create a settings page for users.

Usage of the components in the context of our system is presented in section 8.2.

### 8.1.3. Python API architecture

For prediction processing and serving models, we use external Python API. It utilizes REST communication provided by the Flask [56] package and also consists of several components.

- **Title extractors**

Titles are extracted from HTML code using the BeautifulSoup4 [57] package.

- **Body extractor**

To extract the article body, we use the Trafilatura [58] package, which has pre-trained models capable of extracting the main content from the HTML code.

- **Detection models**

Our API uses a previously trained clickbait detection model. It utilizes OpenAI embeddings and a selection of informativeness measures, which are passed to the XGBoost model.

- **Explanations module**

Explanations for clickbait detection are provided by calculating informativeness measures. While it is not strictly connected to the clickbait detection model, it presents the characteristics of the title that may impact the prediction.

- **Spoiling module**

This module is responsible for generating spoilers based on the article body and title. We use the GPT-4o-mini model for short spoiler generation.

It serves two endpoints which are used by the extension:

- **POST/extract\_and\_predict**

Accepts a payload with URL and HTML content, runs the full pipeline including clickbait detection, explanation creation, and spoiler generation. Returns a dictionary containing all model outputs. Used in post-click detection mode.

### Request Payload

```
{
  "url": "string",
  "html": "string"
}
```

### Response

```
{
  "prediction": int,
  "probability": float,
  "explanation": "string",
  "spoiler": "string"
}
```

- **POST/predetect**

Accepts a payload containing URL and HTML content. Returns a dictionary where each URL is a key, and the corresponding prediction (clickbait probability) is the value. Used in the pre-click detection mode.

### Request Payload

```
{
  "url": "string",
  "html": "string"
}
```



**Response**

```
{
  "predictions": {
    "url1": float,
    "url2": float,
    ...
  }
}
```

**8.2. Modes**

The extension has two main modes: post-click and pre-click detection.

**8.2.1. Post-click detection**

The first mode is post-click detection. It scans the content and runs detection after the user enters a desired article page. Its purpose is to showcase what we have achieved and how our models work. What's more, it serves some additional roles:

- **educational** role as prediction explanations are be provided,
- **time-saving** role as it is saving time by spoiling clickbait titles

There are 4 different configurations for this mode.

- **Manual-only**, where the user manually triggers detection by clicking the "check" button in the popup.
- **Monitored**, where the extension automatically checks for clickbaits on sites matching URL patterns provided by the user
- **Semi-automatic**, where the extension automatically checks for clickbaits on sites specified by the user and on pages containing relevant meta tag (`<meta property="og:type" content="article" />`).
- **Automatic**, where the extension checks for clickbaits on every page.

Users can configure and change it in the options panel, and the settings are saved in the browser sync storage, including URLs added to the monitored sites list. Figure 8.2 presents a sequence diagram for semi-automatic post-click detection.

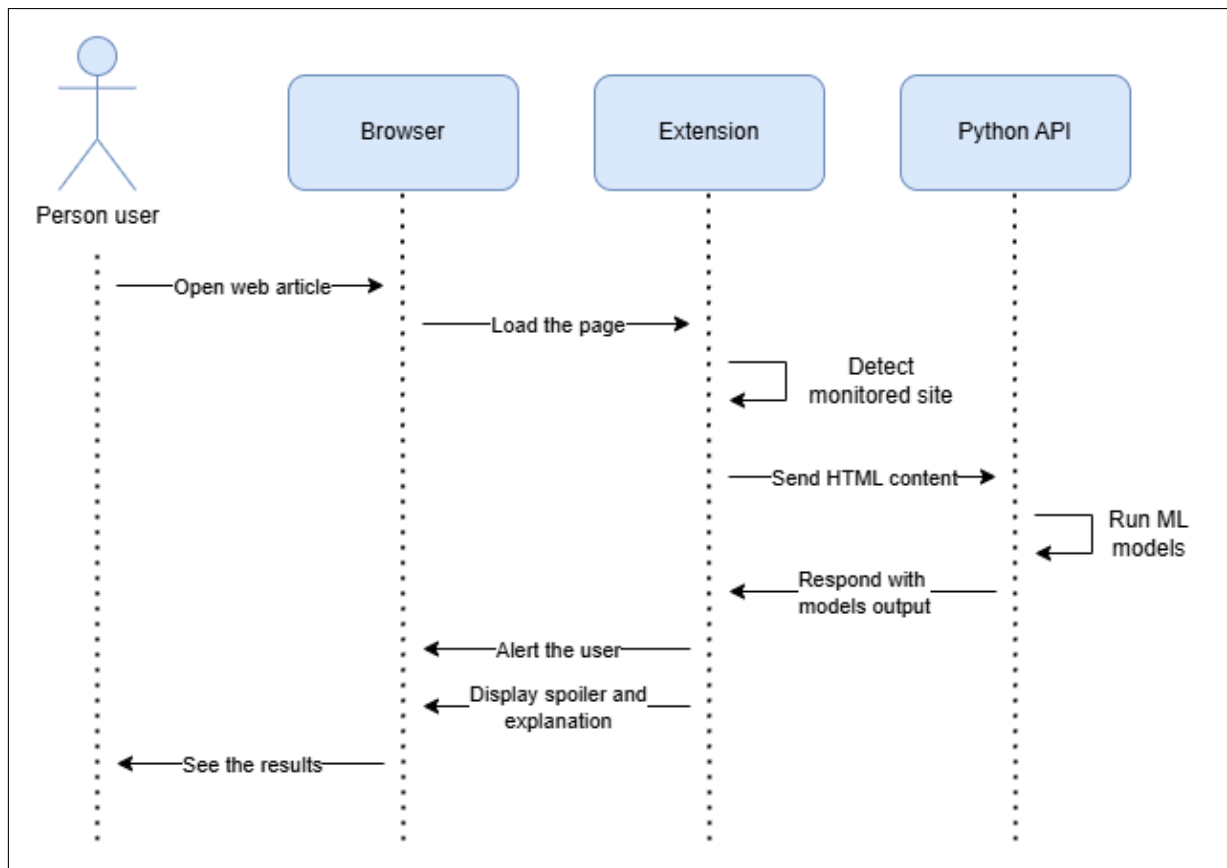


Figure 8.2: Sequence diagram for post-click detection

When a user opens the page, the content script checks if the detection should be run on this site; this includes checking if the page was already processed. If it qualifies for the detection, then HTML content is grabbed and sent in a POST request to the Python API. The same can be achieved manually by the user from the popup, which injects a script with similar functionality into the current page.

In the API, the title and article body are extracted from the HTML using tools mentioned in subsection 8.1.3. Then the data are transformed and passed to the models. After processing, the model outputs are returned.

The data go back to the script, which saves the predictions in the browser's local storage, sets a colored badge and shows a popup to alert the user. The spoiler, explanation, and clickbait probability will appear in the extension popup.

As already mentioned, all predictions are saved to the browser's local storage. Thanks to this, in the future, when a user enters the previously processed page, all the predictions and badges will be instantly visible. That's the role of the service worker script, which handles it on every tab change.

### 8.2.2. Pre-click detection

The main goal of our methods is to warn users in advance. However, it is almost impossible to create a tool that detects clickbaits automatically on every existing website. Every website has a unique structure, and it would require unique configurations for every page existing on the Internet. This task is beyond the time and resources we have got.

Still, to present the potential of our methods to warn users in advance, we added a pre-click detection mode. It's implemented only on selected websites (e.g., Google Search or The Sun). This allows us to highlight that our tool is capable of filtering this type of content while maintaining focus on developing the methods rather than losing our very limited time on creating multiple page configurations.

Page-specific configurations are implemented on the Python API side. The main issue is to define which links are article links and where the title of the article is located. This requires manual inspection of the original page structure to filter it using appropriate class names and IDs. Later in the extension, there is one universal function that sets an adequate icon next to each link returned in the response from the API.

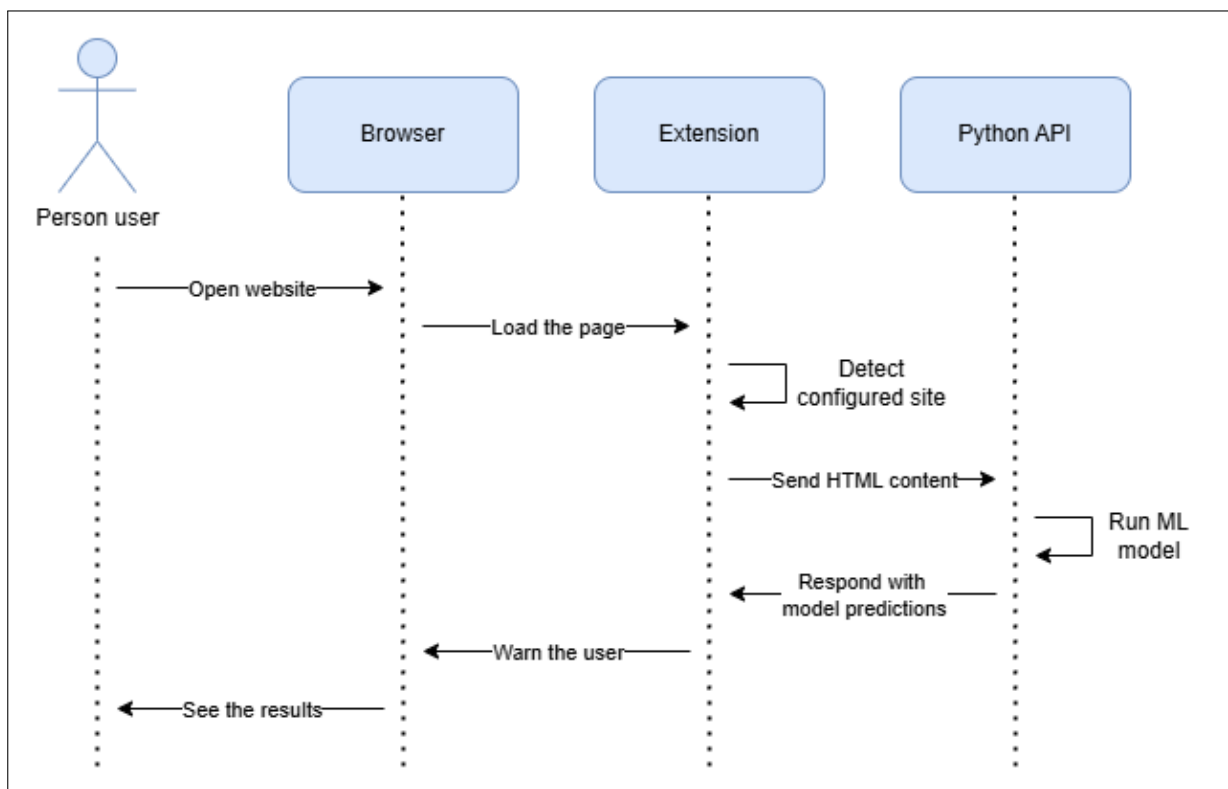


Figure 8.3: Sequence diagram for pre-click detection

At first glance, the sequence diagram for pre-click detection (see Figure 8.3) is similar to the one in Figure 8.2. However, they differ much in functionality, especially in the types of pages that

### 8.3. GUI

are analyzed. Previously it was an article page, now it is a page containing links to the articles. When a user enters such a site, and there is an existing configuration for it, the HTML content is sent over to API. Content scripts are used for it.

In Python API, we use web scraping techniques to extract links and corresponding article titles. This is a fragment that has to be written independently for each website. For each title, the prediction is made, and the data are returned.

The content script then locates corresponding links on the website content and adds appropriate icons next to the link titles to warn users of potential clickbaits.

### 8.3. GUI

In this section, we present user interface design.

#### 8.3.1. Popup

The default popup design can be seen in Figure 8.4. Figure 8.5 shows the exemplary output for the article "How to keep your workout clothes from stinking?". The *check* button is left even after the predictions are made, so the user can always re-run the detection.



Figure 8.4: Before detection

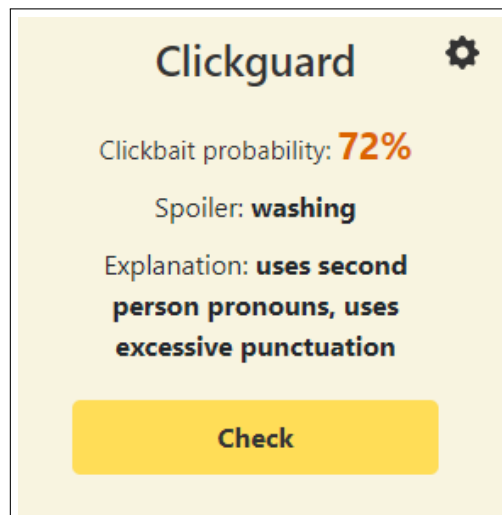


Figure 8.5: After detection

#### 8.3.2. Extension icon and badges

Here, you can find badge designs. Badges are shown in the extension bar on processed article pages in post-click detection. They are super small, hence the resolution of the images is low. If the probability of title being a clickbait is below 0.4, the green icon is shown. If the probability

is between 0.4 and 0.6, a yellow warning sign is displayed. For titles with a clickbait probability exceeding 0.6, a red exclamation mark badge is shown to alert a user.

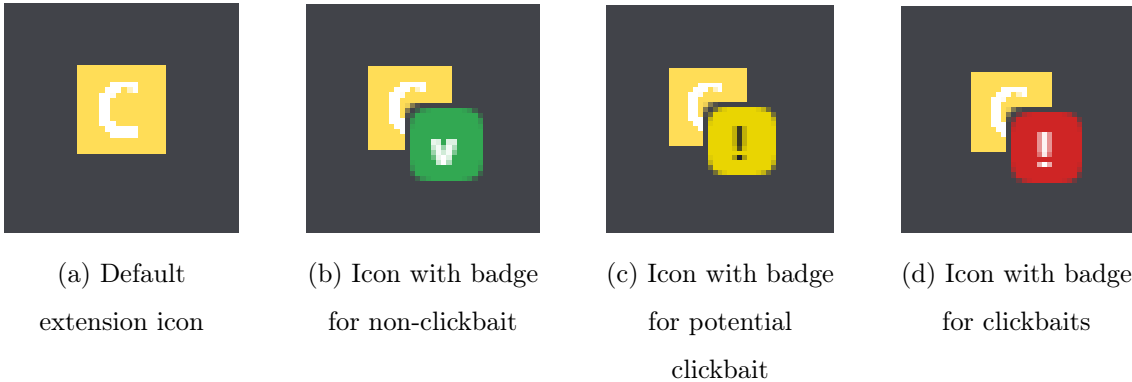


Figure 8.6: Badges

8.3.3. In-page icons

In the pre-click detection, icons are placed to the right of the link titles. As previously, the red one is used for clickbait titles, yellow for potential clickbait, and green for titles with low probability of being clickbait. Examples are shown in Figure 8.7 and in Figure 8.8.

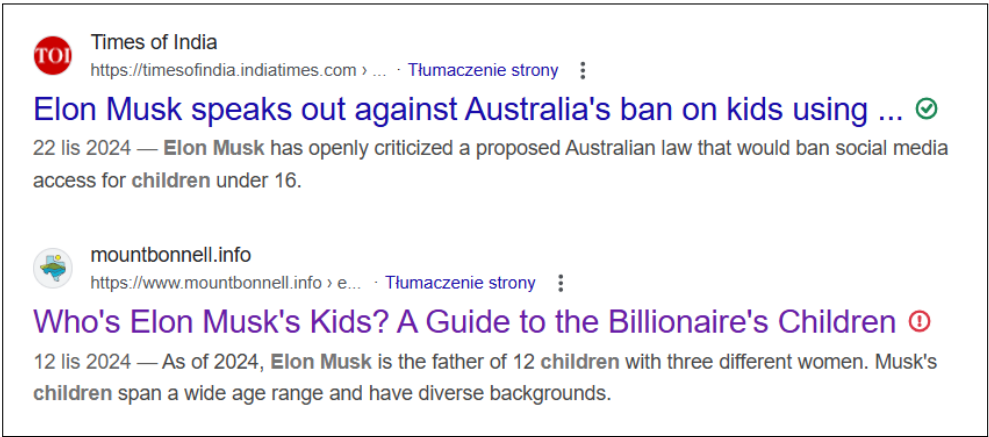


Figure 8.7: Pre-click warning design

### 8.3. GUI

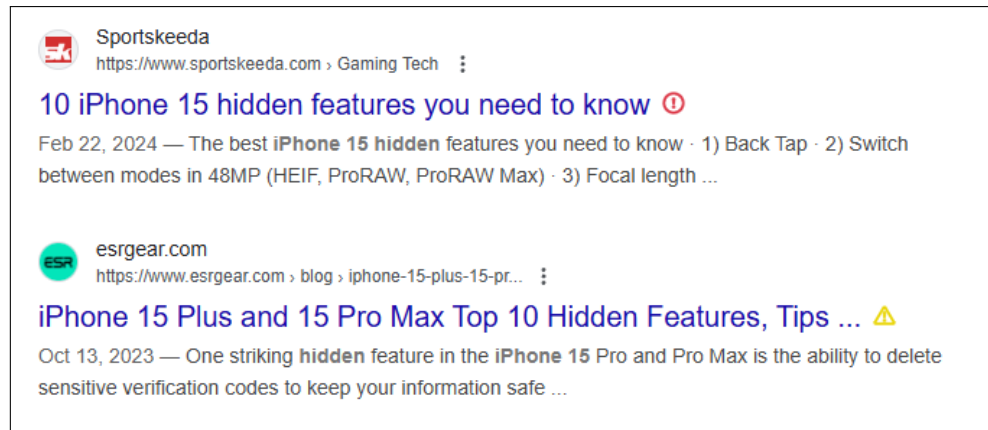


Figure 8.8: Pre-click warning design

#### 8.3.4. Options page

The design of the options page is also important so the user can easily configure the extension to suit their preferences. In Figure 8.9 can be seen the pre-click settings panel. It utilizes toggle switches for turning on and off some options.

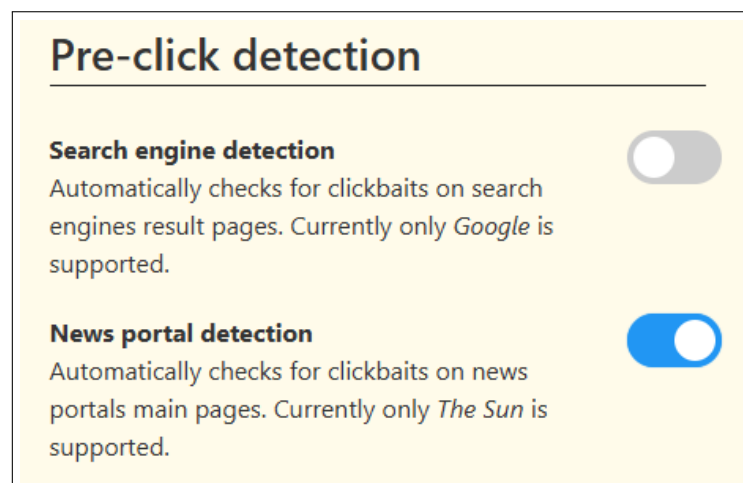


Figure 8.9: Pre-click settings panel

In the post-click detection settings panel (see Figure 8.10), you can choose the detection type using radio buttons.

## Post-click detection

**Manual-Only Detection**  
To check if an article contains a clickbait title, click on the extension icon and then "Check" button in the popup.

☐

**Monitored Detection**  
Automatically checks for clickbaits on monitored URLs.

☐

**Semi-Automatic Detection** (recommended)  
Automatically checks for clickbaits on monitored URLs and on pages containing relevant meta tags.

☒

**Automatic Detection**  
Automatically checks for clickbaits on every page. May slow down the browser.

☐

Save

Figure 8.10: Post-click settings panel

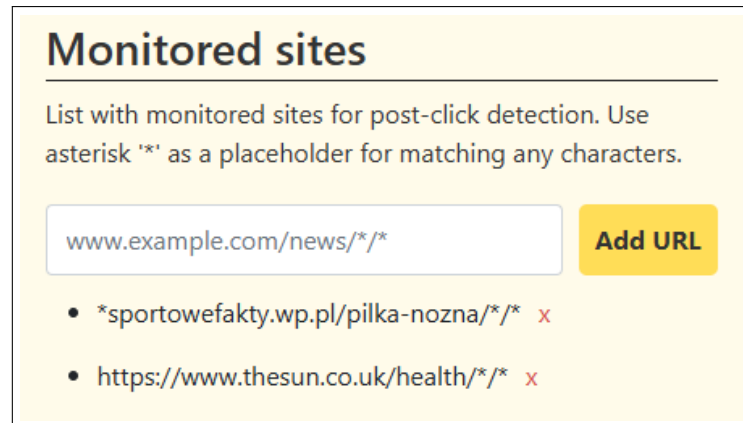
There is also an option to turn on and off spoiler generation as shown in Figure 8.11.

**Spoiler generation**  
Shows article spoiler in the popup, only for articles classified as clickbait. Slows down clickbait alerts.

☒

Figure 8.11: Spoiler generation switch

The last panel of the settings is the monitored sites input (see Figure 8.12), where the user can add URL patterns for monitored and semi-automatic clickbait detection.

The image shows a web interface titled "Monitored sites". Below the title is a text box containing the instruction: "List with monitored sites for post-click detection. Use asterisk '\*' as a placeholder for matching any characters." Below this is a text input field with the value "www.example.com/news/\*/\*" and a yellow button labeled "Add URL". Below the input field is a list of two monitored sites, each followed by a red 'x' icon: "\*sportowefakty.wp.pl/pilka-nozna/\*/\*" and "https://www.thesun.co.uk/health/\*/\*".

**Monitored sites**

List with monitored sites for post-click detection. Use asterisk '\*' as a placeholder for matching any characters.

**Add URL**

- \*sportowefakty.wp.pl/pilka-nozna/\*/\* x
- https://www.thesun.co.uk/health/\*/\* x

Figure 8.12: Monitored sites panel

## 8.4. Testing

### 8.4.1. Unit Tests

For unit testing, we used built-in *unittest* module. Unit tests were performed for all modules on Python API side.

- All utils and helper functions were tested whether they behave correctly, including regex patterns which were tested against many input variations.
- Each informativeness measure was checked whether it is being calculated as expected.
- We tested methods for title extraction to ensure all titles are extracted properly from the HTML files.
- Every function that sends a request to an external server was tested, including asynchronous ones.
- API endpoints were tested to determine whether they accept payloads properly and return correct responses
- ML models and other functions for modeling were checked to see if they correctly handle input and return predictions or generated texts.

### 8.4.2. Integration Tests

For integration tests, we have tested whole pipelines on exemplary data, e.g., we saved an HTML file for a particular site and put it into the pipeline function, which includes titles extraction, calls to OpenAI API, calculating informativeness measures, making predictions and



optionally generating explanations and spoiler. We asserted everything runs smoothly, ensured the data flow is correct, and that the final output is accurate.

### 8.4.3. Performance Tests

Performance tests were run in order to check the time of processing both for post-click and pre-click detection.

For post-click detection, we ran the pipeline 20 times using an exemplary clickbait article and calculated descriptive statistics of the execution times. The results are shown in Table 8.1. It can be seen that generating a spoiler extends execution time three times. Because of that, we decided to add an option to turn off spoiler generation, as it massively impacts the performance of detection pipeline.

Table 8.1: Post-click detection pipeline execution time

Settings	Mean (s)	Std (s)	Median (s)	Min (s)	Max (s)
With spoiling	1.783	0.260	1.724	1.368	2.351
Without spoiling	0.627	0.136	0.627	0.359	0.958

For pre-click detection, we ran similar tests. However, here we have tested the impact of a number of article links on the website on overall execution time. In Table 8.2 we can see that websites with a smaller number of article links are processed really fast (under tenths of seconds), while pages with a lot of links are processed way more time. This is because, for each link, an asynchronous request is sent to open API, so sending many requests simultaneously slows it down.

Table 8.2: Pre-click detection pipeline execution time

Number of links	Mean (s)	Std (s)	Median (s)	Min (s)	Max (s)
10	0.094	0.053	0.073	0.067	0.277
50	2.464	1.252	1.948	1.140	4.846

### 8.4.4. User Interface Tests

GUI was tested manually. We checked whether all buttons and UI elements behave as expected e.g., adding items to monitored sites list, changing settings or running detection manually from the popup. We also ensured that all UI elements and icons are displayed properly.

## 8.4. TESTING

### 8.4.5. Usability tests

We conducted usability tests on five independent users. Their overall satisfaction was good, however they had some concerns:

- They expressed opinion that there should be another icon for clickbait probabilities around 50%, because binary icons provide insufficient insight. This issue has been handled.
- Processing time for post-click detection with spoilers was a little bit too long. This aligns with our performance tests results.
- They also noted that adding more page configurations for pre-click detection would make this tool more useful.

### 8.4.6. Acceptance Tests

The browser extension for clickbait detection was tested in various scenarios. Firstly, we evaluated whether the program correctly classifies clickbaits. We have selected really obvious clickbait article (see Figure 8.13) and one (see Figure 8.14), which should not be marked as clickbait. Everything was classified properly.

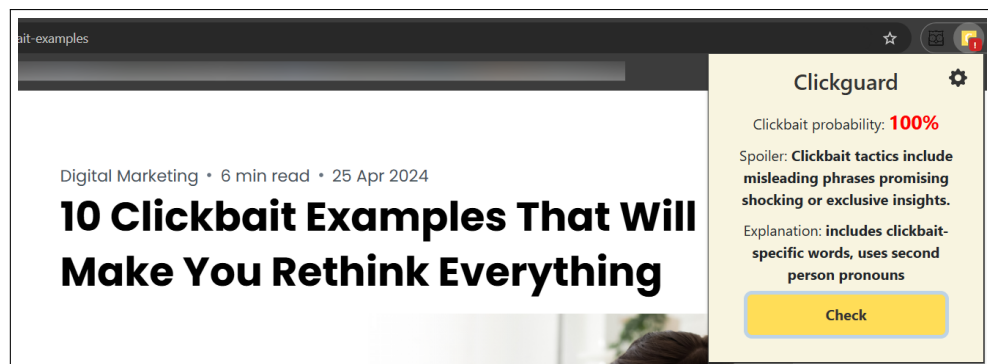


Figure 8.13: Clickbait article properly classified



Figure 8.14: Unlikely (18%) clickbait

Tests also included checking if the generated explanations and spoilers make sense. All were passed, e.g., in Figure 8.15 part of explanation is "uses superlative words", and we can see that in the title there is a word "most".



Figure 8.15: Probable (71%) clickbait with spoiler and explanation

We have tested the predictions also in the pre-click detection. In figure 8.16 it can be seen that articles are classified properly and icons are set correctly.

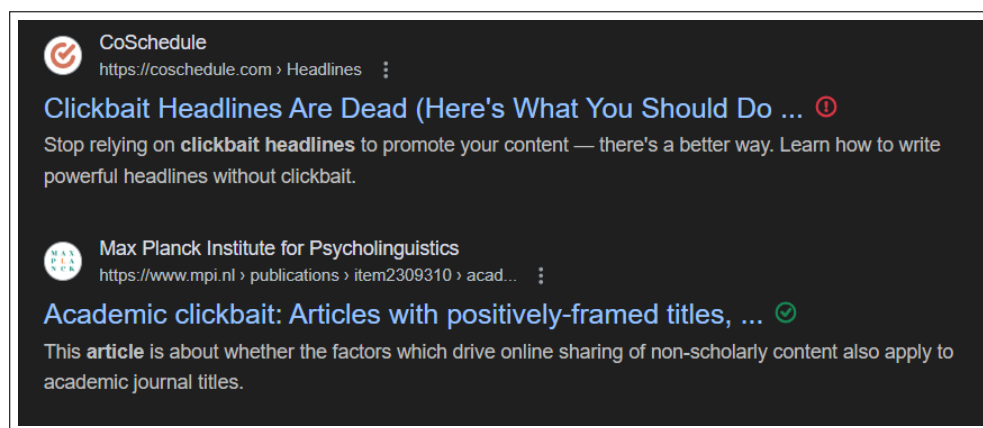


Figure 8.16: Accurate warnings before reading the articles on Google

Moreover, we tested its functionality on the supported page, *The Sun*.

It functions correctly, as adequate icons appear next to each title, indicating potential deceiving articles.

Apart from testing the model's behaviour, all technical aspects of the extension were checked.

- Popup pops up to alert the user only for articles classified as clickbaits.
- Color of the badge set on the extension icon corresponds to the predicted probability.
- The badge itself is visible even after reopening the website.
- Options that were once set are properly saved and remain also after closing the browser.
- Detection based on monitored url patterns works and recognizes correct websites.

## 8.4. TESTING



Figure 8.17: Accurate warnings before reading the articles on The Sun

- Semi automatic recognition based on 'article' meta tag behaves correctly and detection is run automatically after entering sites tagged as an article.
- Turning on and off spoiler generation works good.
- Turning on and off both news portal and search engine pre-detection works and predictions are made automatically on configured sites (Google and The Sun).

App was also tested and accepted by the supervisor, cooperators, and end users. Everything worked as expected and nobody had any issues.

## 9. Summary

In this thesis, we explored the problem of clickbait detection and developed machine learning models and a browser extension that warns internet users about the potential threat of deceptive online content. Our title-based model with the best performance leveraged OpenAI embeddings combined with 15 custom measures, achieving an F1 score of 0.909. We also introduced our own custom *baitness* measure. A key contribution of our work is a browser extension that integrates the model to provide real-time feedback on the likelihood of a headline being clickbait. It has a friendly layout, and there are multiple settings, including post-click and pre-click detection. The API for the extension is hosted on the Google Cloud Platform.

Through this project, we learned to build browser extensions, leverage large language models, write effective prompts, and fine-tune machine learning models in NLP to achieve optimal results. Additionally, it provided an excellent opportunity to strengthen teamwork, utilize GitHub effectively, and collaborate with our partners.

### 9.1. Future work and limitations

During the development of our machine learning models and browser extension, numerous ideas for future work emerged. Improvements in machine learning models could be achieved by collecting more datasets or even creating our own data, which would enhance the model's reliability and robustness. For the browser extension, the user interface (UI) could be enhanced to improve its visual appeal and usability. Another potential feature would be displaying the likelihood of clickbait next to each article in the pre-click detection mode, which could provide users with a more nuanced understanding of content quality.

One major limitation is the restricted accessibility of the browser extension, as it has not yet been published on platforms such as the Chrome Web Store. This limits its reach to a broader audience. Additionally, currently the browser extension supports only chromium-based browsers. In addition, the pre-click detection mode can be used only on pre-configured websites. Because of that, this tool is not as universal as it could be.

## Bibliography

- [1] A. Razaque, B. Alotaibi, M. Alotaibi, S. Hussain, A. Alotaibi, and V. Jotsov, “Clickbait detection using deep recurrent neural network,” *Applied Sciences*, vol. 12, no. 1, 2022. Accessed: 29 October 2024. DOI: 10.3390/app12010504.
- [2] Y. Liu, T. Han, S. Ma, J. Zhang, Y. Yang, J. Tian, H. He, A. Li, M. He, Z. Liu, Z. Wu, L. Zhao, D. Zhu, X. Li, N. Qiang, D. Shen, T. Liu, and B. Ge, “Summary of chatgpt-related research and perspective towards the future of large language models,” *Meta-Radiology*, vol. 1, p. 100017, Sept. 2023. Accessed: 29 October 2024. DOI: 10.1016/j.metrad.2023.100017.
- [3] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “Roberta: A robustly optimized bert pretraining approach,” *arXiv*, 07 2019. Accessed: 28 October 2024. DOI: 10.48550/arXiv.1907.11692.
- [4] J. Patel and A. Prajapati, “Cline ai chrome extension.” [https://github.com/Janakarpatel/CliNe.AI\\_Chrome-Extension](https://github.com/Janakarpatel/CliNe.AI_Chrome-Extension), 2023. Accessed: 22 January 2025.
- [5] “Clickbaitdetector website.” <https://clickbaitdetector.com/>. Accessed: 22 January 2025.
- [6] E. H. Marinho and R. F. Resende, “Quality factors in development best practices for mobile applications,” in *Computational Science and Its Applications – ICCSA 2012* (B. Murgante, O. Gervasi, S. Misra, N. Nedjah, A. M. A. C. Rocha, D. Taniar, and B. O. Apduhan, eds.), (Berlin, Heidelberg), pp. 632–645, Springer Berlin Heidelberg, 2012. Accessed: 29 October 2024. DOI: 10.1007/978-3-642-31128-4\_47.
- [7] E. Gürel, “Swot analysis: A theoretical review,” *Journal of International Social Research*, vol. 10, pp. 994–1006, 08 2017. Accessed: 29 October 2024. DOI: 10.17719/jisr.2017.1832.
- [8] A. Anand, “Clickbait dataset.” <https://www.kaggle.com/datasets/amananandrai/clickbait-dataset/>, 2019. Accessed: 28 October 2024.

- [9] V. Singh, “News clickbait dataset.” <https://www.kaggle.com/datasets/vikassingh1996/news-clickbait-dataset>, 2020. Accessed: 28 October 2024.
- [10] “Clickbait detection challenge 2017.” <https://webis.de/events/clickbait-challenge/shared-task.html>, 2017. Accessed: 28 October 2024.
- [11] M. Potthast, T. Gollub, M. Hagen, and B. Stein, “The clickbait challenge 2017: Towards a regression model for clickbait strength.” <https://arxiv.org/abs/1812.10847>, 2018. Accessed: 28 October 2024.
- [12] “Clickbait challenge at semeval 2023 - clickbait spoiling.” <https://pan.webis.de/semeval23/pan23-web/clickbait-challenge.html>, 2023. Accessed: 28 October 2024.
- [13] M. Hagen, M. Fröbe, A. Jurk, and M. Potthast, “Clickbait spoiling via question answering and passage retrieval,” *arXiv*, 03 2022. Accessed: 29 October 2024. DOI: 10.48550/arXiv.2203.10282.
- [14] C. Chen and L. Breiman, “Using random forest to learn imbalanced data.” [https://www.researchgate.net/publication/254196943\\_Using\\_Random\\_Forest\\_to\\_Learn\\_Imbalanced\\_Data](https://www.researchgate.net/publication/254196943_Using_Random_Forest_to_Learn_Imbalanced_Data), 01 2004. Accessed: 30 October 2024.
- [15] A. Géron, “Hands-on machine learning with scikit-learn, keras and tensorflow: Concepts, tools, and techniques to build intelligent systems.” [http://14.139.161.31/OddSem-0822-1122/Hands-On\\_Machine\\_Learning\\_with\\_Scikit-Learn-Keras-and-TensorFlow-2nd-Edition-Aurelien-Geron.pdf](http://14.139.161.31/OddSem-0822-1122/Hands-On_Machine_Learning_with_Scikit-Learn-Keras-and-TensorFlow-2nd-Edition-Aurelien-Geron.pdf), 2019. Accessed: 8 November 2024.
- [16] R. S. Petr Silhavy, “Artificial intelligence application in networks and systems.” [https://www.google.pl/books/edition/Artificial\\_Intelligence\\_Application\\_in\\_N/TmjKEAAQBAJ?hl=en&gbpv=0](https://www.google.pl/books/edition/Artificial_Intelligence_Application_in_N/TmjKEAAQBAJ?hl=en&gbpv=0), 2023. Accessed: 11 November 2024.
- [17] D. Jurafsky and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, vol. 2. 02 2008. Accessed: 29 October 2024.
- [18] M. N. C. B. Centre Borelli, Stanislas Morbieu Kernix, “A survey on recent advances in named entity recognition.” <https://arxiv.org/pdf/2401.10825v1>, 01 2024. Accessed: 8 November 2024.
- [19] “Spacy library documentation.” <https://spacy.io/>. Accessed: 1 November 2024.

- [20] S. S. Dipankar Das, Anup Kumar Kolya, “Computational intelligence applications for text and sentiment data analysis.” <https://www.sciencedirect.com/science/article/abs/pii/B9780323905350000112>, 2023. Accessed: 8 November 2024.
- [21] P. Shah, “Sentiment analysis using textblob and its working!.” <https://towardsdatascience.com/sentiment-analysis-using-textblob-and-its-working-2020>, June 2020. Accessed: 5 November 2024.
- [22] “Langid library documentation.” <https://pypi.org/project/langid/1.1dev/>. Accessed: 1 November 2024.
- [23] H. H. Hobson Lane and C. Howard, *Natural Language Processing in Action*. Manning Publications, 2019.
- [24] J. Yu, S. Wang, H. Yin, Z. Sun, R. Xie, B. Zhang, and Y. Rao, “Multimodal clickbait detection by de-confounding biases using causal representation inference.” <https://arxiv.org/abs/2410.07673>, 2024. Accessed: 29 October 2024.
- [25] M. Reisenbichler, T. Reutterer, D. A. Schweidel, and D. Dan, “Frontiers: Supporting content marketing with natural language generation.” <https://doi.org/10.1287/mksc.2022.1354>, 2022. Accessed: 29 October 2024.
- [26] J. Torruella and R. Capsada, “Lexical statistics and tipological structures: A measure of lexical richness.” [https://www.researchgate.net/publication/273851867\\_Lexical\\_Statistics\\_and\\_Tipological\\_Structures\\_A\\_Measure\\_of\\_Lexical\\_Richness](https://www.researchgate.net/publication/273851867_Lexical_Statistics_and_Tipological_Structures_A_Measure_of_Lexical_Richness), 2013. Accessed: 30 October 2024.
- [27] F. deBoer, “Evaluating the comparability of two measures of lexical diversity.” <https://www.sciencedirect.com/science/article/pii/S0346251X14001742>, 2014. Accessed: 30 October 2024.
- [28] A. Chowanda, N. Nadia, and L. Kolbe, “Identifying clickbait in online news using deep learning,” *Bulletin of Electrical Engineering and Informatics*, vol. 12, pp. 1755–1761, 06 2023. Accessed: 29 October 2024. DOI: 10.11591/eei.v12i3.4444.
- [29] C. Wu, F. Wu, T. Qi, and Y. Huang, “Clickbait detection with style-aware title modeling and co-attention,” in *Proceedings of the 19th Chinese National Conference on Computational Linguistics* (M. Sun, S. Li, Y. Zhang, and Y. Liu, eds.), (Haikou, China), pp. 1143–1154, Chinese Information Processing Society of China, Oct. 2020. <https://aclanthology.org/2020.cc1-1.106/>. Accessed: 29 October 2024.



- [30] “Clickbait dataset.” <https://www.kaggle.com/datasets/amananandrai/clickbait-dataset>, 2019. Accessed: 30 October 2024.
- [31] “Fake news challange.” <http://www.fakenewschallenge.org/>, 2016. Accessed: 31 October 2024.
- [32] “Word2vec Google documentation.” <https://code.google.com/p/word2vec/>, 2013. Accessed: 28 October 2024.
- [33] R. Singh, “Featurization of text data. Medium.” <https://medium.com/analytics-vidhya/featurization-of-text-data-bow-tf-idf-avgw2v-tfidf-weighted-w2v-7a6c62e8b097>, 2019. Accessed: 28 October 2024.
- [34] J. Lilleberg, Y. Zhu, and Y. Zhang, “Support vector machines and word2vec for text classification with semantic features,” in *2015 IEEE 14th International Conference on Cognitive Informatics Cognitive Computing (ICCI\*CC)*, pp. 136–140, 2015. <https://ieeexplore.ieee.org/document/7259377>.
- [35] J. Pennington, R. Socher, and C. Manning, “GloVe: Global vectors for word representation.” <https://aclanthology.org/D14-1162>, Oct. 2014. Accessed: 28 October 2024.
- [36] J. von der Mosel, A. Trautsch, and S. Herbold, “On the validity of pre-trained transformers for natural language processing in the software engineering domain,” *IEEE Transactions on Software Engineering*, vol. 49, pp. 1487–1507, Apr. 2023. Accessed: 29 October 2024. DOI: 10.1109/TSE.2022.3178469.
- [37] G. Biau, “Analysis of a random forests model,” *Journal of Machine Learning Research*, vol. 13, 05 2010. <https://arxiv.org/abs/1005.0208>. Accessed: 29 October 2024.
- [38] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, pp. 785–794, ACM, Aug. 2016. Accessed: 29 October 2024. DOI: 10.1145/2939672.2939785.
- [39] R. Fiagbe, *High-Dimensional Random Forests*. PhD thesis, 05 2021. [https://scholarworks.utep.edu/cgi/viewcontent.cgi?article=4251&context=open\\_etd](https://scholarworks.utep.edu/cgi/viewcontent.cgi?article=4251&context=open_etd). Accessed: 2 November 2024.
- [40] Y. HaCohen-Kerner, D. Miller, and Y. Yigal, “The influence of preprocessing on text classification using a bag-of-words representation,” *PLoS ONE*, vol. 15, no. 5, p. e0232525, 2020. Accessed: 29 October 2024. DOI: 10.1371/journal.pone.0232525.

- [41] A. K. Uysal and S. Gunal, “The impact of preprocessing on text classification,” *Information Processing Management*, vol. 50, pp. 104–112, 01 2014. Accessed: 28 October 2024. DOI: 10.1016/j.ipm.2013.08.006.
- [42] “Nltk documentation.” <https://www.nltk.org/team.html>. Accessed: 28 October 2024.
- [43] “Tf-idf documentation.” [https://scikit-learn.org/1.5/modules/generated/sklearn.feature\\_extraction.text.TfidfVectorizer.html](https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html). Accessed: 28 October 2024.
- [44] R. Řehůřek, “Gensim word2vec documentation.” <https://radimrehurek.com/gensim/models/word2vec.html>, 2024. Accessed: 28 October 2024.
- [45] Kaggle, “Glove 100 dimensional.” <https://www.kaggle.com/datasets/danielwillgeorge/glove6b100dtxt>. Accessed: 31 October 2024.
- [46] OpenAI, “Rate limits.” <https://platform.openai.com/docs/guides/rate-limits>, 2024. Accessed: 31 October 2024.
- [47] OpenAI, “Prompt engineering.” <https://platform.openai.com/docs/guides/prompt-engineering>, 2024. Accessed: 31 October 2024.
- [48] Christinacdl, “Xlm roberta clickbait detection new.” [https://huggingface.co/christinacdl/XLM\\_RoBERTa-Clickbait-Detection-new](https://huggingface.co/christinacdl/XLM_RoBERTa-Clickbait-Detection-new). Accessed: 1 November 2024.
- [49] “Semeval-2023 task 5: Clickbait spoiling.” [https://downloads.webis.de/publications/papers/froebe\\_2023d.pdf](https://downloads.webis.de/publications/papers/froebe_2023d.pdf), 2023. Accessed: 30 October 2024.
- [50] OpenAI, “Pricing.” <https://openai.com/api/pricing/>, 2024. Accessed: 26 November 2024.
- [51] A. Lavie and A. Agarwal, “Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments,” pp. 228–231, 07 2007. <https://api.semanticscholar.org/CorpusID:16289845>. Accessed: 29 October 2024.
- [52] H. Face, “Transformers.” <https://huggingface.co/docs/transformers/en/index>, 2024. Accessed: 26 November 2024.
- [53] F. W. K. Q. W. Tianyi Zhang\*, Varsha Kishore and Y. Artzi, “Bertscore.” <https://pypi.org/project/bert-score/>, 2023. Accessed: 26 November 2024.
- [54] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in *North American Chapter of the Association for Computational Linguistics*, 2019. Accessed: 29 October 2024. DOI: 10.18653/v1/N19-1423.

- [55] H. F. bert-base-uncased model. <https://huggingface.co/google-bert/bert-base-uncased>. Accessed: 08 December 2024.
- [56] M. Grinberg, “Flask web development: developing web applications with python.” [https://coddyschool.com/upload/Flask\\_Web\\_Development\\_Developing.pdf](https://coddyschool.com/upload/Flask_Web_Development_Developing.pdf), 2018. Accessed: 31 October 2024.
- [57] L. Richardson, “Beautiful soup documentation.” <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>, 2007. Accessed: 31 October 2024.
- [58] A. Barbaresi, “Trafilatura: A web scraping library and command-line tool for text discovery and extraction,” pp. 122–131, 01 2021. Accessed: 29 October 2024. DOI: 10.18653/v1/2021.acl-demo.15.

## List of symbols and abbreviations

Application Programming Interface (API)	Connection between computer programs as a software interface. An API offers a service to other pieces of software.
Article	A written piece of content on a particular subject that is published online, typically on news websites or blogs. Usually, it consists of a title, body, and multimedia elements like images, videos, and hyperlinks. They may vary in their format and can be opinion pieces, short news reports, or research analyses.
Browser Extension	Functional module in the form of a programming script for customizing a web browser. Extensions add new features or modify browser behavior.
Clickbait	<p>An internet story, title, image, etc., that is intended to attract attention and encourage people to click on a link. Examples of clickbait article titles:</p> <ul style="list-style-type: none"><li>• “6 Things Most Doctors Won’t Tell You About Dieting.”</li><li>• “This Simple Investment Trick Could Make You Rich Overnight.”</li><li>• “She Tried This Weird Hack to Lose Weight—The Results Will Shock You!”</li></ul> <p>Very often they generate strong emotions and exaggerate information, leading users to click on a link and/or read an article. Sometimes an answer to the article is placed at the very end, deliberately making you scroll through the entire article so that you spend as much time as possible on their website.</p>

Clickbait detection	Process of determining if an article may be a potential clickbait.
Clickbait spoiling	Generating a brief text (or finding a fragment) being a response to a clickbait title.
Exploratory Data Analysis (EDA)	Analyzing datasets to summarize their characteristics using statistical graphics and data visualization techniques.
Machine Learning (ML)	A field of study in artificial intelligence with statistical algorithms that can learn from data and generalize to unseen data. Therefore, it can perform tasks, such as predicting values, without explicit instructions.
Natural Language Processing (NLP)	A subfield of artificial intelligence which is concerned with providing the ability to process data encoded in written or spoken text. Typically used in machine learning and deep learning.

## List of Figures

1.1	Visual abstract of the thesis . . . . .	12
2.1	Main browser extension logic use case diagram . . . . .	15
4.1	Sankey diagram of clickbait distribution by dataset . . . . .	22
4.2	Length of titles in detection datasets . . . . .	24
4.3	Distribution of alphabetical characters in titles after train/test/val split in click- bait detection dataset . . . . .	25
4.4	Length of titles with texts classified as outliers by the IQR criterion (see Section 4.2.3) . . . . .	27
4.5	Most frequent N-grams in titles . . . . .	29
4.6	Most frequent stop words in titles . . . . .	30
4.7	Named Entity Recognition in titles . . . . .	31
4.8	Sentiment analysis in titles . . . . .	33
4.9	Distribution of Spoiler type by Tag . . . . .	35
5.1	Distribution of metrics related to capital letters . . . . .	40
5.2	Distribution of <i>baitness</i> score on the test set . . . . .	44
5.3	ROC curve for <i>baitness</i> score used for classification . . . . .	45
6.1	Confusion matrix for raw dataset . . . . .	55
6.2	Confusion matrix for cleaned dataset . . . . .	55
6.3	Confusion matrix for Random Forest model trained on raw dataset . . . . .	56
6.4	Effect of number of dimensions of embeddings on F1-score . . . . .	61
6.5	Confusion matrix for <i>GPT-4o-mini</i> model with second prompt . . . . .	64
6.6	Confusion matrix for the XGBoost model using 1000-dimensional embeddings combined with informativeness measures . . . . .	66
7.1	Cosine Similarity between generated and original texts . . . . .	75

7.2	BLEU similarity between generated and original texts, weights [0.40, 0.30, 0.20, 0.10] (log scale) . . . . .	76
7.3	METEOR similarity between generated and original texts . . . . .	77
7.4	BERT-Score between generated and original texts . . . . .	78
7.5	Data points of BERT-Score among Spoilers and Paragraphs . . . . .	79
7.6	Data points of BLEU score among Spoilers and Paragraphs . . . . .	80
8.1	System architecture . . . . .	83
8.2	Sequence diagram for post-click detection . . . . .	87
8.3	Sequence diagram for pre-click detection . . . . .	88
8.4	Before detection . . . . .	89
8.5	After detection . . . . .	89
8.6	Badges . . . . .	90
8.7	Pre-click warning design . . . . .	90
8.8	Pre-click warning design . . . . .	91
8.9	Pre-click settings panel . . . . .	91
8.10	Post-click settings panel . . . . .	92
8.11	Spoiler generation switch . . . . .	92
8.12	Monitored sites panel . . . . .	93
8.13	Clickbait article properly classified . . . . .	95
8.14	Unlikely (18%) clickbait . . . . .	95
8.15	Probable (71%) clickbait with spoiler and explanation . . . . .	96
8.16	Accurate warnings before reading the articles on Google . . . . .	96
8.17	Accurate warnings before reading the articles on The Sun . . . . .	97

## List of Tables

1.1	The contribution of the authors of the thesis . . . . .	13
2.1	Functional requirements . . . . .	16
2.2	FURPS Analysis . . . . .	16
2.3	SWOT Analysis . . . . .	17
3.1	Summary of clickbait detection and spoiling datasets . . . . .	20
4.1	Character composition analysis of clickbait detection datasets . . . . .	23
4.2	Average token length for titles in clickbait detection datasets . . . . .	24
4.3	Merged dataset: descriptive statistics by clickbait status . . . . .	26
4.4	Named entity recognition categories . . . . .	31
4.5	Null counts for the SemEval Spoiling Dataset dataset . . . . .	34
4.6	Spoiler examples by Tag in 'tags' column . . . . .	34
4.7	Spoiling dataset: descriptive statistics by spoiler tag . . . . .	35
5.1	Informativeness measures used . . . . .	37
5.2	Measures averages for clickbait and non-clickbait titles . . . . .	39
5.3	Lexical diversity of merged clickbait titles and merged non-clickbait titles . . . . .	40
5.4	Measures averages for clickbait and non-clickbait articles . . . . .	41
5.5	<i>Baitness</i> measure statistics compared against clickbaits on the test set . . . . .	44
6.1	Performance metrics for models with informativeness measures as features . . . . .	53
6.2	Performance metrics for Random Forest models on Clickbait Challenge dataset . . . . .	54
6.3	Performance metrics for TF-IDF with LDA . . . . .	55
6.4	Performance metrics for raw and cleaned dataset . . . . .	56
6.5	Performance metrics for Random Forest and XGBoost models using Word2Vec embeddings . . . . .	58
6.6	Performance metrics for Random Forest and XGBoost models using GloVe embeddings . . . . .	58



6.7	Performance metrics for Random Forest and XGBoost models for TF-IDF weighted Word2Vec . . . . .	59
6.8	Performance metrics for Random Forest and XGBoost models using OpenAI embeddings . . . . .	61
6.9	Performance metrics for direct prompting with OpenAI . . . . .	63
6.10	Performance metrics for fine-tuned version of RoBERTa model . . . . .	64
6.11	Measures used for clickbait detection . . . . .	65
6.12	Performance metrics for OpenAI embeddings combined with Informativeness measures . . . . .	66
7.1	Clickbait spoiling metrics with descriptions . . . . .	69
7.2	Article and Spoiler Data with Extracted Paragraphs . . . . .	72
7.3	Mean values of performance metrics for generated spoilers . . . . .	78
7.4	Spoiler example generated by <i>RoBERTa</i> with <i>BestMatch25</i> . . . . .	81
8.1	Post-click detection pipeline execution time . . . . .	94
8.2	Pre-click detection pipeline execution time . . . . .	94

## List of appendices

1. Environments
2. Installation manual
3. User manual

## 1. Environments

### 1.1. Development environment

Our app consists of two main modules - browser extension and Python API. For development purposes, Python API was run locally. In order to do that:

- Python 3.11 or higher must be installed
- Optionally create a virtual environment
- Install all necessary packages

```
$ pip install -r requirements
```

- After that start the service

```
$ python main.py
```

- Server will be running on `http://127.0.0.1:8080`.
- In order to allow browser extension to communicate with the API, in *background.js* set `PROXY_URL` variable to `http://127.0.0.1:8080`.

Browser extension was developed writing plain JS scripts, which were then loaded into chrome browser for testing and running the app.

### 1.2. Production environment

Production deployment is done with Google Cloud Platform, but only for the python API. The browser extension does not require any dedicated configuration or infrastructure - it is deployed simply by loading it into Chrome in unpacked form. GCP supports Flask apps and can handle sophisticated web configuration automatically, which makes deployment very easy. In order to run the production one has to do the following steps.

## 1.2. PRODUCTION ENVIRONMENT

- Open cloud shell on GCP
- Pull the code from Github repository. Flask server initialization should be in file named *main.py*, and all requirements should be added to *requirements.txt*. Only then Google Cloud will be able to correctly install all dependencies and run the app.
- Run deployment command and specify environment variables including `OPEN_API_KEY` and `RESTRICTED` flag, which indicates whether API endpoints should have access restricted only to extensions.


```
$ gcloud run deploy --source . --set-env-vars  
OPEN_API_KEY=<key>,RESTRICTED=True
```

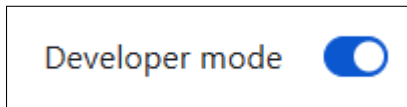
- After running the command type project name (e.g. clickguard) and choose adequate server region (e.g. Europe Central).
- After that, container with the app is created, requirements are installed and app is deployed.

Thanks to the service being deployed, browser extension can communicate with it from every device having internet access.

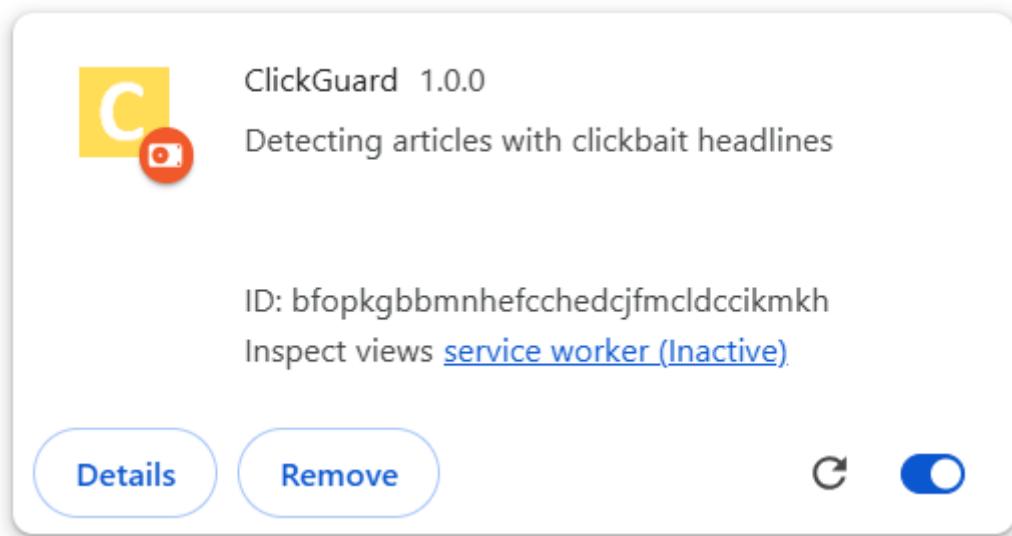
## 2. Installation manual

To start using our app, one should follow the steps below.

- Download the folder containing extension files. If it was downloaded as a zip, unpack it.
- Open a browser, click on the extensions icon  and go to *manage extensions* page.
- Turn on developer mode using a switch in the upper right corner.



- Click *load unpacked* button in the top left corner. In the popup select previously downloaded folder with the extension files.
- A new tile presenting the extension should appear.



- In order to use all functionalities, pin the extension to the bar by clicking pin icon.



## 3. User manual

### 3.1. Overview

Extension has two main modes: post-click detection, for running detection after entering an article page, and pre-click detection, for detecting clickbaits beforehand. Main UI element is a popup, where you can trigger actions manually or set them to trigger automatically.

#### 3.1.1. Popup

In order to start using the extension, click on the yellow extension icon.



After that, popup will appear. In the popup user can do several things.

- Access options page by clicking on the gear icon.
- Perform manual check by clicking yellow *Check* button.
- View predictions results.

The default popup design can be seen in Figure 3.1. Figure 3.2 shows the prediction output, which consists of clickbait probability, explanation and optionally spoiler.



Figure 3.1: Before detection

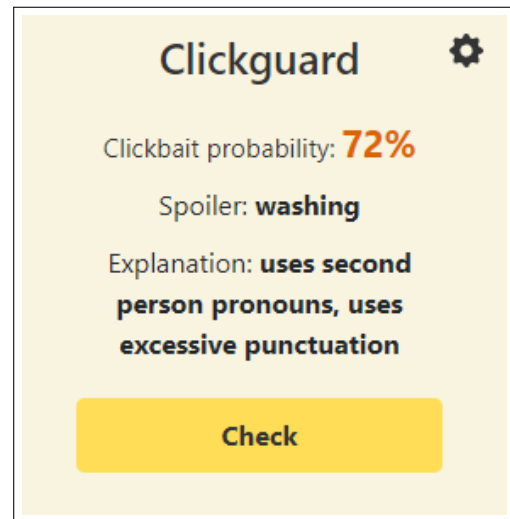


Figure 3.2: After detection

### 3.1.2. Options page

In the options page, user can configure the behaviour of the app. Settings are saved in the browser sync storage. More detailed description of available options is presented in next sections.

### 3.1.3. Badges and Icons

The extension icon is used not only to trigger the popup, but also to display badges. Badges are shown in the extension bar on processed article pages in post-click detection. If the probability of title being a clickbait is below 0.4, the green icon is shown. If the probability is between 0.4 and 0.6, a yellow warning sign is displayed. For titles with a clickbait probability exceeding 0.6, a red exclamation mark badge is shown to alert a user.

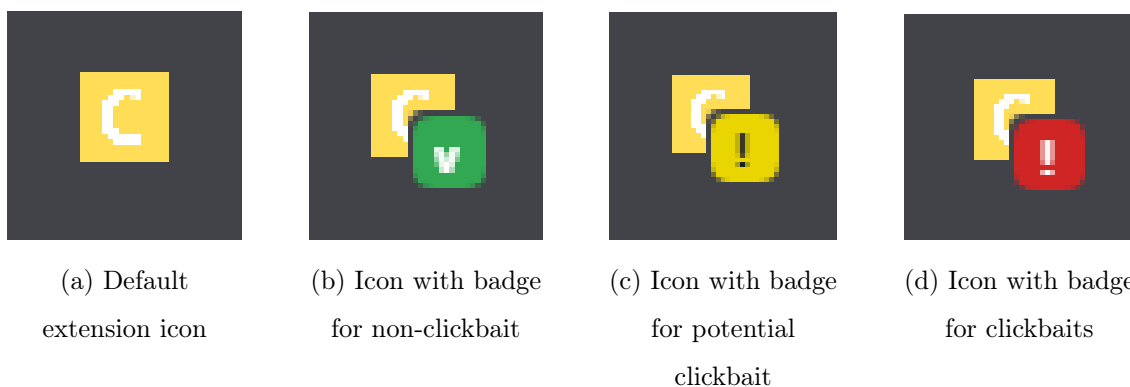


Figure 3.3: Badges

## 3.2. POST-CLICK DETECTION

In the pre-click detection, icons are placed to the right of the link titles. As previously, the red one is used for clickbait titles, yellow for potential clickbait, and green for titles with low probability of being clickbait. Examples are shown in Figure 3.4 and in Figure 3.5.

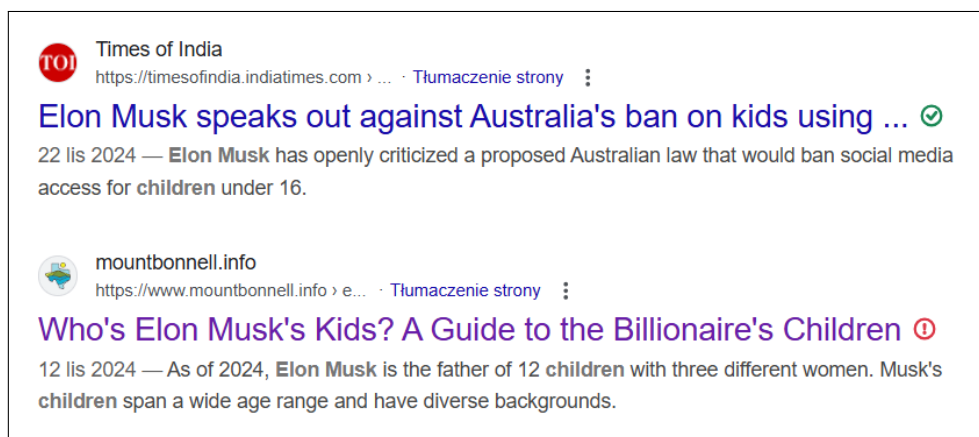


Figure 3.4: Pre-click warning design

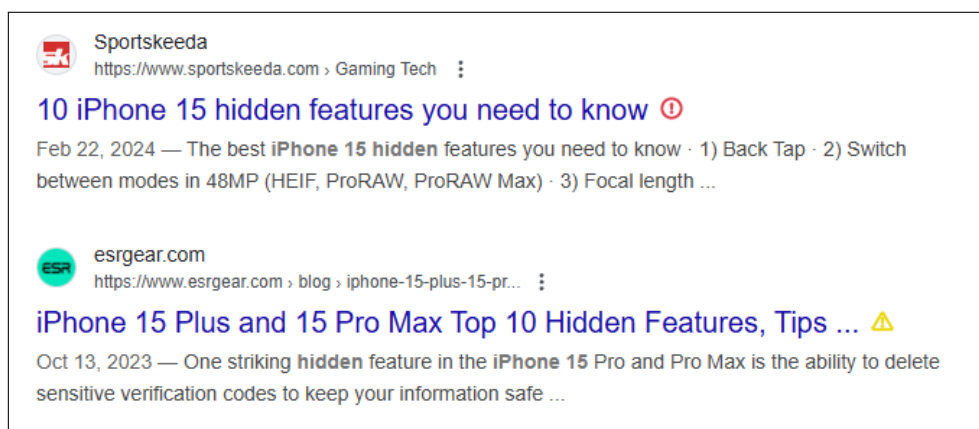


Figure 3.5: Pre-click warning design

## 3.2. Post-click detection

The first mode is post-click detection. It scans the content and runs detection after the user enters a desired article page.

### 3.2.1. Configuration

There are 4 different configurations for this mode.

- **Manual-only**, where the user manually triggers detection by clicking the "check" button in the popup.



- **Monitored Detection**, where the extension automatically checks for clickbaits on sites specified by the user.
- **Semi-automatic**, where the extension automatically checks for clickbaits on sites specified by the user and on pages containing relevant meta tag (`<meta property="og:type" content="article" />`).
- **Automatic**, where the extension checks for clickbaits on every page.

Preferred detection option can be chosen in the options page. To save the choice, click on the 'save' button.

**Post-click detection**

**Manual-Only Detection**  
To check if an article contains a clickbait title, click on the extension icon and then "Check" button in the popup.

**Monitored Detection**  
Automatically checks for clickbaits on monitored URLs.

**Semi-Automatic Detection (recommended)**  
Automatically checks for clickbaits on monitored URLs and on pages containing relevant meta tags.

**Automatic Detection**  
Automatically checks for clickbaits on every page. May slow down the browser.

**Save**

Figure 3.6: Post-click settings panel

### 3.2.2. Adding sites to the monitored sites list

Sites can be added to the list (as well as deleted) in the options page. Use asterix to match any characters. Asterix is added at the beginning, if the provided pattern does not start with *http*.

#### HINT:

When adding a url, try to represent the article url patterns. For example, *\*thesun.co.uk/\** is

### 3.3. PRE-CLICK DETECTION

too loose and will match also home pages. Pattern *\*thesun.co.uk/\*/\*/\*/* properly matches url schema *https://www.thesun.co.uk/<section>/<id>/<title>/*.

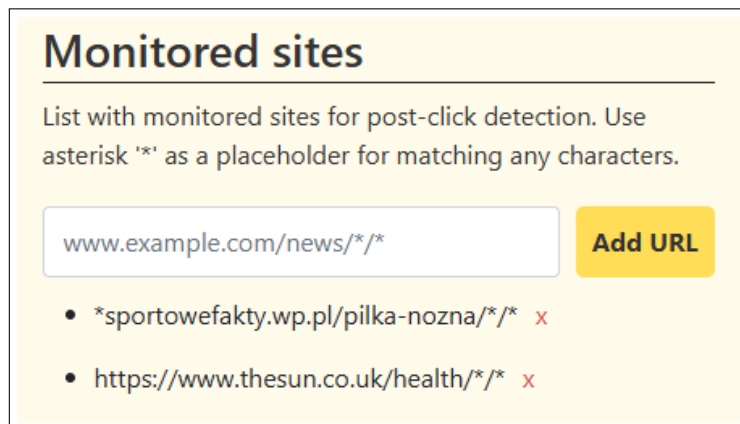


Figure 3.7: Monitored sites panel

#### 3.2.3. Detection pipeline

After detection is made:

- A badge is displayed on the extension icon.
- Prediction results are displayed in the popup.
- Popup pops up automatically, if the prediction is a clickbait.

Spoilers are generated only for articles with titles classified as clickbaits. However, user can use the toggle switch to turn on and off spoiler generation.

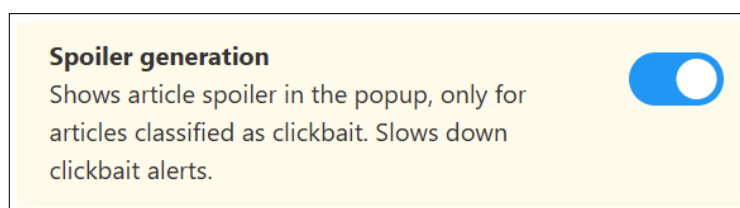


Figure 3.8: Spoiler generation toggle

### 3.3. Pre-click detection

In the pre-click mode, detection is run before a user accesses an article.

### 3.3.1. Configurations

There are two configurations available. One for Google Search, and one for The Sun news portal. It can be turned on using toggle switches in the options panel.

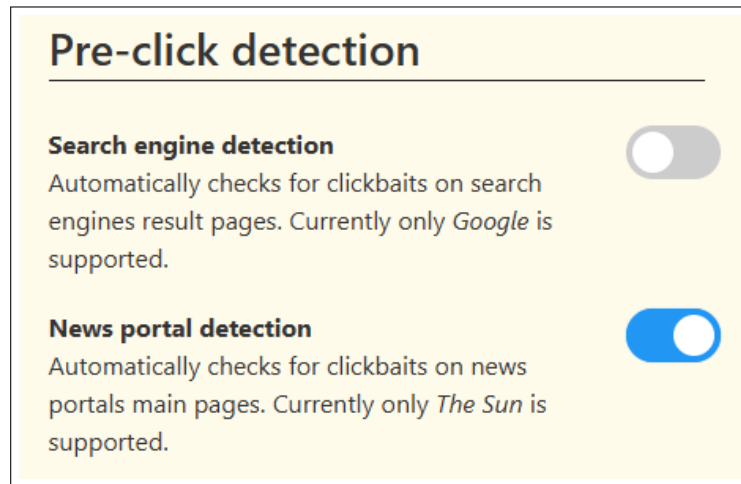


Figure 3.9: Pre-click settings panel