



Cairo University
Faculty of Computers and Artificial intelligence
Department of Computer Sciences



Course Pilot

Supervised by
Dr. Hanaa Bayoumi
TA. Mohamed Atta

Implemented by

20190041	Ahmed Tarek Fawzy Ibrahim Moharm
20190562	Mo'men Hatem Mohamed Ali
20190019	Ahmed Badr Shaban
20190050	Ahmed Essam-Eldin Abdelfattah
20190632	Youssef Khaled Abd-ElShafi

Graduation Project
Academic Year 2022-2023
Final Year Documentation

Table of Content

Chapter 1: Introduction	6
1.1 Motivation	6
1.2 Problem definition	7
1.3 Project Objective (suggested solution)	7
1.4 Gantt chart of project time plan	9
1.4.1 Project Analysis	9
1.4.2 Project Implementation	10
1.5 Project development methodology	11
1.6 The used tools in the project (SW and HW)	12
1.7 Report Organization (summary of the rest of the report)	13
Chapter 2: Related Work	14
2.1 The closest examples	14
2.2 Main Differences	15
Chapter 3: System analysis	17
3.1 Functional requirements	17
3.2 Non-functional requirements	17
3.3 Use Case Diagram	18
Chapter 4: System Design	19
4.1 System Component Diagram	19
4.2 System Class Diagram	20
4.3 Sequence Diagrams	21
4.4 Project ERD	22
4.5 System Architecture	23
4.6 System GUI Design	24
Chapter 5: Implementation and testing	29
5.1 Challenges	29
5.2 Solution	30

5.3 Implementation	31
5.3.1 Data preparation	31
5.3.2 Arabic Dialect detection.....	32
5.3.3 Cleaning phase in Arabic language.....	33
5.3.4 Cleaning phase in English language	34
5.4 Algorithms.....	35
5.5 Summarization	51
5.6 deployment for each model.....	52
5.7 APIs Implementation	54
Reference.....	57

Table of Figures

FIGURE 1 GANTT CHART PART 1	9
FIGURE 2 GANTT CHART PART 2.....	10
FIGURE 3: USE CASE	18
FIGURE 4: SYSTEM COMPONENT	19
FIGURE 5: CLASS DIAGRAM	20
FIGURE 6: SEQUENCE DIAGRAM FOR WRITE COMMENT	21
FIGURE 7 : SEQUENCE DIAGRAM FOR MAKE SUMMARY.....	21
FIGURE 8 : SEQUENCE FOR PROFESSOR SHOW REVIEWS	22
FIGURE 9: ERD	22
FIGURE 10: SYSTEM MONOLITHIC ARCHITECTURE.....	23
FIGURE 11: SIGN UP GUI	24
FIGURE 12: LOG IN GUI	24
FIGURE 13: HOME PAGE GUI.....	25
FIGURE 14: TOP RATED GUI.....	25
FIGURE 15: USER PROFILE GUI.....	26
FIGURE 16: CREATE COURSE GUI	26
FIGURE 17: UPDATE MATERIAL IN COURSE GUI	27
FIGURE 18: COURSE DETAILS PAGE GUI	27
FIGURE 19: WRITE REVIEW	28
FIGURE 20 UNBALANCED BRAD DATASET	29
FIGURE 21 UNBALANCED AMAZON DATASET	29
FIGURE 22 BALANCED DATASET.	30
FIGURE 23 TREE STRUCTURE MODEL.....	30
FIGURE 24 CODE FOR TRANSFORM LABELS TO OBJECTIVE AND SUBJECTIVE.	31
FIGURE 25 CLASSES DISTRIBUTION FOR POSITIVE AND NEGATIVE MODEL.....	31
FIGURE 26 CODE FOR ARABIC DIALECT DETECTION	32

FIGURE 27 ARABIC DIALECT PERCENTAGE.....	32
FIGURE 28 ARABIC CLEANING AND PREPROCESSING FUNCTION	33
FIGURE 29 ENGLISH CLEANING AND PREPROCESSING FUNCTION	34
FIGURE 30 SVM AND VECTORIZATION CODE	35
FIGURE 31 FITTING DATA SET.....	35
FIGURE 32 PREDICT CODE FOR SVM.....	35
FIGURE 33 CODE FOR KNN MODEL AND PIPELINE	36
FIGURE 34 CODE FOR LOGISTIC REGRESSION MODEL AND PIPELINE	37
FIGURE 35 CODE FOR DECISION TREE MODEL AND PIPELINE	37
FIGURE 36 CODE FOR NAIVE MODEL AND PIPELINE.....	38
FIGURE 37 CODE FOR LSTM MODEL USING TENSORFLOW.	39
FIGURE 38 CODE FOR CNN MODEL USING TENSORFLOW.	40
FIGURE 39 CODE FOR CNN-LSTM MODEL USING PYTORCH.....	43
FIGURE 40 TRANSFORMER PREDICT FUNCTION.....	44
FIGURE 41 DATA PREPARATION FOR TRANSFORMER AND MODEL CREATION FOR ENGLISH.....	45
FIGURE 42 TRAINER OBJECT AND TRAINING PHASE AND EVALUATION	46
FIGURE 43 DATASET CLASS TO HANDLE DATASET AND ITEMS.	48
FIGURE 44 DATA PREPARATION FOR TRANSFORMER AND MODEL CREATION FOR ARABIC.	50
FIGURE 45 TRAINER OBJECT AND TRAINING PHASE AND EVALUATION.	50
FIGURE 46 METRICS OF TRANSFORMER ARABIC MODEL	51
FIGURE 47 TXT LENGTH HISTOGRAM	52
FIGURE 48 LENGTH OF SUMMARY HISTOGRAM	52
FIGURE 49 LANGUAGE DETECTION.....	53
FIGURE 50 ENGLISH SPELLING CORRECTION	53
FIGURE 51 ARABIC SPELLING CORRECTION	53
FIGURE 52 SENTIMENT ANALYSIS API USING FLASK.....	55

Table of Lists

1: TABLE OF TOOLS THAT USED.	8
TABLE 2 CLASSES FOR EACH LABEL DISTRIBUTION.....	31
TABLE 3 CLASSES FOR EACH LABEL DISTRIBUTION.....	31
TABLE 4 METRICS OF SVM MODEL.	36
TABLE 5 METRICS OF KNN MODEL.	36
TABLE 6 METRICS OF LOGISTIC REGRESSION MODEL.	37
TABLE 7 METRICS OF DECISION TREE MODEL.	38
TABLE 8 METRICS OF NAIVE MODEL.....	38
TABLE 9 METRICS OF LSTM MODEL.	40
TABLE 10 METRICS OF CNN MODEL.	41
TABLE11 METRICS OF CNN-LSTM MODEL.....	43
TABLE 12 METRICS OF TRANSFORMER ENGLISH MODEL.	47
TABLE 13:SUMMARY TESTING RESULTS	52

Table of Abbreviations

MSA	Middle East and South Asia	OM	Oman
KW	Kuwait	YE	Yemen
BH	Bahrain	DZ	Algeria
EG	Egypt	MA	Morocco
SA	Saudi Arabia	PL	Poland
SD	Sudan	JQ	Jordan
AE	United Arab Emirates	TN	Tunisia
IQ	Iraq	SY	Syria
LY	Libya	QA	Qatar
LB	Lebanon	LSTM	Long short-term memory
CNN	Convolutional Neural Networks		

Chapter 1: Introduction

1.1 Motivation

Previously, if a student wanted to complain about a particular course or express his opinion, he would write a survey if the institute offering this course cared about students' opinions about the learning system.

When it comes to exams or you have an assignment and there is a lot of material to study, it can be difficult to study all that material before the deadline. Teachers may need to create a survey and ask the student to fill it out to find out what they think about the course. They may choose a random number of reviews to consider rather than reading them all to represent average student satisfaction.

Any student before joining the course would like to know the opinions of other students and their previous experiences with the course to see whether it is suitable for him or not. but it is not easy even to collect needed information from several students.

This service provides all statistics and analyzes based on the opinions of previous users of the service for both the user (to see if it is more convenient for him) and the instructor (to improve the service).

Main technique and application:

in Machine learning algorithms, we use:

- Transformer model: neural network that learns context and therefore meaning by tracing relationships in sequential data such as the words in this sentence. Transformers eliminate the need to train neural networks with large, labeled data sets that were costly and time consuming by mathematically finding patterns between elements.
- Pretrained model from hugging-face for text extractive summarization. Extractive summarization identifies a subset of sentences from the text to form a summary that frames the task as a binary classification problem for each sentence in the text to define it in the abstract. It will rate each sentence in the document against all others, based on how well each line is explained.

1.2 Problem definition

Nowadays, there are a lot of courses in various fields that the university can offer which has many students. But only a few teachers can control and monitor them, so there is a great need for feedback from students enrolled in these courses.

Perhaps the number of students is so large that it is almost impossible for the teaching assistant to consider everyone's opinion on all courses in each department.

Reviews about courses also need analysis from the instructors so they can see where the problem is coming from so if they fix it, this will prevent more problems in the future.

Online courses are becoming increasingly popular, and many students choose this style of learning. With the increase in the number of online courses available, it has become imperative to have a way to determine the quality of these courses. One way to do this is to collect and analyze student evaluations. However, manually analyzing many reviews can be time consuming and inefficient.

Problem for institution: The problem which always existed is that there are a lot of government institutions with a certain number of managers who try to manage and lead this institution, but because there are a lot of aspects which need to be observed, they always need to evaluate their work to put them in the right track.

Problem for students: Students usually spend time looking for courses that are suitable for them, but they find many similar courses that offer the same content, but in different ways. Here, students search for the opinions of others and the evaluations of those who have studied this course before, until they find the best or suitable course for them.

A student may find it difficult to study a course if there is a lot of course material to study in a short period of time because there is an exam, exam or assignment nearing their deadline.

1.3 Project Objective (suggested solution)

The project will develop a website that can collect reviews from students who have taken online courses. The website will then use Natural Language

Processing (NLP) techniques to analyze the sentiment of the reviews and create an overall sentiment score. This sentimental score will help other students interested in taking the course make an informed decision.

Furthermore, the site will generate a summary of the course materials to help students understand the course content. The summary will provide an overview of the course, including key concepts, topics, and learning objectives. This summary will be particularly useful to students who are interested in the course but are unsure whether it is suitable for their needs.

Here comes our application that takes feedback from students and analyzes it, performs some probability calculations for them, and then sends it to course directors to note if they are doing a good job or if something is wrong with a particular dispute. In addition, when the principal solves a problem, he can post a blog to all the users who have been concerned about the problem, assuring the students that their feedback is being taken care of to improve the learning process.

This service allows users to view all reviews related to a particular course, as well as view previous analyses based on the complaints of others, to enable him to know if he is the only one who sees that there is a problem like his. For example, in Cairo University courses, based on previous reviews of the system {30% negative, 50% positive, 20% neutral}.

generate a summary of all course material in pdf format (book and lecture slides) for users so that they can study course material in less time.

Implementing this application needs tools like:

- PHP Laravel for back-end	- Angular Framework for front-end
- HTML, CSS for front-end	- Python Programming language
- JavaScript Programming language	- TypeScript
- PyTorch Framework	- TensorFlow Framework
- NLTK library	- TextBlob library
- Scikit-Learn library	- Regular expression ReGx
- Machine learning algorithms	- Deep learning architecture
- Transformers and pre-trained models	- Flask Framework for ML APIs
- Postman for testing	- Word2Vector and Gloves

1: Table of tools that used.

1.4 Gantt chart of project time plan

1.4.1 Project Analysis

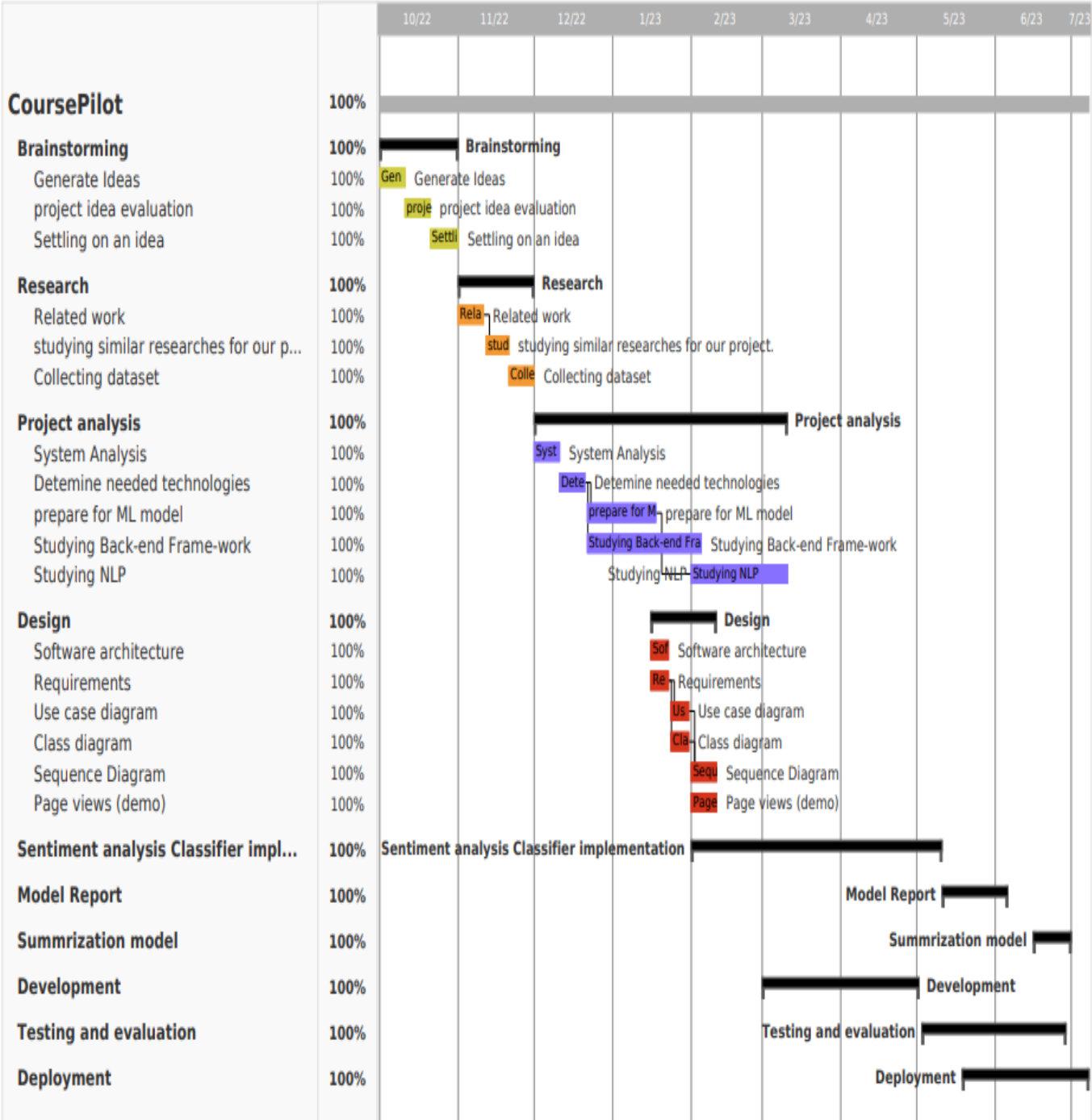


Figure 1 Gantt Chart part 1

1.4.2 Project Implementation

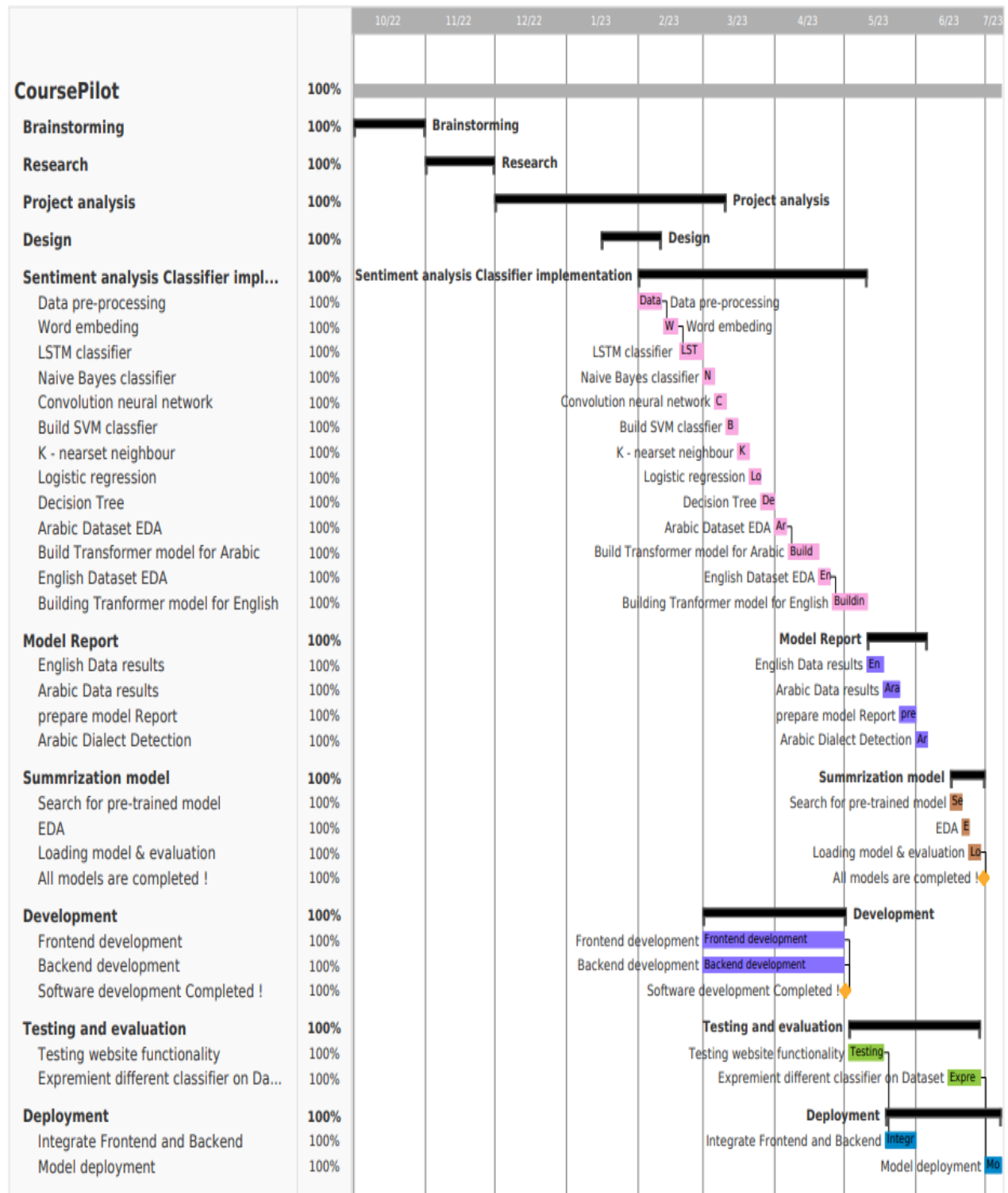


Figure 2 Gantt chart part 2

1.5 Project development methodology

In the software development life cycle, we use the waterfall model which uses the logical sequence of SDLC steps for a project. Waterfall is an appropriate method because it aims to achieve this goal. It enables early system design and specification changes to be made easily and clearly sets milestones and deadlines. The entire software development process is divided into separate phases. Usually, the result of one stage serves as the input for the next stage sequentially. Stages:

Requirements Gathering: The first phase in the waterfall methodology is requirements gathering. In this phase, we determined user needs to gather all the functional and non-functional requirements for the project. The requirements should include the main functionality of the project, which are sentiment analysis of students' reviews and the summary of the course materials.

Analysis and Design: In this phase, we take the requirements collected in the first phase and create a detailed analysis and design plan. This plan includes all technical specifications and design documents for the website. We also identified potential risks or issues that could arise during the development process.

Implementation: The implementation phase involves the actual development and coding of the website. We followed the design documents and coding standards to build the project. The sentiment analysis classifier, material summarization functionalities and website backend and frontend are implemented at this stage.

Testing: Once the implementation phase is completed, we conducted thorough testing of the website to ensure that it meets all the requirements and functions as intended. Sentiment analysis classifier and summarization model are tested extensively to ensure their accuracy and reliability. website is tested using automation testing tools and quality assurance techniques.

Maintenance: Once the website is deployed, it will require ongoing maintenance and updates to ensure that it continues to function correctly. We develop a maintenance schedule and plan to ensure that the website remains up-to-date and relevant. Also issues or bugs that arise should be addressed in a timely manner to maintain the functionality of the website.

1.6 The used tools in the project (SW and HW)

Front-end:

- Angular framework based on TypeScript programming language
- VSCode development environment
- Pencil for Designing GUI Wireframes
- Photopea.com for Designing the pages
- ColorHunt website for help determining Color contrast
- Github for Version Control with backend
- Postman for testing API's

Back-end:

- Php
- Laravel
- MySQL as the database

Machine learning:

- Python programming language: Numpy, Pandas, Sklearn, Matplotlib for visualization.
- Book review Arabic Dataset (BRAD): to train sentiment analysis classifier based on Arabic reviews with rating from 1 to 5 values.
- Amazon book reviews: English dataset contain book reviews with rating from 1 to 5 values.
- ArabicBert Dialect identification: pretrained model from huggingface that identifies dialect of Arabic language of review.
- extractive summarization model: pretrained model from "huggingFace" that summarizes book content by 15% of book size.
- Transformer. - PyTorch and Tensorflow
- Kaggle environment.
- Flask: model deployment to build API that is used by Back-end.

1.7 Report Organization (summary of the rest of the report)

The rest of the document discusses the different phases that describe and illustrate the characteristics of the project.

Chapter Two: Related Works, here we compare other learning websites that offer courses to our project, determine user problems with current related websites and similar solutions to that problem.

Chapter Three: System Analysis, including functional requirement of the system according to what we found in related works, non-functional requirement that indicate the performance of the system, use case diagram that shows stakeholders and also use case scenarios.

Chapter Four: System Design, include all diagrams that describe relations between classes and component in system and how they stored/retrieved in database.

- Component Diagram.
- Class Diagram.
- Sequence Diagrams.
- Entity relation Diagram.
- GUI design.

Chapter Five: Implementation and Testing, include reports about machine learning models and Back-end and testing scenarios results.

Summary

The project aims to develop a courses website that can analyze the sentiment of student reviews on the courses they have enrolled in. The website will collect reviews from students and use NLP techniques to determine the overall sentiment of the reviews. Additionally, the website will provide a summary of the course material to assist students in understanding the content of the course.

To achieve the goals of the project, the site will use NLP techniques such as sentiment analysis to analyze reviews. The website will also use machine learning algorithms to generate a summary of the course materials. Machine learning algorithms will be trained on a large dataset of course materials to ensure accurate and relevant summaries.

Chapter 2: Related Work

2.1 The closest examples

A Novel Approach for Course Recommendation System using Sentiment Analysis and Summarization Techniques.

This paper makes course recommendation that combines sentiment analysis and summarization technique to sentiment analysis technique to analysis student feedback and generate concise and informative summaries.

This system uses TextBlob a python library for NLP for sentiment analysis and TextRank algorithm for summarization.

The algorithm extracts keywords and phrases from the student feedback and ranks them based on their importance, which is determined by their frequency and position in the text.

The system then recommends courses to students based on their preferences and the sentiment of the feedback.

Sentiment Analysis and Summarization of Student Feedback for Course Improvement.

This paper that uses sentiment analysis and summarization technique to analysis Feedback and provide techniques to analysis students feedback and provide suggestion for course improvement.

The system uses a modified Naive Bayes classifier for sentiment analysis and a sentence extraction approach for summarization.

The algorithm first preprocesses the student feedback by removing stop words and punctuation, and then applies sentiment analysis to classify each sentence as positive, negative, or neutral. The system then extracts the most informative sentences from the feedback and generates a summary that highlights the key issues and areas for improvement.

Automated Course Evaluation Based on Semantic Analysis and Summarization Techniques.

This paper is a system that uses sentiment analysis and summarization techniques to automate the process of course evaluation.

The system uses the SentiWordNet lexicon for sentiment analysis and the TextRank algorithm for summarization.

The algorithm first applies sentiment analysis to classify the feedback as positive, negative, or neutral, and then uses TextRank to extract the most informative sentences from the feedback. The system then generates a concise and informative report of the course evaluation that includes a summary of the feedback, the sentiment of the feedback, and suggestions for improvement.

Course Evaluation Analysis using Sentiment Analysis, Summarization and Visualization Techniques.

This study proposes a system that uses sentiment analysis and visualization techniques to analyze student feedback and provide insights for course improvement.

The system uses the Vader sentiment analysis tool and a word cloud visualization to present the results.

Sentiment Analysis on Online Course Reviews: A Study on Coursera.

The authors use the Support Vector Machine (SVM) classifier for sentiment analysis on online course reviews. They focus on analyzing the sentiment of course reviews rather than summarization or recommendation.

"Sentiment Analysis on MOOCs Reviews: A Hybrid Approach".

The authors propose a hybrid approach for sentiment analysis and summarization of MOOC reviews that combines a convolutional neural network (CNN) for sentiment analysis and a graph-based summarization algorithm. They also evaluate the performance of their approach on a dataset of MOOC reviews.

2.2 Main Differences

From previous projects that related to student engagement, we notice many things like most of them use machine learning for sentiment analysis like SVM, modified naïve Bayes, TextBlob that depending on frequency of each word to make sentiment analysis.

And another student engagement that use SentiWordNet that works by assigning three sentiment scores to each sentence that depending on WordNet, a large lexical database of English words and their semantic relationships.

And CNN model depending on word embedding which transfer each word to dense vector representation.

Previous projects in summarization, they use TextRank which extract the most informative sentences from the feedback.

In our project we take feedbacks of students in course and make sentiment analysis on it to predict it positives or negative or neutral and make summarization of any content student want to summarize.

We use in sentiment analysis tools DistilBert transformer model in English text and train it with Amazon book review dataset, use Transformer architecture specifically the BERT model architecture in summarization and use Billsun dataset to train the model.

We treat also with multilingual like when user write Arabic review or English and Arabic because we Use MarBERT transformer model in sentiment analysis with Arabic texts and use Brad dataset to train model on Arabic texts.

Examples of Arabic dialects we treat with in our data set (Brad dataset)

MSA, KW, BH, EG, SA, SD, AE, IQ, OM, YE, DZ, MA, PL, JQ, TN, SY, LY, QA, LB

And we treat it with diacritical marks and make model learn from it and it make result become better.

Chapter 3: System analysis

3.1 Functional requirements

- **User authentication:** web application must allow users to login and reach and access their accounts securely.
- **Dealing with large data:** software must deal with large number of files that instructors uploaded.
- **Integration with other systems:** The software must be able to integrate with other systems or applications as we deal with YouTube subtitles, and videos.
- **Analysis reviews:** the system will analysis user's reviews and show for each course its own statistics.
- **Summarize materials:** the system will analysis and summarize material that user asked for.
- **Dealing with multiple languages:** system will analysis users reviews for English and Arabic languages.
- **Visualization analysis:** the system should visualize the statistics for courses.
- **User profiles:** system allow to user to show its information and update it.

3.2 Non-functional requirements

- **Usability:** The system should be easy to use and navigate, with clear instructions and an intuitive interface.
- **Performance:** The system should be responsive and able to handle many users and files without experiencing delays or crashes.
- **Reliability:** The system should be reliable and available 24/7, with minimal downtime or maintenance.
- **Scalability:** The system should be able to scale up or down to accommodate changes in user demand or file sizes and amounts.
- **Security:** The system should have appropriate security measures in place to protect user data and prevent unauthorized access or data breaches.
- **Compatibility:** The system should be compatible with a variety of devices and web browsers and should be accessible to users with disabilities.
- **Maintainability:** The system should be easy to maintain and update, with clear documentation and a modular design that allows for easy changes and upgrades.

3.3 Use Case Diagram

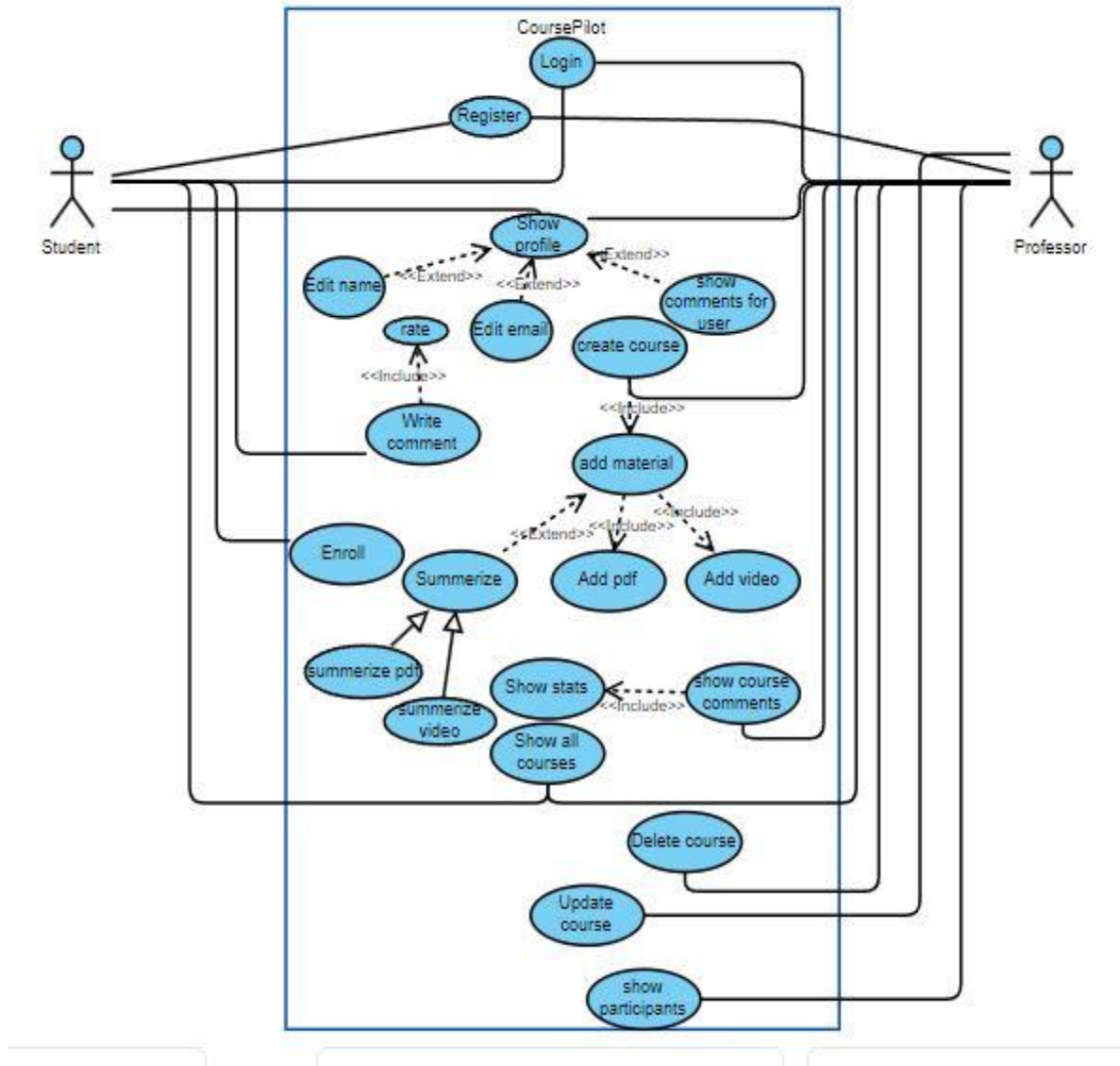


Figure 3: Use Case

Chapter 4: System Design

4.1 System Component Diagram

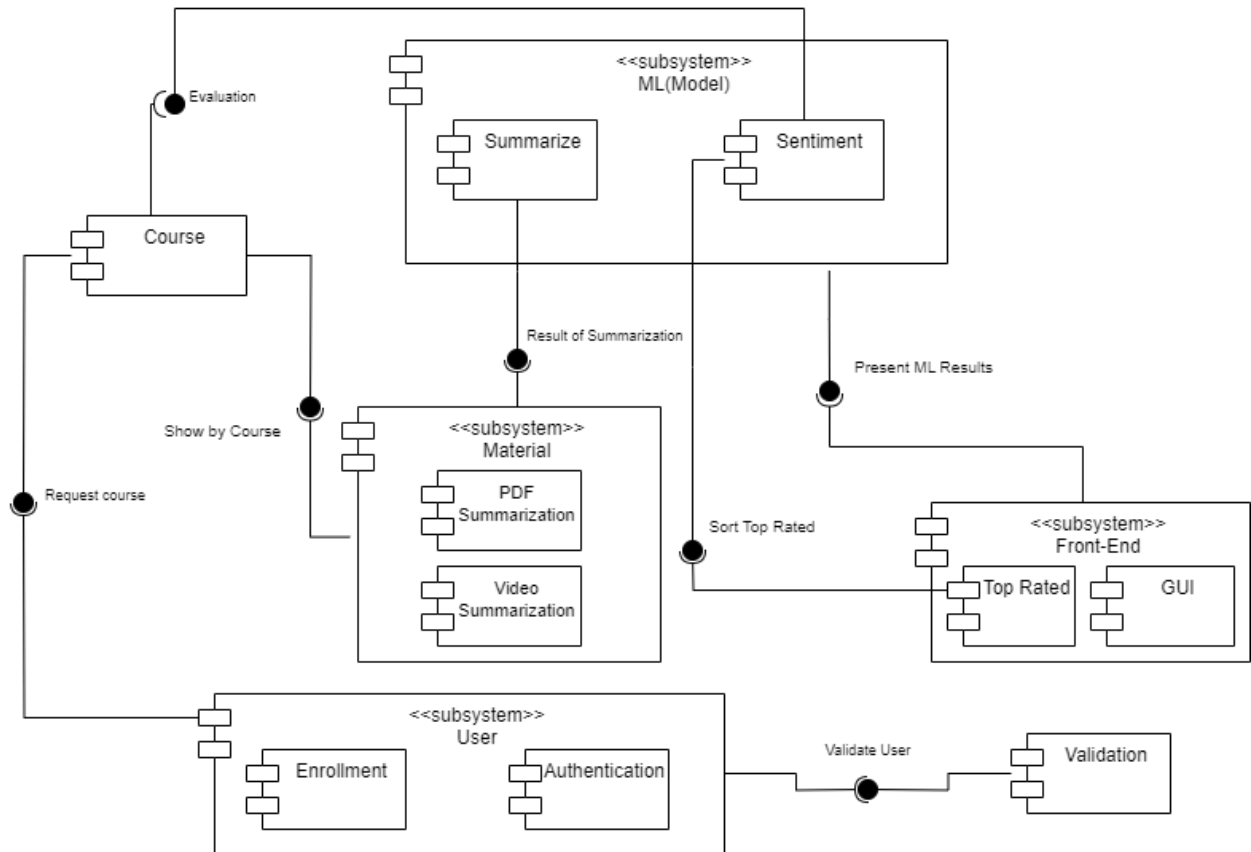


Figure 4: System Component

4.2 System Class Diagram

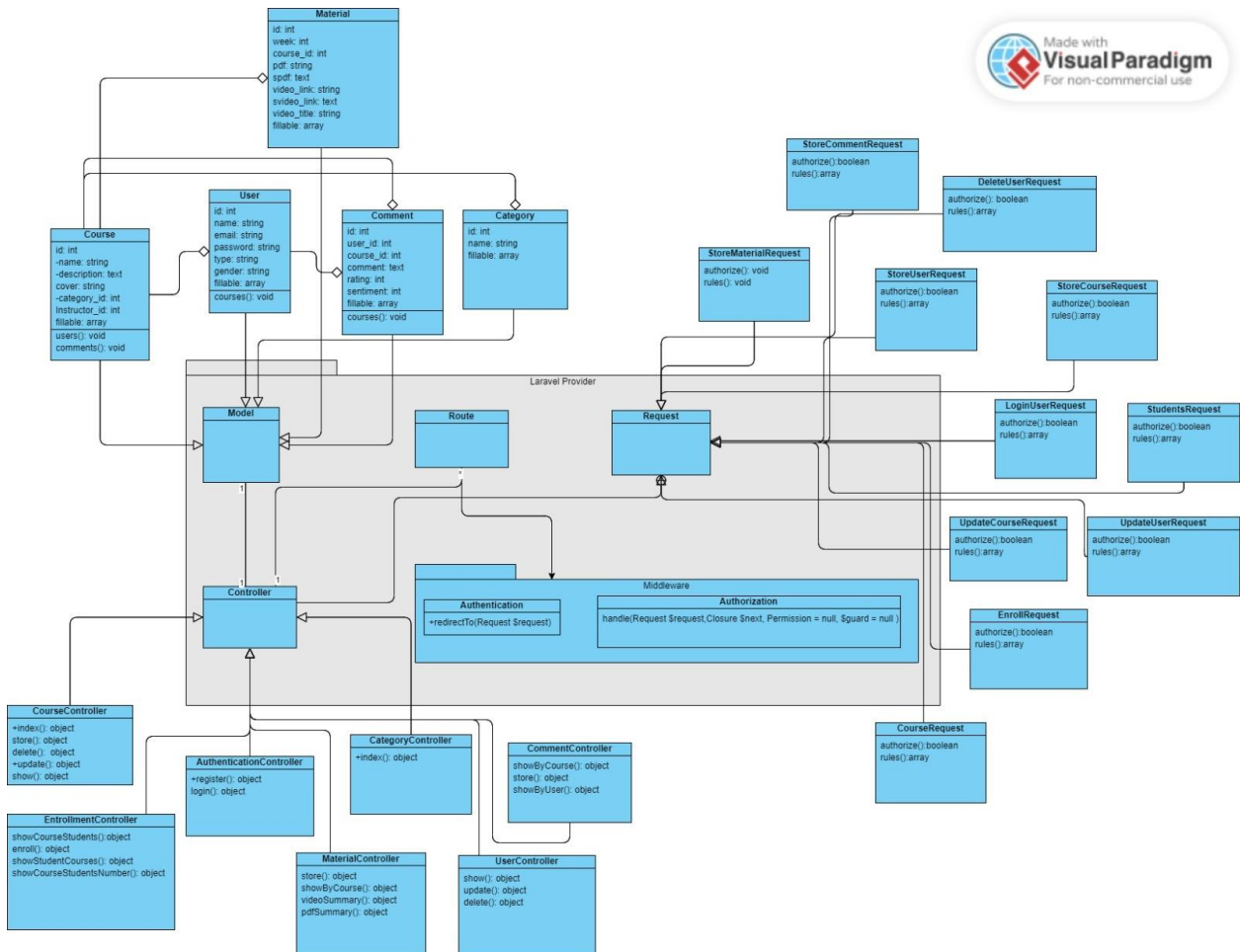


Figure 5: Class Diagram

4.3 Sequence Diagrams

Student Write Comment

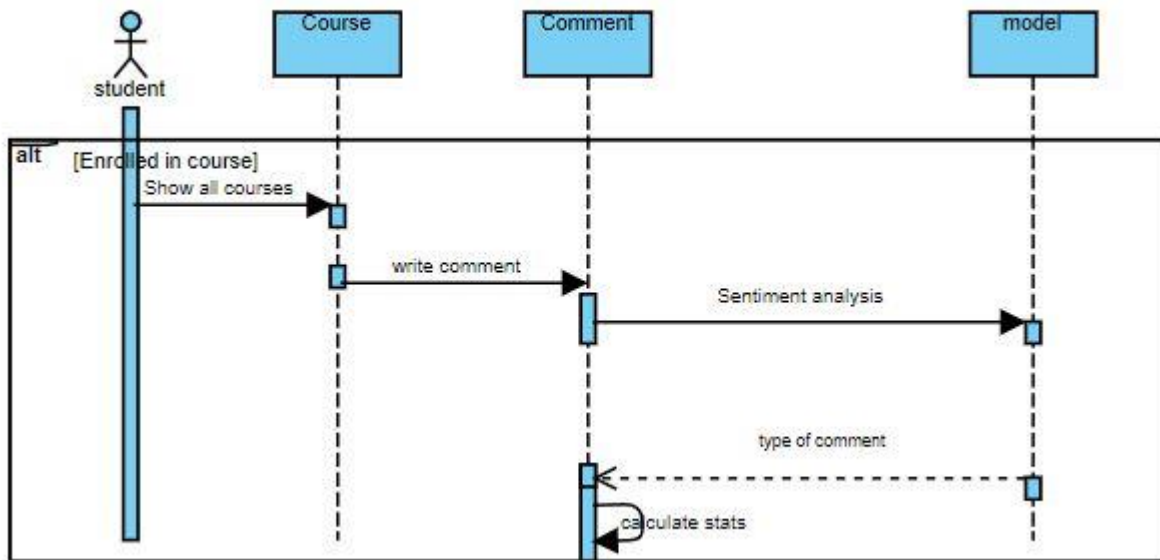


Figure 6: Sequence diagram for write comment

Student Make Summary

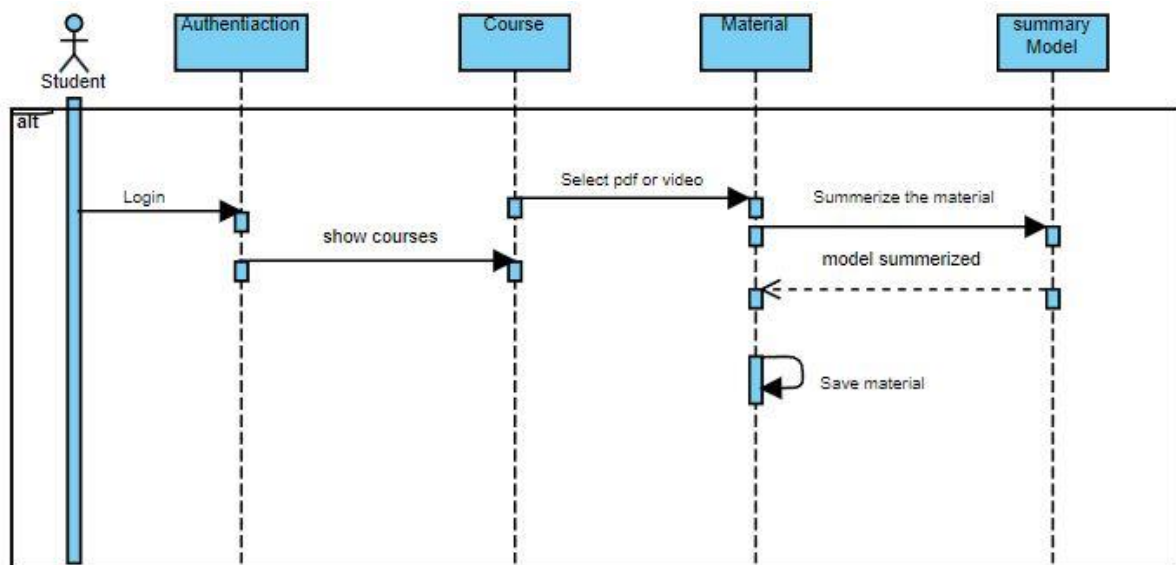


Figure 7 : Sequence diagram for make summary

Professor Show Reviews

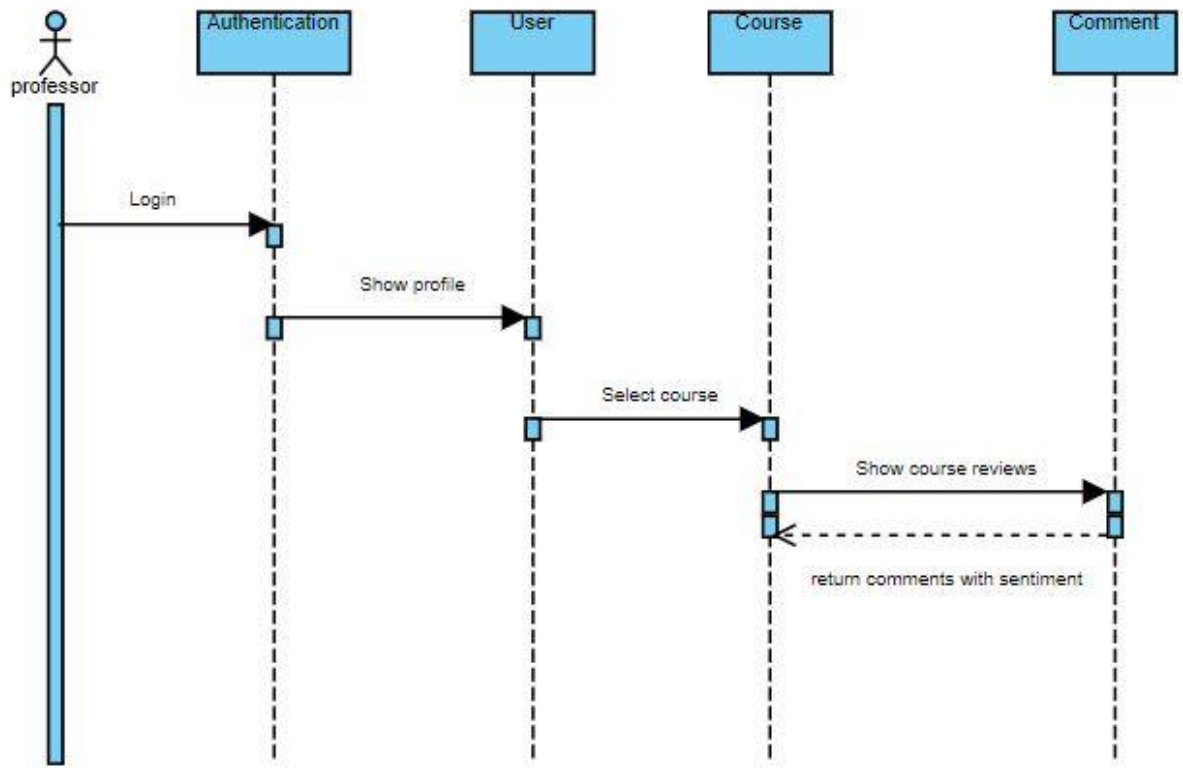


Figure 8 : Sequence for professor show reviews

4.4 Project ERD

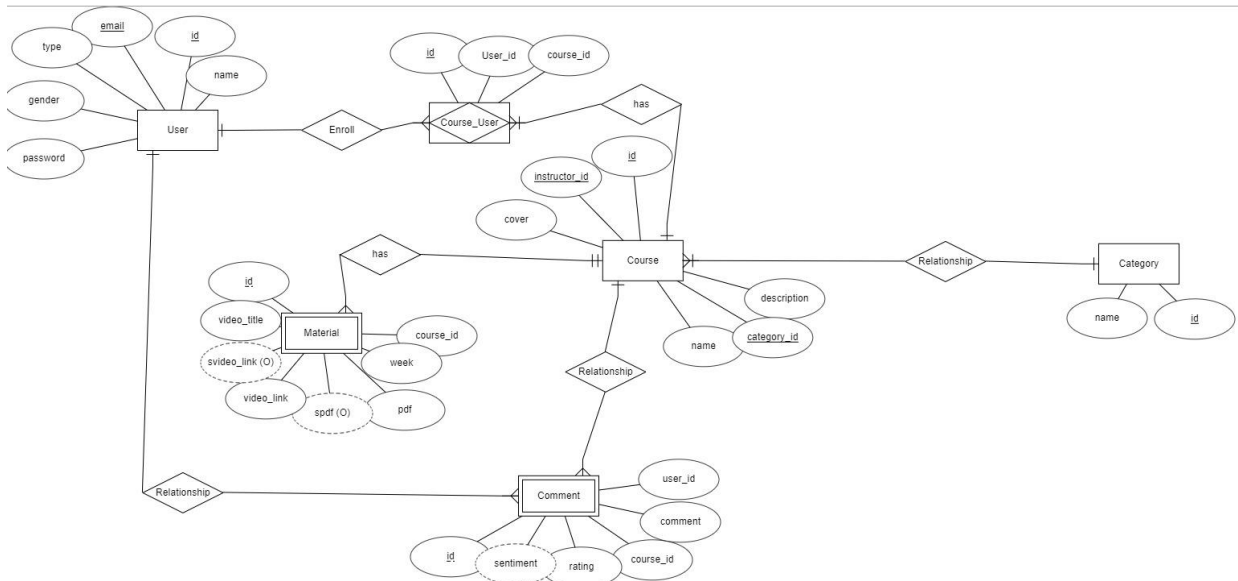


Figure 9: ERD

4.5 System Architecture

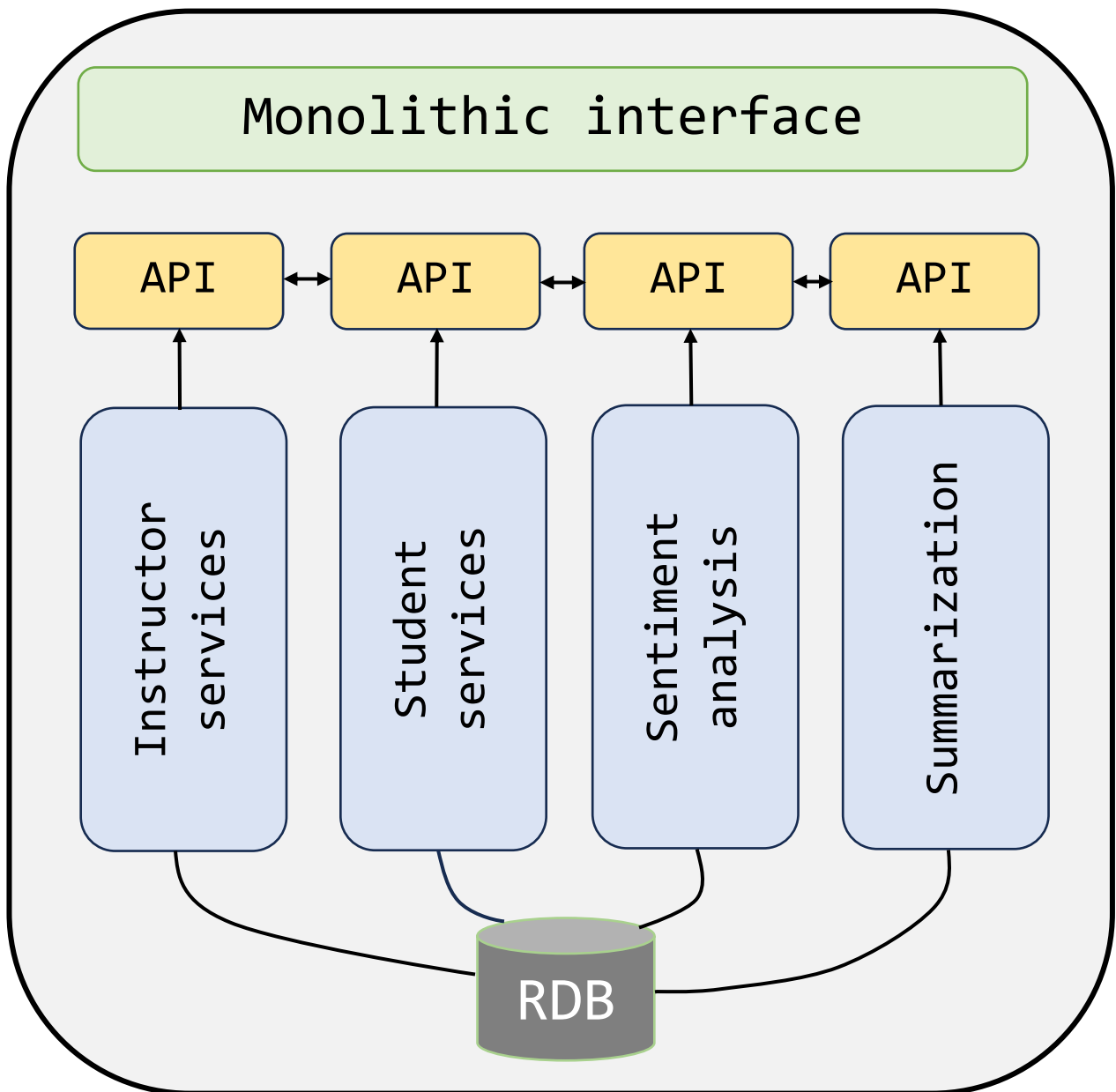
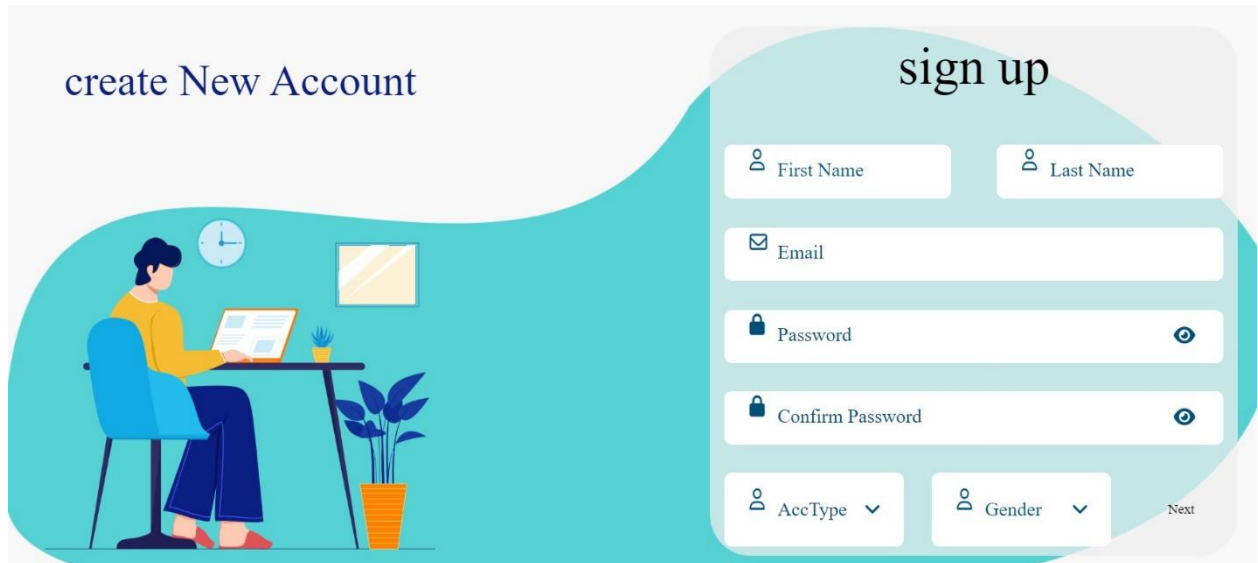


Figure 10: System Monolithic Architecture

4.6 System GUI Design

Sign Up



The Sign Up GUI features a light blue background with a stylized illustration of a person sitting at a desk with a laptop, a clock, and a potted plant. The text "create New Account" is displayed in a dark blue serif font. The sign up form is a light blue rounded rectangle with the title "sign up" in a dark blue serif font. It contains input fields for First Name, Last Name, Email, Password, and Confirm Password, each with a corresponding icon (person, envelope, or lock). There are also dropdown menus for AccType and Gender, and a "Next" button.

create New Account

sign up

First Name

Last Name

Email

Password

Confirm Password

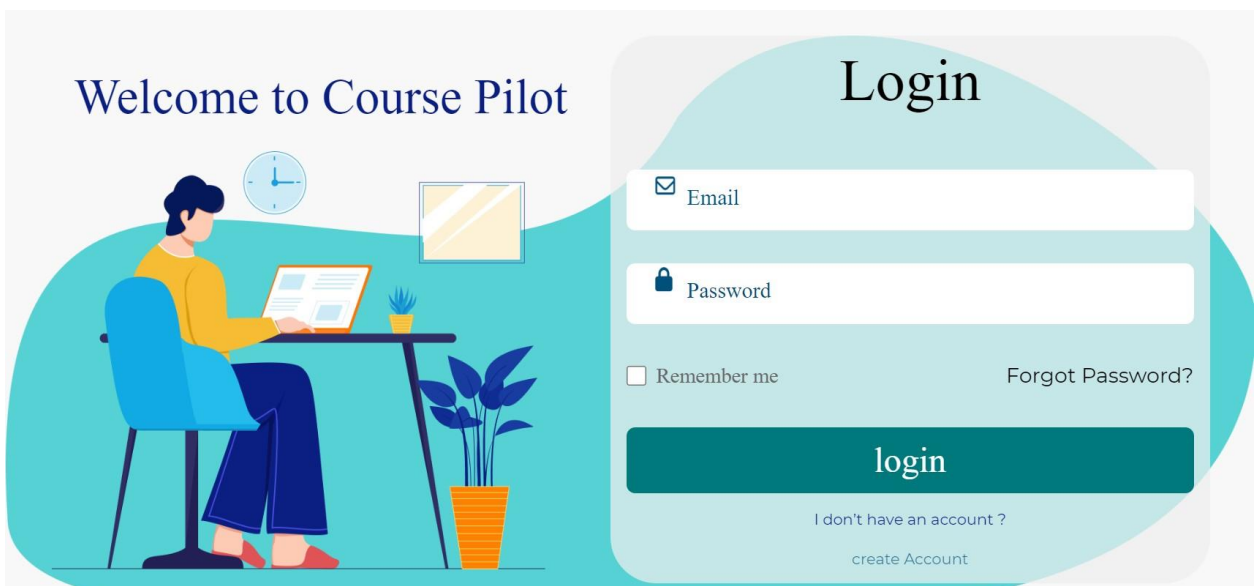
AccType

Gender

Next

Figure 11: Sign Up GUI

Log in



The Log in GUI features a light blue background with a stylized illustration of a person sitting at a desk with a laptop, a clock, and a potted plant. The text "Welcome to Course Pilot" is displayed in a dark blue serif font. The login form is a light blue rounded rectangle with the title "Login" in a dark blue serif font. It contains input fields for Email and Password, each with a corresponding icon (envelope or lock). There is a "Remember me" checkbox and a "Forgot Password?" link. A large dark blue "login" button is at the bottom. Below the button are links for "I don't have an account ?" and "create Account".

Welcome to Course Pilot

Login

Email

Password

Remember me

Forgot Password?

login

I don't have an account ?

create Account

Figure 12: Log in GUI

Home Page

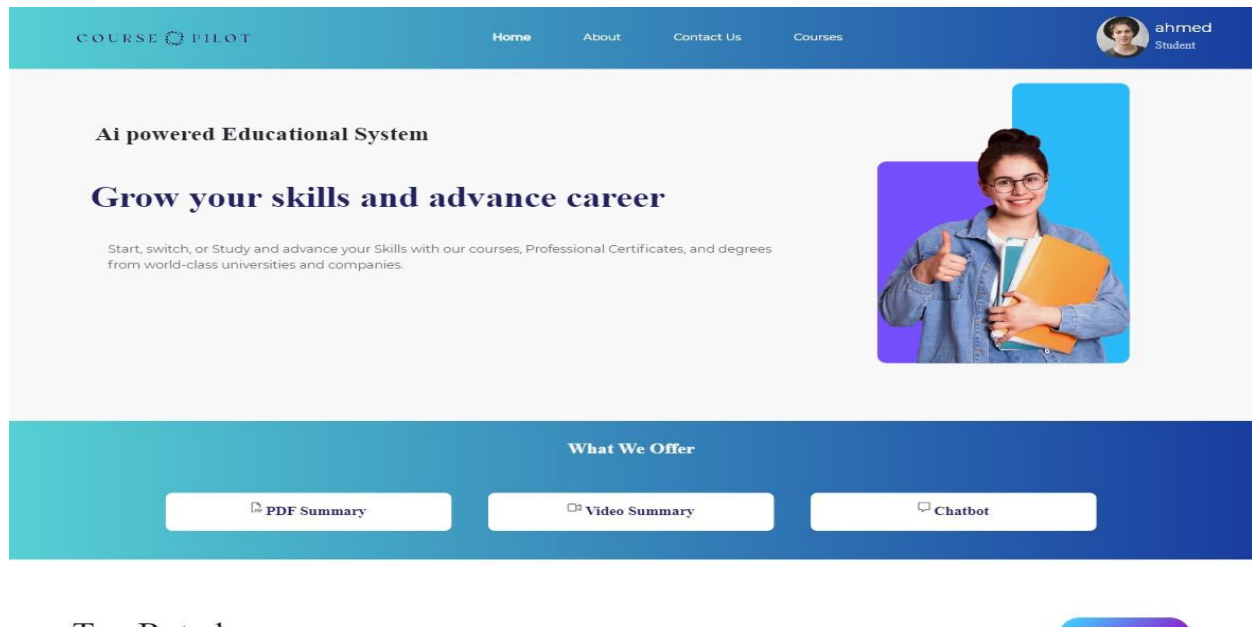


Figure 13: Home Page GUI

Top Rated (Based of Courses Reviews Analysis)

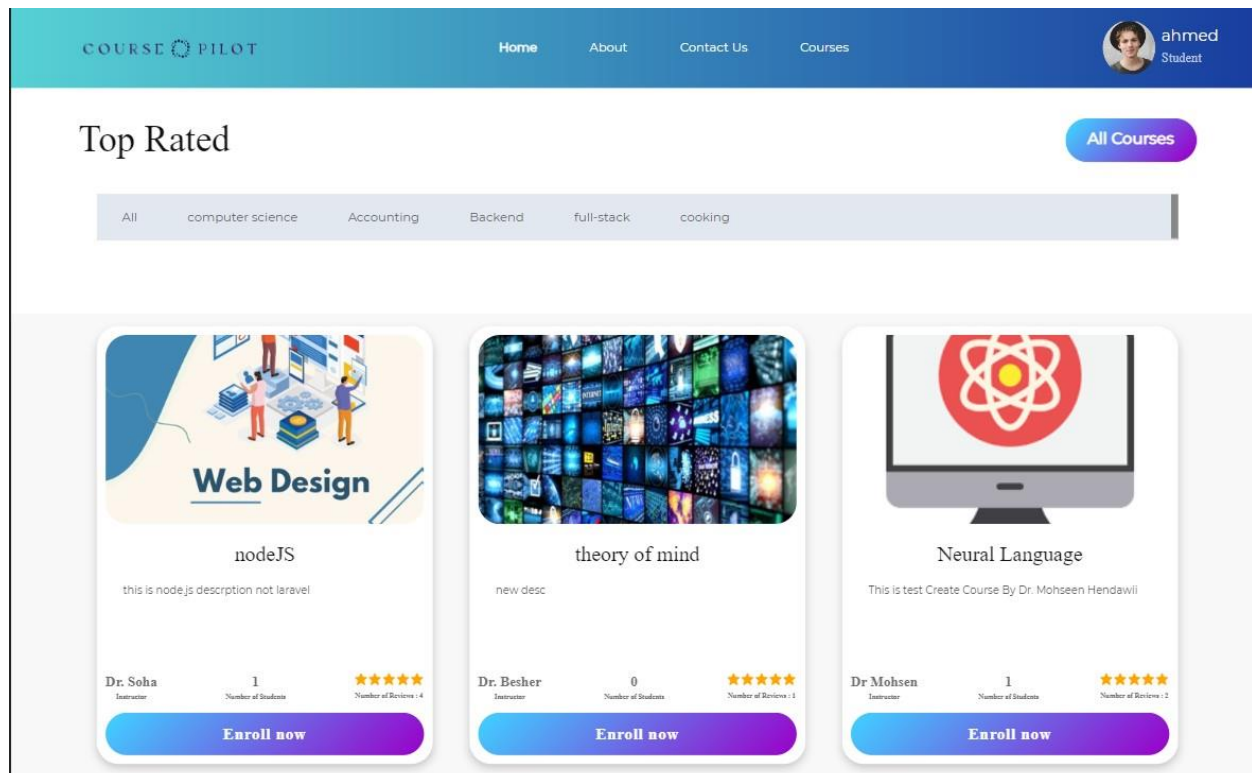


Figure 14: Top Rated GUI

User Profile (Instructor)

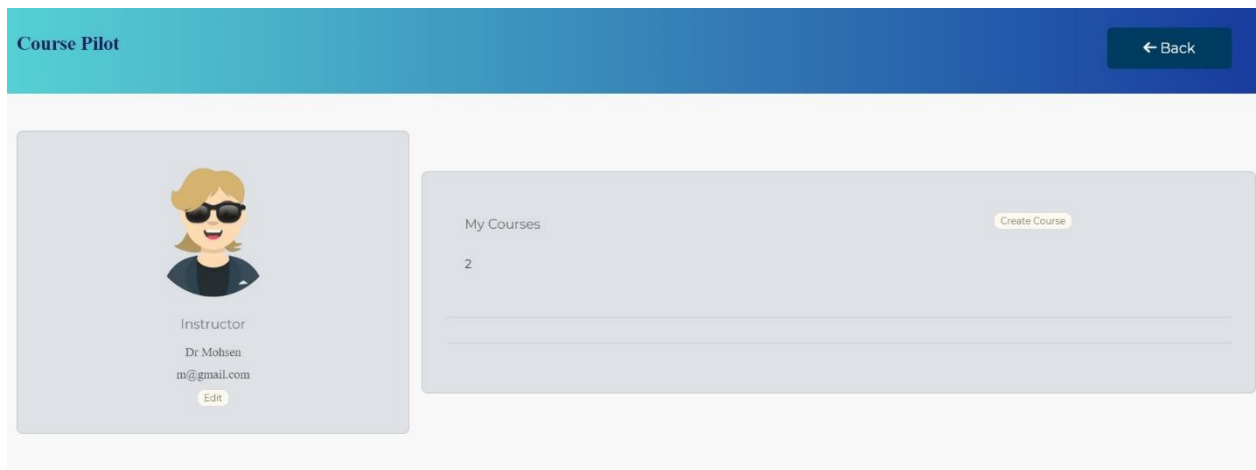


Figure 15: User Profile GUI

Create Course

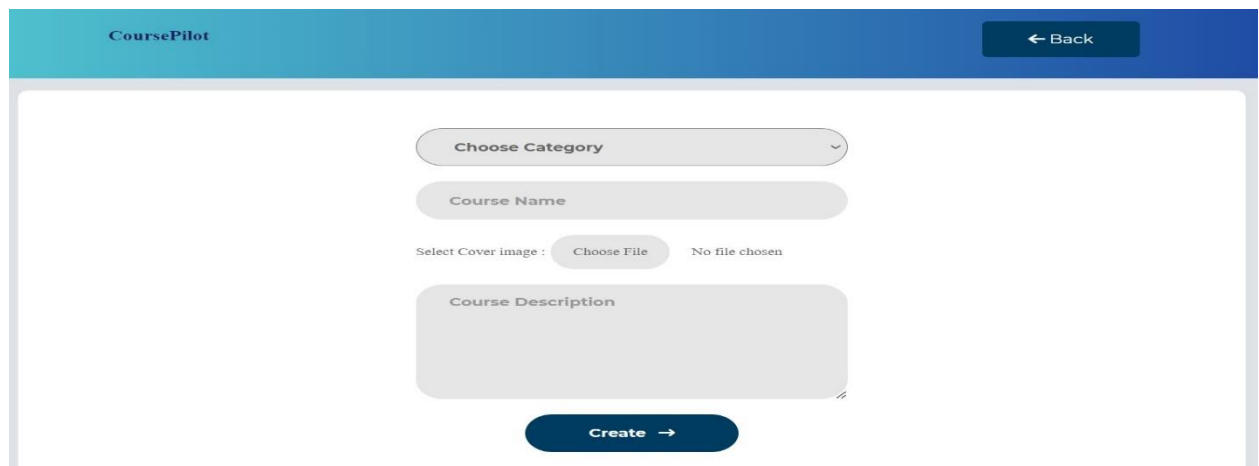
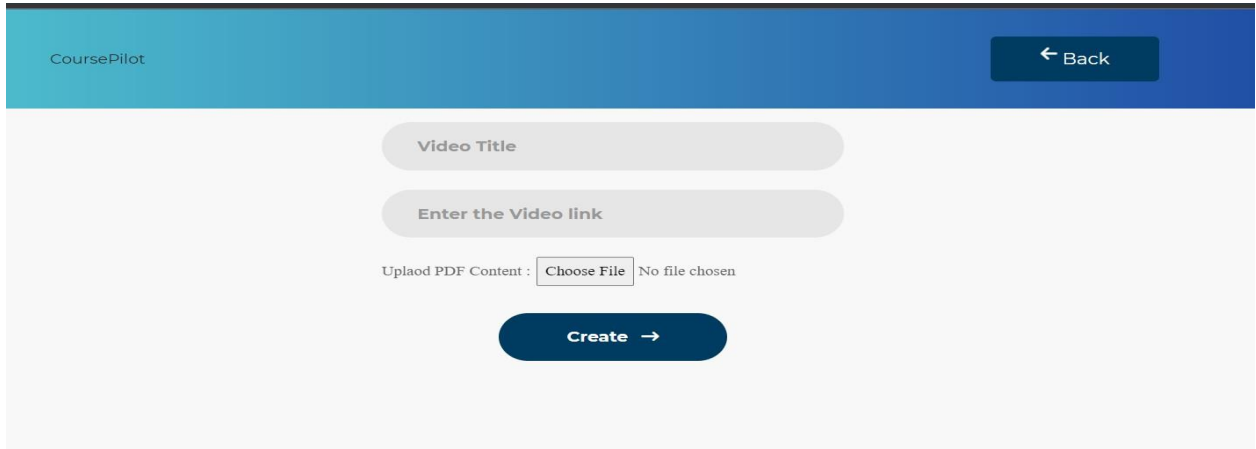


Figure 16: Create Course GUI

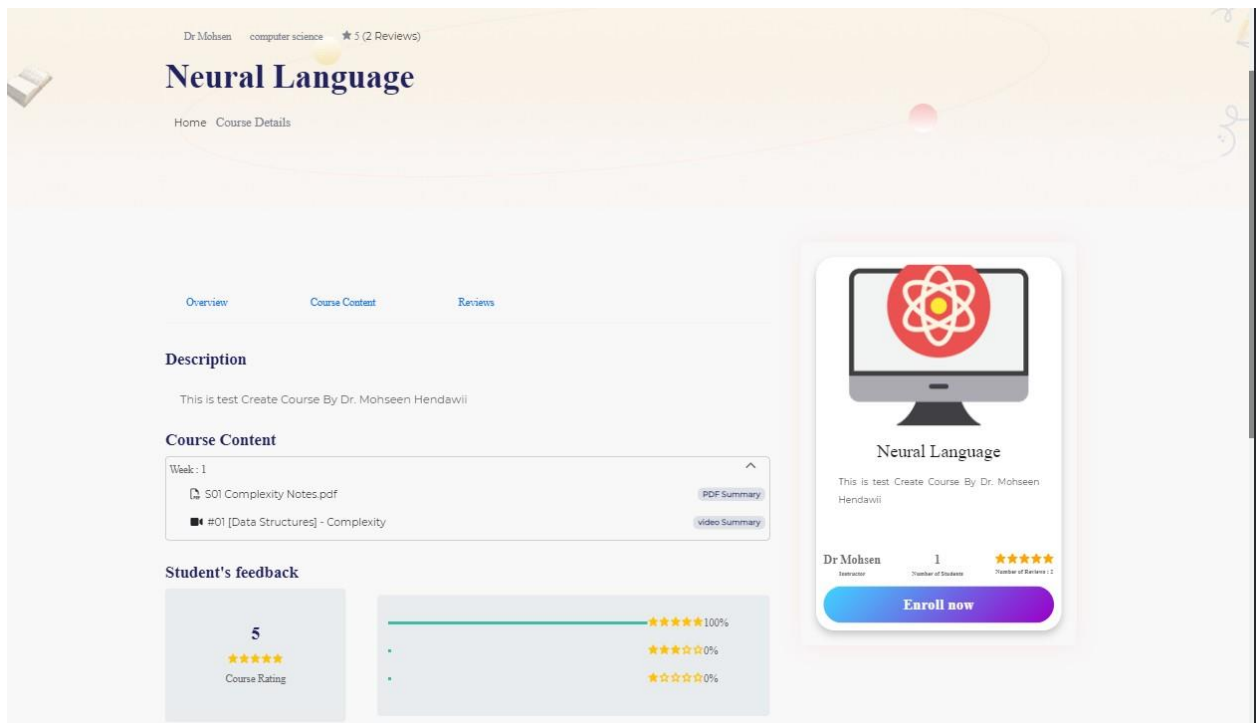
Upload Material in Course



The image shows a web interface for uploading course material. At the top, there is a blue header with the text "CoursePilot" on the left and a "Back" button with a left arrow on the right. Below the header, there are three input fields: "Video Title", "Enter the Video link", and "Upload PDF Content :". The "Upload PDF Content :" field has a "Choose File" button and the text "No file chosen". At the bottom, there is a large blue button labeled "Create" with a right arrow.

Figure 17: Update Material in Course GUI

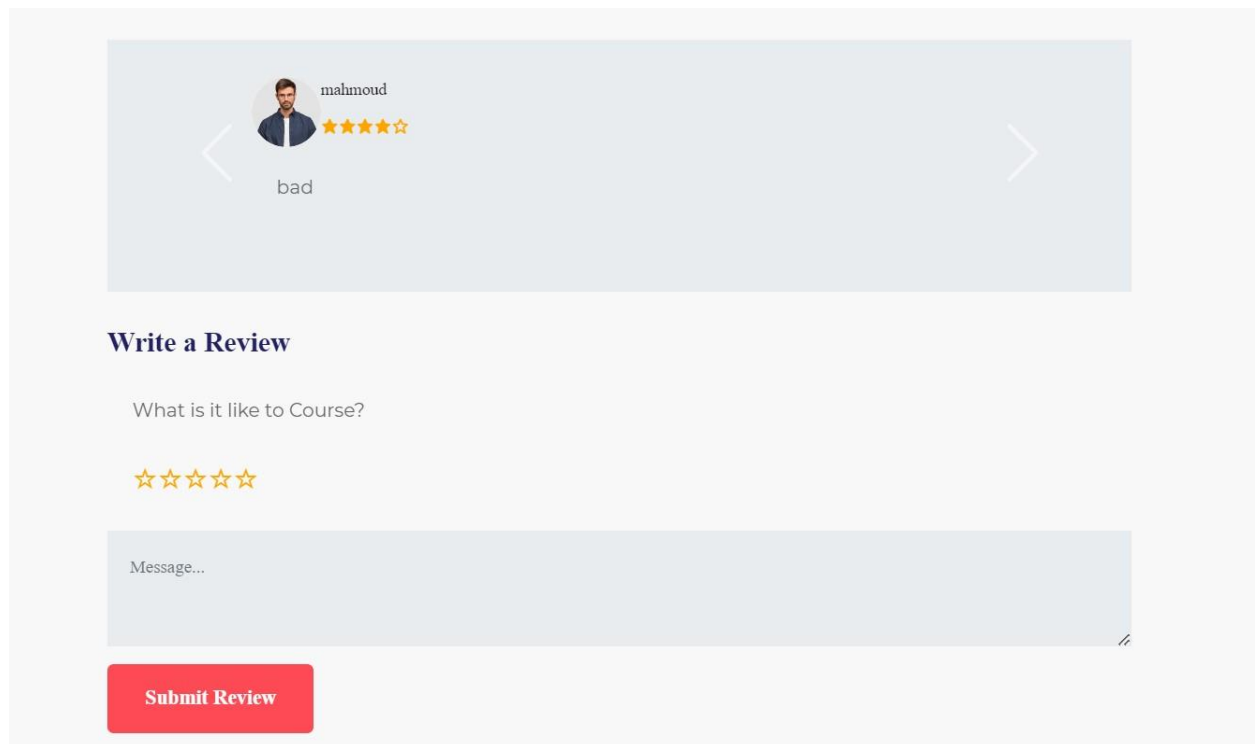
Course Details Page



The image shows a course details page for "Neural Language". At the top, there is a header with the course name "Neural Language" and the instructor "Dr. Mohsen" (computer science) with a star rating of 5 (2 Reviews). Below the header, there is a navigation bar with "Overview", "Course Content", and "Reviews". The "Overview" section contains a description: "This is test Create Course By Dr. Mohseen Hendawli". The "Course Content" section shows a list of items: "Week: 1", "S01 Complexity Notes.pdf", and "#01 [Data Structures] - Complexity". The "Student's feedback" section shows a course rating of 5 stars and a progress bar for the course content. On the right side, there is a course card with the course name "Neural Language", the instructor "Dr. Mohsen", and a button "Enroll now".

Figure 18: Course Details Page GUI

Write Review



The image shows a 'Write Review' form. At the top, there is a header bar with a user profile for 'mahmoud' (a man with a beard and glasses) and a 5-star rating. Below this, the word 'bad' is written. The main section is titled 'Write a Review' and contains a text input field with the placeholder 'What is it like to Course?'. Below the text field is a 5-star rating system. At the bottom, there is a text area with the placeholder 'Message...' and a red 'Submit Review' button.

Write a Review

What is it like to Course?

☆☆☆☆☆

Message...

Submit Review

Figure 19: Write Review

Chapter 5: Implementation and testing

5.1 Challenges

We have 2 datasets: Brad dataset for Arabic and Amazon book reviews for English.

First Challenge: When labeling of Brad dataset, we notice label 2 (positive class) is 63.7%, label 0 (Negative class) is 15.4%, label 1 (Natural class) is 20.9%

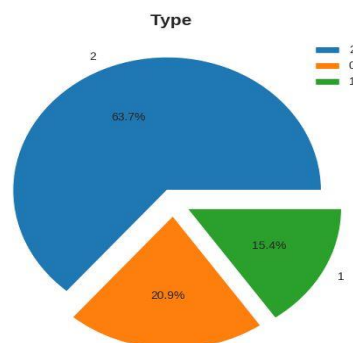


Figure 20 Unbalanced BRAD dataset

In Amazon book review English dataset, we found that 79% of data represent positive value, 12.4% represent neutral values and 8% represent negative value.

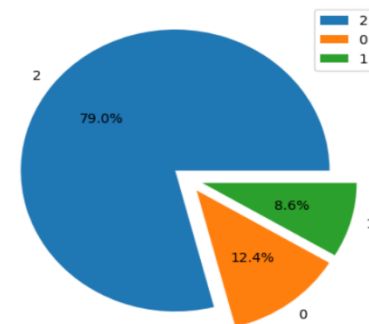


Figure 21 Unbalanced Amazon Dataset

Second Challenge: was when train two datasets after balancing we found that score for each class was less than 75% and neutral class biased with positive and negative classes.

Third Challenge was in diacritical marks in brad dataset for Arabic language when count it in dataset was 1345161 diacritical marks.

Fourth Challenge BRAD dataset contains more than one Arabic dialect that prevents model from determining correct label of the review.

Fifth Challenge review may be a combination of 2 language (Arabic and English).

Sixth Challenge review contains an equal number of words of the two languages.

5.2 Solution

For the first challenge we balance the two datasets according to their labels to prevent model to biased to positive value when predicting review sentiment.

Label 2 (Positive class) became 38.9%, Label 0 (Negative class) became 33.3%, Label 1 (Neutral class) became 27.8%.

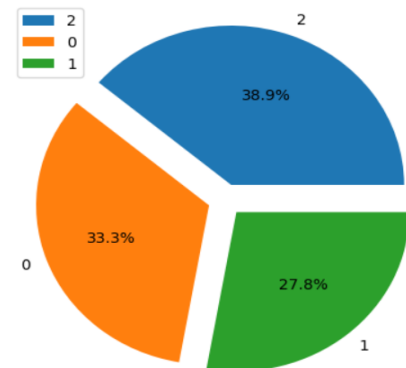


Figure 22 Balanced dataset.

For the second challenge we use Subjective – Objective structure to increase the score of each class and decrease problem of neutral class biased with positive and negative classes.

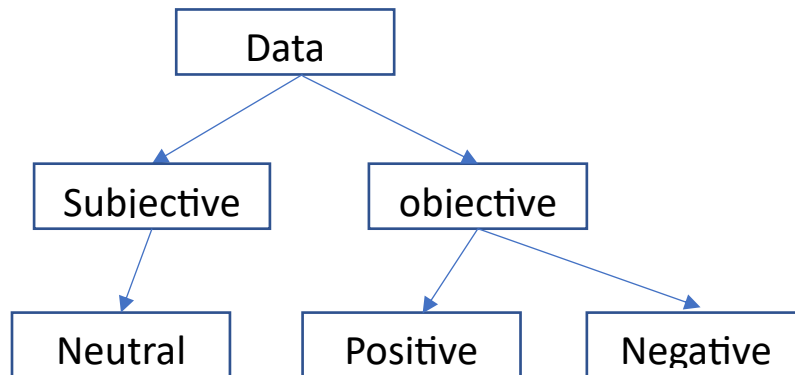


Figure 23 Tree Structure model

For third challenge we don't remove the diacritical marks in cleaning phase, and it makes result better in evaluation and testing phase like “الكتاب جيد جدا” or “الكتاب سيء جدا” make right prediction to this sentence.

For fourth challenge, for dialect detection, we use pre-trained model from hugging face called Arabic Bert Dialect identification. and the model accuracy according to the model's page on Hugging Face, it has achieved state-of-the-art performance on a number of Arabic dialect identification benchmarks, with an average accuracy of 88.2% on the MADAR dataset and an average F1 score of 0.918 on the ArTen-T Dataset.

For fifth challenge, make a language detection to determine the language of review based on the number of words of that language. Then apply review to its language model as we create two different models for each language.

For sixth challenge, If the number of words for both languages are equal, we apply the review to the Arabic model as it was multilingual and can accept any language or any combination of words of different languages.

5.3 Implementation

5.3.1 Data preparation

In this phase we transform the rating for each row to its class and then make preprocessing on review.

In Transform phase we deal with 2 models

In first model treat with 2 labels Subjective (Neutral) and Objective (Positive - Negative) classes and this is example of function we used

```
1- def mark_sentiment(rating):
2-     if (rating == 3):
3-         return 1
4-     else:
5-         return 0
```

Figure 24 Code for transform labels to objective and subjective.

Class	Subjective	Objective
Rating	[3]	[1 , 2 , 4 , 5]

Table 2 Classes for each label distribution

In second model treat with 2 labels Positive and Negative classes and this is example of function we used

```
1- def mark_sentiment(rating):
2-     if (rating < 3):
3-         return 1
4-     else:
5-         return 0
```

Figure 25 classes distribution for Positive and Negative model

Class	Negative	Positive
Rating	[1 , 2]	[4 , 5]

Table 3 Classes for each label distribution

5.3.2 Arabic Dialect detection

```

1- from transformers import AutoTokenizer,
2- AutoModelForSequenceClassification
3- tokenizer = AutoTokenizer.from_pretrained("lafifi-
4- 24/arabicBert_arabic_dialect_identification" , model_max_length=512)
5-
6- model = AutoModelForSequenceClassification.from_pretrained("lafifi-
7- 24/arabicBert_arabic_dialect_identification")
8-
9- def detectDialect(review):
10-     encoding = tokenizer(review, return_tensors='pt', padding=True,
11- truncation=True)
12-
13-     output = model(**encoding)
14-     predictions = output.logits.argmax(dim=1)
15-     p = int(predictions)
16-     label_map={
17- 0:'EG', 1:'SY', 2:'PL', 3:'KW', 4:'LB', 5:'LY', 6:'JO', 7:'DZ',
18- 8:'QA', 9:'AE', 10:'BH', 11:'SA', 12:'OM', 13:'MA', 14:'IQ',
19- 15:'TN', 16:'SD', 17:'YE', 18:'MSA'}
20-
21-     return label_map[p]

```

Figure 26 Code for Arabic dialect detection

After identifying the dialect of sentences, we found that there are 19 Dialects among BRAD.

- The most frequent dialect was Modern Standard Arabic and represents more than one third of data (35%).
- The Egyptian dialect represent more than 15% of the data.
- Both Bahrain and Kuwait dialects came about less than 10% of data.
- The rest of the Dialects came with a very close ratio but less than 5%.

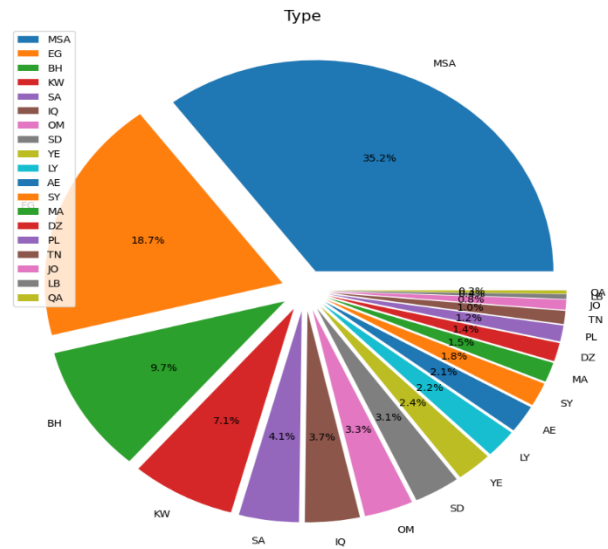


Figure 27 Arabic dialect percentage

5.3.3 Cleaning phase in Arabic language

```
1- def clean_text(text):
2-     ## Remove punctuations
3-     text = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-
4-     ./:;<=>?@[ ]^`{|}~"""), ' ', text)
5-
6-     ## remove extra whitespace
7-     text = re.sub('\s+', ' ', text)
8-     ## Convert text to lowercases
9-     text = text.lower()
10-    ## Remove stop words
11-    text = remove_stop_words(text)
12-    ## Remove numbers
13-    text = re.sub("\d+", " ", text)
14-    text = re.sub('[A-Za-z]+', ' ', text)
15-    text = re.sub(r'\u[A-Za-z0-9\]+', ' ', text)
16-    ## remove extra whitespace
17-    text = re.sub('\s+', ' ', text)
18-    #Stemming
19-    text = stemNLTK(text)
20-    return text
```

Figure 28 Arabic cleaning and preprocessing function

Here we do some steps to clean Arabic text.

Input sentence is “(VERY GOOD !100 ما اجمل الطبيعه)”

- Remove Punctuations after using will be “VERY GOOD 100 ما اجمل الطبيعه”
- Remove extra whitespace.
- Convert text to lowercase after using will be “very good 100 ما اجمل الطبيعه”
- Remove stop words will be after using will be “very good 100 اجمل الطبيعه”
- Remove numbers after using will be “very good اجمل الطبيعه”
- Remove English will be “اجمل الطبيعه”
- Stemming after using will be “جمل طبع”

5.3.4 Cleaning phase in English language

```
1- def clean_text(text):
2-     # Remove URLs
3-     text = re.sub(r'http\S+', ' ', text)
4-     # Remove punctuation and special characters
5-     text = re.sub(r'^\w\s]', ' ', text)
6-     # Lowercase the text
7-     text = text.lower()
8-     # Remove stop words
9-     stop_words = set(stopwords.words('english'))
10-    words = text.split()
11-    words = [word for word in words if word not in stop_words]
12-    text = ' '.join(words)
13-    # Lemmatization
14-    lemmatizer = WordNetLemmatizer()
15-    words = text.split()
16-    words = [lemmatizer.lemmatize(word) for word in words]
17-    text = ' '.join(words)
18-
19-    return text
```

Figure 29 English cleaning and preprocessing function

Here we make steps to clean English text

Input sentence: "Check out this amazing article on natural language processing: <https://www.example.com/nlp-article>! It's a great read, full of interesting insights."

Preprocessing steps:

Link Removal: The URL link in the input sentence is removed:

"Check out this amazing article on natural language processing! It's a great read, full of interesting insights."

Convert all sentences to small alphabet: switch each capital character to lower, to decrease number of unique words.

"check out this amazing article on natural language processing! it's a great read, full of interesting insights."

Stopword Removal: The stopwords in the input sentence are removed.

Stopwords are common words that do not carry much meaning, such as "this", "on", "a", "of", and "it":

"check amazing article natural language processing great read full interesting insights."

Punctuation Removal: The punctuation marks in the input sentence are removed:

"check amazing article natural language processing great read full interesting insights"

Lemmatization: The remaining words in the input sentence are lemmatized, which involves reducing them to their base form:

"check amaze article natural language process great read full interesting insight."

5.4 Algorithms

Support Vector Machine (SVM) model

Support Vector Machine is a popular machine algorithm tried for sentiment analysis in our dataset in Arabic (Brad dataset), we use it and make feature extraction which converts the preprocessed text data into numerical features with TF-IDF.

This example of code to use TF-IDF model.

```
1- from sklearn.feature_extraction.text import TfidfVectorizer
2- from sklearn.svm import SVC , SVR , LinearSVC
3- from sklearn.pipeline import make_pipeline
4- vectorizer=TfidfVectorizer()
5- clf = LinearSVC(dual=False)
6- pipe=make_pipeline(vectorizer,clf)
```

Figure 30 SVM and vectorization code

Here we import libraires to make feature extraction and use algorithms of support vector machine, create a pipeline function of data preprocessing and machine learning steps.

```
1- pipe.fit(X_train,y_train)
```

Figure 31 fitting data set.

The fit method is used to train the SVM model on the training data. It takes the training data and pipeline start to extract meaningful feature from the text data and use them to predict the sentiment of new text data like so:

```
1- y_pred=pipe.predict(X_test)
2- print(y_pred[1])
```

Figure 32 predict code for SVM.

The Classification report of first model (Positive – Negative), second model (Subjective – Objective)

Objective - Subjective		Positive - Negative	
F1 class 0: 0.75	F1 class 1: 0.70	F1 class 0: 0.83	F1 class 1: 0.85
F1: 0.73	Accuracy: 73.2%	F1: 0.84	Accuracy: 84%

Table 4 metrics of SVM model.

KNN model

In BRAD dataset, we tried K-nearest neighbors to make sentiment analysis classifier that predicts the sentiment based on the similar reviews in data of input with odd number k=7 and convert review into numerical for using TF-IDF vectorizer.

```

1- from sklearn.neighbors import KNeighborsClassifier
2- from sklearn.feature_extraction.text import TfidfVectorizer
3- from sklearn.pipeline import make_pipeline
4- vectorizer=TfidfVectorizer()
5- knn_Classifier = KNeighborsClassifier(n_neighbors=11 ,
6- algorithm='auto')
7- knn_pipe=make_pipeline(vectorizer,knn_Classifier)
8- knn_pipe.fit(X_train,y_train)

```

Figure 33 Code for KNN model and pipeline

Here is the classification report of positive-negative model and subjective-objective model.

Objective - Subjective		Positive - Negative	
F1 class 0: 0.69	F1 class 1: 0.18	F1 class 0: 0.15	F1 class 1: 0.71
F1: 0.55	Accuracy: 55%	F1: 0.56	Accuracy: 57%

Table 5 metrics of KNN model.

Logistic regression model

In BRAD dataset, we also tried logistic regression algorithm to make sentiment analysis classifier that predicts the sentiment based on estimation is made by applying binary classification on the data allocated to training and test data. and convert review into numerical for using TF-IDF vectorizer.

```
1- from sklearn.linear_model import LogisticRegression
2- from sklearn.pipeline import make_pipeline
3- from sklearn.feature_extraction.text import TfidfVectorizer
4- vectorizer=TfidfVectorizer()
5- LR_Classifier = LogisticRegression()
6- lr_pipe=make_pipeline(vectorizer,LR_Classifier)
7- lr_pipe.fit(X_train,y_train)
8- lr_y_pred=lr_pipe.predict(X_test)
9- lr_y_pred[1]
```

Figure 34 Code for Logistic regression model and pipeline

Here the classification report of positive-negative model and subjective-objective model:

Objective - Subjective		Positive - Negative	
F1 class 0: 0.76	F1 class 1: 0.71	F1 class 0: 0.81	F1 class 1: 0.83
F1: 0.74	Accuracy: 74%	F1: 0.82	Accuracy: 82%

Table 6 metrics of Logistic regression model.

Decision tree model

Decision trees can be used for classification tasks. It can handle different data types and discrete or continuous values, and continuous values can be converted into categorical values using thresholds. It can also handle missing values.

```
1- from sklearn.tree import DecisionTreeClassifier
2- from sklearn.pipeline import make_pipeline
3- DT_Classifier = DecisionTreeClassifier(criterion="entropy")
4- DT_pipe=make_pipeline(vectorizer,DT_Classifier)
5- DT_pipe.fit(X_train,y_train)
6- DT_y_pred=DT_pipe.predict(X_test)
7- DT_y_pred[1]
```

Figure 35 Code for Decision tree model and pipeline

Objective - Subjective		Positive - Negative	
F1 class 0: 0.64	F1 class 1: 0.58	F1 class 0: 0.66	F1 class 1: 0.71
F1: 0.61	Accuracy: 61%	F1: 0.69	Accuracy: 69%

Table 7 metrics of Decision tree model.

Naïve bayes model

It's probabilistic algorithm that uses the Bayes' theorem to classify a given text into one of sentiment categories (positive – negative - neutral) and the basic idea behind it is calculate the probability of a given text belonging to each sentiment category based on the frequency of word in the text each sentiment category in the training data. We use Count Vectorize to make feature extraction example.

```

1- from sklearn.feature_extraction.text import CountVectorizer
2- from sklearn.feature_extraction.text import TfidfTransformer
3- from sklearn.naive_bayes import MultinomialNB
4- X_train = X_train.values
5- y_train = y_train.values
6- clf = CountVectorizer()
7- X_train_cv = clf.fit_transform(X_train)
8- X_test_cv = clf.transform(X_test)
9- tf_transformer = TfidfTransformer(use_idf=True).fit(X_train_cv)
10- X_train_tf = tf_transformer.transform(X_train_cv)
11- X_test_tf = tf_transformer.transform(X_test_cv)
12- nb_clf = MultinomialNB()
13- nb_clf.fit(X_train_tf, y_train)

```

Figure 36 Code for Naive model and pipeline

First, we use Count Vectorizer to convert text into a matrix of token counts to data of train and test.

And implement naïve Bayes and start to fit and train the data and then predict and evaluate.

Here the classification report of positive-negative model and subjective-objective model:

Objective - Subjective		Positive - Negative	
F1 class 0: 0.76	F1 class 1: 0.65	F1 class 0: 0.80	F1 class 1: 0.84
F1: 0.72	Accuracy: 72%	F1: 0.82	Accuracy: 82%

Table 8 metrics of Naive model

LSTM Model with TensorFlow

LSTM is a type of recurrent neural network (RNN) that is commonly used for sequence- to-sequence modeling, where the input and output are both sequences of variable length. In the context of sentiment analysis, the input sequence and the output is a sentiment label.

Here is an example of LSTM model we tried it.

```
1- #define LSTM model
2- model = Sequential()
3- model.add(Embedding(vocab_size, 100, weights=[embedding_matrix],
4- input_length=max_length, trainable=True))
5- model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
6- model.add(Dense(2, activation='sigmoid'))
7- # compile model
8- model.compile(loss='binary_crossentropy', optimizer='adam',
9- metrics=['accuracy'])
10- # one hot encoding for labels
11- y_train = pd.get_dummies(y_train).values
12- y_test = pd.get_dummies(y_test).values
13- # training model
14- from keras.callbacks import EarlyStopping
15- history = model.fit(X_train, y_train, epochs=10, batch_size=1024 ,
16- validation_split=0.1 , verbose=1,
17- callbacks = [EarlyStopping(monitor='val_loss', patience=3,
18- min_delta=0.0001)])
19- # evaluation
20- loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
21- print("Test Accuracy: {:.4f}".format(accuracy))
```

Figure 37 Code for LSTM model using TensorFlow.

In this code the vocab size represents the vocabulary and embedding matrix using Glove word embedding, and max length represents the maximum length of the input sequence.

We use embedding layer to map each word in the input sequence to a dense vector representation. Then LSTM layer in model can capture the context and long-term dependencies between words in the input sequence and use Dense layer to make binary classification between to classes using sigmoid function and *binary_crossentropy* loss function is measure the error between predicted sentiment label and true sentiment label, *adam* optimizer to minimize the loss function and train model and finally evaluate it.

Here the classification report of positive-negative model and subjective-objective model:

Objective - Subjective		Positive - Negative	
F1 class 0: 0.75	F1 class 1: 0.69	F1 class 0: 0.82	F1 class 1: 0.84
F1: 0.72	Accuracy: 72%	F1: 0.83	Accuracy: 83%

Table 9 metrics of LSTM model.

CNN model with TensorFlow

CNN is type of neural network that is applied to text classification tasks such as sentiment analysis. In the context of sentiment analysis, the input is a sequence of words or tokens in a sentence and the output is a sentiment label.

Here is an example of CNN model we tried it.

```

1- #define CNN model
2- from keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten,
3- Dense
4- model = Sequential()
5- model.add(Embedding(vocab_size, 100, weights=[embedding_matrix],
6- input_length=max_length, trainable=True))
7- model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
8- model.add(MaxPooling1D(pool_size=2))
9- model.add(Flatten())
10- model.add(Dense(2, activation='sigmoid'))
11- # compile model
12- model.compile(loss='binary_crossentropy', optimizer='adam',
13- metrics=['accuracy'])
14- # one hot encoding for labels
15- y_train = pd.get_dummies(y_train).values
16- y_test = pd.get_dummies(y_test).values
17- #training
18- from keras.callbacks import EarlyStopping
19- history = model.fit(X_train, y_train, epochs=10, batch_size=1024 ,
20- validation_split=0.1, verbose=1,
21- callbacks=[EarlyStopping(monitor='val_loss', patience=3,
22- min_delta=0.0001)])
23- # evaluation
24- loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
25- print("Test Accuracy: {:.4f}".format(accuracy))

```

Figure 38 Code for CNN model using TensorFlow.

The Covn1D layer in model applies a 1D convolution operation to the input sequence, which can capture local features and pattern in the input sequence. The MaxPooling1D layer in model reduces the dimensionality of the features maps by taking the maximum value over each feature map.

Here the classification report of positive-negative model and subjective-objective model:

Objective - Subjective		Positive - Negative	
F1 class 0: 0.75	F1 class 1: 0.68	F1 class 0: 0.82	F1 class 1: 0.85
F1: 0.72	Accuracy: 72%	F1: 0.84	Accuracy: 84%

Table 10 metrics of CNN model.

CNN-LSTM model with PyTorch

CNN-LSTM architecture is a type of neural network that combines two powerful deep learning models: convolutional neural networks (CNNs) and long short-term memory (LSTM) networks.

The CNN layer is used to extract important features from the text, such as word embeddings and n-grams, and the LSTM layer is used to model the sequence of words in the text and capture the context and sentiment of the text. The output of the LSTM layer can then be fed into a fully connected neural network that produces the final sentiment prediction.

By combining these two models, the CNN-LSTM architecture can effectively capture both the local and global features of the text and can learn to classify the sentiment of the text accurately. This architecture has been shown to achieve state-of-the-art performance on sentiment analysis tasks in several studies.

The architecture consists of embedding layer that initially takes weights from W2V model that trained, then two convolution layers and five stacked LSTM component, and finally use ANN layer, also we used *Adam optimizer* and *Cross Entropy Loss* function.

Here is an example of CNN-LSTM model we tried it.

```

1- # create Tensors
2- train_data = TensorDataset(torch.from_numpy(X_train),
3- torch.from_numpy(y_train))
4- test_data = TensorDataset(torch.from_numpy(X_test),
5- torch.from_numpy(y_test))
6- valid_data = TensorDataset(torch.from_numpy(X_valid),
7- torch.from_numpy(y_valid))
8- # create data loaders
9- BATCH_SIZE = 50
10- train_loader = DataLoader(train_data, shuffle=True,
11- batch_size=BATCH_SIZE, drop_last=True)
12- valid_loader = DataLoader(valid_data, shuffle=True,
13- batch_size=BATCH_SIZE, drop_last=True)
14- test_loader = DataLoader(test_data, shuffle=True,
15- batch_size=BATCH_SIZE, drop_last=True)
16- # class of CNN LSTM
17- class CNNLSTMModel(nn.Module):
18-     def __init__(self, embedding_dim, hidden_dim, vocab_size,
19- num_classes=2):
20-         super(CNNLSTMModel, self).__init__()
21-         self.embedding = nn.Embedding(vocab_size, embedding_dim)
22-         self.conv1 = nn.Conv1d(in_channels=embedding_dim,
23- out_channels=hidden_dim, kernel_size=3, padding=1)
24-         self.conv2 = nn.Conv1d(in_channels=hidden_dim,
25- out_channels=hidden_dim, kernel_size=3, padding=1)
26-         self.lstm = nn.LSTM(input_size=hidden_dim,
27- hidden_size=hidden_dim, num_layers=5, batch_first=True)
28-         self.fc = nn.Linear(hidden_dim, num_classes)
29-
30-     def forward(self, x):
31-         x = self.embedding(x)
32-         x = x.permute(0, 2, 1)
33-         x = self.conv1(x)
34-         x = F.relu(x)
35-         x = self.conv2(x)
36-         x = F.relu(x)
37-         x = x.permute(0, 2, 1)
38-         x, _ = self.lstm(x)
39-         x = x[:, -1, :]
40-         x = self.fc(x)
41-         return x
42- # create model and copy data to GPU
43- model = CNNLSTMModel(EMBEDDING_DIM, 128 , VOCAB_SIZE)
44- model = model.to('cuda')
45- #Initialize embedding with the previously defined embedding matrix
46- model.embedding.weight.data.copy_(
47- torch.from_numpy(embedding_matrix))
48- #Allow the embedding matrix to be fined tuned to better adapt to out
49- dataset and get higher accuracy
50- model.embedding.weight.requires_grad=True

```

```

51- # define optimizer
52- import torch.optim as optim
53- optimizer = optim.Adam(model.parameters())
54-
55- # Define the loss function
56- loss_fn = nn.CrossEntropyLoss()
57-
58- N_EPOCHS = 10
59-
60- for epoch in range(N_EPOCHS):
61-     train_loss, train_acc = train(model, train_loader, optimizer,
62-     loss_fn)
63-     valid_loss, valid_acc = evaluate(model, valid_loader, loss_fn)
64-
65-     print(f'Epoch: {epoch+1:02}')
66-     print(f'\tTrain Loss: {train_loss:.3f} | Train Acc:
67- {train_acc*100:.2f}%')
68-     print(f'\t Val. Loss: {valid_loss:.3f} | Val. Acc:
69- {valid_acc*100:.2f}%')

```

Figure 39 Code for CNN-LSTM model using PyTorch.

Here the classification report of positive-negative model and subjective-objective model:

Objective - Subjective		Positive - Negative	
F1 class 0: 0.72	F1 class 1: 0.67	F1 class 0: 0.85	F1 class 1: 0.82
F1: 0.70	Accuracy: 71%	F1: 0.838	Accuracy: 83.9%

Table11 metrics of CNN-LSTM model.

Transformer

Transformers models are a type of neural network architecture this is practically well-suited for natural language processing tasks such as sentiment analysis. They are based on the attention mechanism, which allows the model to selectively focus on certain parts of the input sequence.

To perform sentiment analysis with *PyTorch* transformers models, we would typically start by preprocessing our text data to convert it into a format that can be fed into the model. This might involve tokenizing the text into individual words or sub words, and then converting these tokens into numerical representations such as embeddings.

After preprocessed our data, we train a *PyTorch* transformer model on a labeled dataset of text samples with corresponding sentiment labels, during training the model learns to predict the sentiment label for each input text sample based on its context and sentiments labels of similar text samples.

After the Training Process, we can use the trained model to predict the sentiment of new text samples.

And this Example of predict function we used:

```
1- device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
2- def predict(text):
3-     text = clean_text(text)
4-     encoded_review = tokenizer.encode_plus(text,
5-     max_length=MAX_LEN,
6-     add_special_tokens=True,
7-     return_token_type_ids=False,
8-     pad_to_max_length=True,
9-     truncation='longest_first',
10-    return_attention_mask=True,
11-    return_tensors='pt'
12-    )
13-    input_ids = encoded_review['input_ids'].to(device)
14-    attention_mask = encoded_review['attention_mask'].to(device)
15-    output = model(input_ids, attention_mask)
16-    _, prediction = torch.max(output[0], dim=1)
17-    return prediction[0].cpu().detach().numpy()
```

Figure 40 transformer predict function

This code defines a function predict(text) that takes a string of text as input, preprocesses it, encodes it using a pre-trained tokenizer, passes it to a pre-trained neural network, and returns the predicted class label for the input text.

Transformer for Amazon book review dataset for English Model

We use DistilBert Transformer for Sentiment analysis in English language, DistilBert is a compressed version of the Bert (Bidirectional Encoder Representations from Transformers) model developed by Hugging Face. It is designed to be smaller and faster than original BERT model while retaining a high level of accuracy.

DistilBert achieves this by using a combination of knowledge distillation and parameter sharing technique.

Example of using DistilBert Transformer:

```

1- import transformers
2- from transformers import BertTokenizer
3- from torch.utils.data import TensorDataset
4- from transformers import DistilBertTokenizerFast
5- MAX_LEN = 128
6- # Load the tokenizer
7- tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-
8- uncased')
9- # Tokenize the sentences and convert them to numerical input
10- input_ids = []
11- attention_masks = []
12- # For every sentence in the dataset
13- for sent in data['review']:
14-     # Encode the sentence using the tokenizer
15-     encoded_sent = tokenizer.encode_plus(
16-         text=sent, # Sentence to encode
17-         add_special_tokens = True, # Add [CLS] and [SEP]
18-         max_length = MAX_LEN, # Truncate longer sentences
19-         pad_to_max_length = True, # Pad shorter sentences
20-         return_attention_mask = True, # Generate attention mask
21-         return_tensors='pt' # Return PyTorch tensors
22-     )
23-     # Add the numerical input and attention mask to the lists
24-     input_ids.append(encoded_sent['input_ids'])
25-     attention_masks.append(encoded_sent['attention_mask'])
26-
27- # Load the pre-trained model
28- model = AutoModelForSequenceClassification.from_pretrained(
29-     "distilbert-base-uncased",
30-     num_labels=2, # Number of output labels
31-     output_attentions=False,
32-     output_hidden_states=False,
33- )

```

Figure 41 data preparation for transformer and model creation for English.

In this code load tokenizer from distilbert model from hugging face
 And then start to give input ids and attention masks for each review.
 And load pre-trained model “distilbert - base-uncased”

```

1- # Define the training arguments
2- training_args = TrainingArguments(
3-     output_dir='./results',
4-     num_train_epochs = 4,
5-     per_device_train_batch_size = 128,
6-     per_device_eval_batch_size = 128,
7-     warmup_steps=0,
8-     weight_decay=0.01,
9-     logging_dir='./logs',
10-    logging_steps=10,
11-    evaluation_strategy='epoch',
12-    save_total_limit=1,
13-    learning_rate=2e-5,
14- )
15-
16- # Assuming you have defined model, eval_dataset, and args
17- def compute_metrics(eval_prediction_output, labels=None):
18-     logits = eval_prediction_output.predictions
19-     preds = logits.argmax(axis=-1)
20-     acc = accuracy_score(labels, preds)
21-     f1 = f1_score(labels, preds, average='weighted')
22-     return {'accuracy': acc, 'f1': f1}
23-
24- trainer = Trainer(
25-     model=model,
26-     args=training_args,
27-     train_dataset = torch.utils.data.TensorDataset(train_inputs,
28- train_masks, train_labels),
29-
30-     eval_dataset = torch.utils.data.TensorDataset(test_inputs,
31- test_masks, test_labels),
32-     data_collator = lambda data: {
33-         'input_ids': torch.stack([item[0] for item in data]),
34-         'attention_mask': torch.stack([item[1] for item in data]),
35-         'labels': torch.stack([item[2] for item in data])},
36-
37-     compute_metrics = lambda eval_prediction_output:
38-         compute_metrics(eval_prediction_output, test_labels))
39-
40- # Train the model
41- trainer.train()
42-
43- # Evaluate the model
44- trainer.evaluate()

```

Figure 42 trainer object and training phase and evaluation

First, we start to initialize important parameters for training like number of epochs and batch size for train and batch and learning rate.

Second, we make compute Metrix that calculate accuracy and f1-score.

Third make trainer then train and evaluate model.

Here the classification report of positive-negative model and subjective-objective model:

Objective - Subjective		Positive - Negative	
F1 class 0: 0.81	F1 class 1: 0. 80	F1 class 0: 0.89	F1 class 1: 0.90
F1: 0.80	Accuracy: 80%	F1: 0.908	Accuracy: 90.8%

Table 12 metrics of transformer English model.

Transformer for Brad dataset for Arabic Model

We use *MARBERT* Transformers for sentiment analysis in Arabic language, The *MARBERT* transformer model is a variant of the Transformer architecture that was specifically developed for Arabic language processing. It is pre-trained on large amounts of Arabic text using a masked language modeling objective, which allows it to learn representations of Arabic words and their relationships to one another.

To use the *MARBERT* transformer model for sentiment analysis in Arabic, would first need to fine-tune it on a labeled dataset of Arabic text that includes sentiment labels (e.g., positive, negative, or neutral). During fine-tuning, the model learns to predict the sentiment label of a given piece of text based on its pre-trained knowledge of Arabic language.

This example of code to load *MARBERT* transformer model and prepare it.

```

1- Model_Used = "UBC-NLP/MARBERT"
2- Task_Name = "classification"
3-
4- class Dataset:
5-     def __init__(
6-         self,
7-         name,
8-         train,
9-         test,
10-        label_list,
11-    ):
12-        self.name = name
13-        self.train = train
14-        self.test = test
15-        self.label_list = label_list
16-
17- class BERTModelDataset(Dataset):
18-     def __init__(self, text, target, model_name, max_len, label_map):
19-         super(BERTModelDataset).__init__()
20-         self.text = text
21-         self.target = target
22-         self.tokenizer_name = model_name
23-         self.tokenizer = AutoTokenizer.from_pretrained(model_name)
24-         self.max_len = max_len
25-         self.label_map = label_map
26-
27-     def __len__(self):
28-         return len(self.text)
29-
30-     def __getitem__(self, item):
31-         text = str(self.text[item])
32-         text = " ".join(text.split())
33-
34-         encoded_review = self.tokenizer.encode_plus(
35-             text,
36-             max_length= self.max_len,
37-             add_special_tokens= True,
38-             return_token_type_ids=False,
39-             pad_to_max_length=True,
40-             truncation='longest_first',
41-             return_attention_mask=True,
42-             return_tensors='pt'
43-         )
44-         input_ids = encoded_review['input_ids'].to(device)
45-         attention_mask = encoded_review['attention_mask'].to(device)
46-         return InputFeatures(input_ids=input_ids.flatten(),
47-                               attention_mask=attention_mask.flatten(),
48-                               label=self.label_map[self.target[item]])

```

Figure 43 Dataset Class to handle dataset and items.

Here we prepare our Arabic dataset to use it in pretrained MARBERT Transformer model by splitting data to train and text and make label list, then start to initialize different things like load tokenizer from MARBERT Model and put maximum length for sentences. Finally, we get the input ids and attention mask for each sentence.

Example of code to initialize model and scores and splitting data:

```
1- def model_init():
2-     return
3-     AutoModelForSequenceClassification.from_pretrained(Model_Used,
4-     return_dict=True, num_labels=len(label_map))
5-
6- def compute_metrics(p): #p should be of type EvalPrediction
7-     preds = np.argmax(p.predictions, axis=1)
8-     assert len(preds) == len(p.label_ids)
9-     print(classification_report(p.label_ids,preds))
10-    macro_f1_pos_neg =
11-    f1_score(p.label_ids,preds,average='macro',labels=[1,2])
12-    macro_f1 = f1_score(p.label_ids,preds,average='macro')
13-    macro_precision =
14-    precision_score(p.label_ids,preds,average='macro')
15-    macro_recall = recall_score(p.label_ids,preds,average='macro')
16-    acc = accuracy_score(p.label_ids,preds)
17-    return {
18-        'macro_f1' : macro_f1,
19-        'macro_f1_pos_neg' : macro_f1_pos_neg,
20-        'macro_precision': macro_precision,
21-        'macro_recall': macro_recall,
22-        'accuracy': acc
23-    }
24-
25- label_list = list(train_set['label'].unique())
26-
27- print(label_list)
28- print(train_set['label'].value_counts())
29-
30- data_set = Dataset( "BRAD", train_set, evaluation_set, label_list )
31-
32- label_map = { v:index for index, v in enumerate(label_list) }
33- print(label_map)
34-
35- train_dataset = BERTModelDataset(train_set['review'].to_list(),
36-
37- train_set['label'].to_list(),Model_Used,Max_Len,label_map)
38-
39- evaluation_dataset =
40- BERTModelDataset(evaluation_set['review'].to_list(),
```

```

41-
42- evaluation_set['label'].to_list(),Model_Used,Max_Len,label_map)
43-
44- test_dataset = BERTModelDataset(train_set['review'].to_list(),
45-
46- train_set['label'].to_list(),Model_Used,Max_Len,label_map)

```

Figure 44 data preparation for transformer and model creation for Arabic.

Here we initialize the model we will use and then prepare scores f1-score, recall, accuracy, precision then passing our data set then split data to train, test, evaluate. Example of code to initialize trainer and train then evaluate model:

```

1- # Setting Attribute
2- training_args = TrainingArguments("./train")
3- training_args.lr_scheduler_type = 'cosine'
4- training_args.evaluate_during_training = True
5- training_args.adam_epsilon = 1e-9
6- training_args.fp16 = True
7- training_args.per_device_train_batch_size = 12 #64
8- training_args.per_device_eval_batch_size = 12 # 64
9- training_args.gradient_accumulation_steps = 2
10- training_args.num_train_epochs= 4
11- training_args.warmup_steps = 0
12- training_args.evaluation_strategy = EvaluationStrategy.EPOCH
13- training_args.logging_steps = 200
14- training_args.save_steps = 100000
15- training_args.seed = 42
16- training_args.disable_tqdm = False
17- import os
18- training_args.dataloader_pin_memory = False
19- torch.cuda.empty_cache()
20- set_seed(Rand_Seed)
21- model1 = model_init()
22- # Passing Attribute
23- trainer = Trainer(
24-     model = model1,
25-     args = training_args,
26-     train_dataset = train_dataset,
27-     eval_dataset= evaluation_dataset,
28-     compute_metrics=compute_metrics
29- )
30- # Train Model
31- trainer.train()
32- # Evalute model
33- trainer.evaluate(test_dataset)

```

Figure 45 trainer object and training phase and evaluation.

First, we start to initialize important parameters for training like number of epochs and batch size for train and batch and learning rate.

Second, we make compute Metrix that calculate accuracy and f1-score

Third make trainer then train and evaluate model.

Here the classification report of positive-negative model and subjective-objective model:

Objective - Subjective		Positive - Negative	
F1 class 0: 0.88	F1 class 1: 0.90	F1 class 0: 0.96	F1 class 1: 0.96
F1: 0.89	Accuracy: 89%	F1: 0.96	Accuracy: 96%

Figure 46 metrics of transformer Arabic model

5.5 Summarization

We use `pszemraj/led-large-book-summary` pre-trained language model that trained using Large-scale Encoder-Decoder (LED) architecture that based on the Large Encoder-Decoder Transformer (LED) architecture, which is a variant of the Transformer architecture used in models like GPT-3. The LED architecture was introduced in the paper "Longformer: The Long-Document Transformer" by Beltagy et al. (2020) and was designed to handle long input sequences, up to several thousand tokens.

the model is fine-tuned on a task of generating summaries of books. The model is trained to take in a long document as input and output a summary of the document. The model was trained on a dataset of books and their summaries, and the goal was to generate summaries that capture the main ideas and themes of the books. With a total of 1.1 billion parameters, which makes it a very powerful language model.

The model was trained using the Hugging Face library and is available for use through the Hugging Face Model Hub. It can be used for various text summarization tasks such as summarizing news articles, scientific papers, and other types of documents.

We used this model because we have not the required computational power for training, and this model trained on same dataset we would use "Bill-Sum" that consists of long sequence texts and its summary sequence:

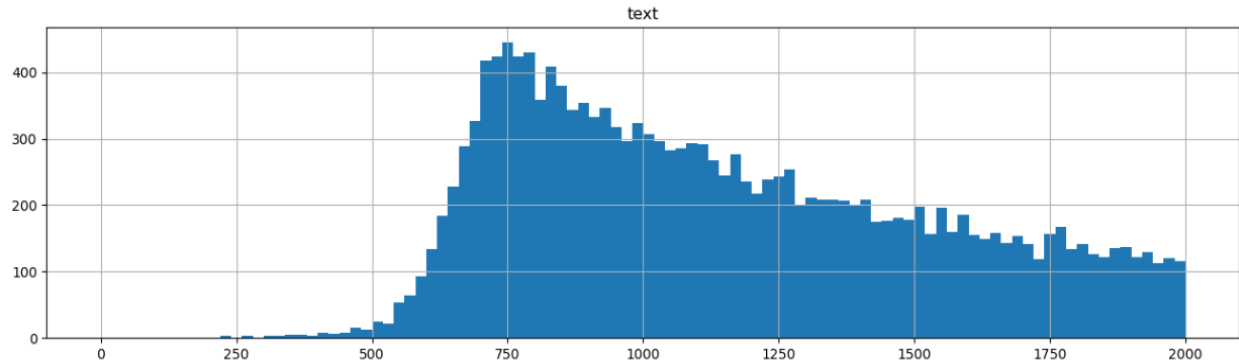


Figure 47 txt length histogram

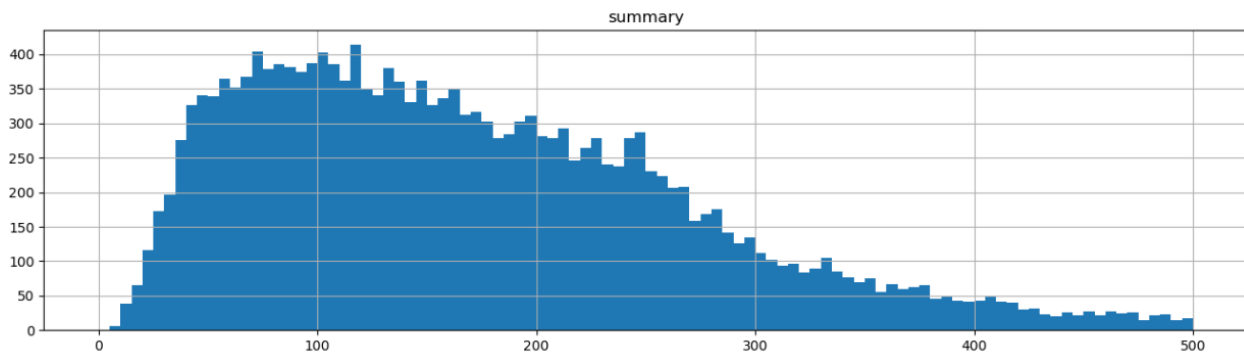


Figure 48 length of summary histogram

The testing phase done on different datasets:

Bill-Sum			
ROUGE-1 = 40.58	ROUGE-2 = 17.34	ROUGE-L = 25.126	ROUGE-LSUM = 34.6
Book-Sum			
ROUGE-1 = 31.7	ROUGE-2 = 5.3	ROUGE-L = 16.1	ROUGE-LSUM = 29

Table 13:Summary Testing Results

5.6 deployment for each model

Language Detection: Here we detect the language of review to determine which model should be used to predict the sentiment of review based on the language of the review.

```
1- from langdetect import detect
2- def detectText(review):
3-     lan = detect(review)
4-     if lan == "en":
5-         return "en"
6-     else:
7-         return "ar"
```

Figure 49 Language Detection

English spelling correction: we used to “autocorrect” library to correct English words.

```
1- from autocorrect import Speller
2- def correctEnglish(review):
3-     spell = Speller()
4-     return spell(review)
```

Figure 50 English spelling correction

Arabic spelling correction: we used ar-corrector library to correct Arabic words.

```
1- from ar_corrector.corrector import Corrector
2- def correctArabic(review):
3-     corr = Corrector()
4-     return corr.contextual_correct(review)
```

Figure 51 Arabic spelling correction

5.7 APIs Implementation

```
1- app = Flask(__name__)
2- @app.route('/predict/<string:sentence>', methods=['GET'])
3- def predict1(sentence):
4-     # if this language of sentence is arabic
5-     if detectText(sentence) == 'ar':
6-         # Load first model of arabic which represent Subjective or
7-         # Objective class
8-         model1 = torch.load('Arabic_Subjective_Objective.pt',
9-                             map_location=torch.device('cpu'))
10-        # Load Tokenizer for MARBERT transformer
11-        tokenizer = AutoTokenizer.from_pretrained("UBC-NLP/MARBERT")
12-        # Make Auto Correction to arabic sentence
13-        sentence_after_correction = corr.contextual_correct(sentence)
14-        temp = sentence_after_correction
15-        # Make Arabic Cleaning process to sentence
16-        sentence_after_correction_cleaned =
17-        clean_text_arabic(sentence_after_correction)
18-        # Make predict (subjective - objective) (1 - 0)
19-        result = predict(sentence_after_correction_cleaned,model1 ,
20-                          tokenizer , 106)
21-        # Result is zero mean it Objective (Postive - Negative )
22-        if str(result) == '0':
23-            # Load second model which represent Positive or Negative
24-            model2 = torch.load('Arabic_Postive_Negative.pt',
25-                                map_location=torch.device('cpu'))
26-            # Make Cleaning proecess to sentence
27-            sentence_after_correction_cleaned =
28-            clean_text_arabic(temp)
29-            # Make Predict (postive - negative)(1 - 0)
30-            result =
31-            predict(sentence_after_correction_cleaned,model2,
32-                    tokenizer, 106)
33-            return str(result)
34-        # Otherwise mean it Subective (Neutral)
35-        else:
36-            return '-1'
37-    # this language of sentence is english
38-    else:
39-        # Load first model of english which represent Subjective or
40-        # Objective class
41-        model1 = torch.load('English_subjective_objective.pt',
42-                            map_location=torch.device('cpu'))
43-        # Load tokenizer for distilbert-base-uncased transformer
44-        tokenizer = AutoTokenizer.from_pretrained("distilbert
45-        baseuncased")
46-        # Make Auto Correction to english sentence
47-        sentence_after_correction = spell(sentence)
```

```

48- temp = sentence_after_correction
49- # Make English Cleaning process to sentence
50- sentence_after_correction_cleaned =
51- clean_text_english(sentence_after_correction)
52- # Make predict (subjective - objective) (1 - 0)
53- result = predict(sentence_after_correction_cleaned,
54- model1 , tokenizer , 128)
55- # Result is zero mean it Objective (Positive - Negative )
56- if str(result) == '0':
57-     # Load second model which represent Positive or Negative
58-     model2 =
59-     torch.load('English_pos_neg.pt',
60-         map_location=torch.device('cpu'))
61-     # Make English Cleaning process to sentence
62-     sentence_after_correction_cleaned =
63-     clean_text_english(temp)
64-     # Make Predict (positive - negative)(1 - 0)
65-     result = predict(sentence_after_correction_cleaned,
66-         model2 , tokenizer , 128)
67-     return str(result)
68- # Otherwise mean it Subjective (Neutral)
69- else:
70-     return '-1'

```

Figure 52 sentiment analysis API using Flask.

Second make Summarization of a collection of texts

```

1- @app.route('/summaryText/<string:text>', methods=['GET'])
2- def summary_text(text):
3-     # Represent name of summary model
4-     model_name = "pszemraj/led-large-book-summary"
5-     # Load Tokenizer for summarization
6-     tokenizer = LEDTokenizer.from_pretrained(model_name)
7-     # The model is also initialized with the pre-trained model name
8-     defined above
9-     model = LEDForConditionalGeneration.from_pretrained(model_name)
10-    # The encoder method
11-    inputs = tokenizer.encode(text, padding="max_length",
12-        truncation=True, max_length=4096, return_tensors="pt")
13-    # Summary of encoded text using pretrained model
14-    summary_ids = model.generate(inputs, max_length=512, num_beams=5,
15-        early_stopping=True)
16-    # Decode the summary output using tokenizer
17-    summary = tokenizer.decode(summary_ids[0],
18-        skip_special_tokens=True)
19-    return summary

```

Third Make Summarization of a YouTube video.

```
1- @app.route('/summaryYoutube/<string:video_id>', methods=['GET'])
2- def summary_youtube(video_id):
3-     # Fetch content of video with its id
4-     textParts = YouTubeTranscriptApi.get_transcript(video_id)
5-     # Merge parts of texts together
6-     text = ' '.join([c['text'] for c in textParts])
7-     # Represent name of summary model
8-     model_name = "pszemraj/led-large-book-summary"
9-     # Load Tokenizer for summarization
10-    tokenizer = LEDTokenizer.from_pretrained(model_name)
11-    # The model is also initialized with the pre-trained model name
12-    # defined above
13-    model = LEDForConditionalGeneration.from_pretrained(model_name)
14-    # The encoder method
15-    inputs = tokenizer.encode(text, padding="max_length",
16-                             truncation=True, max_length=4096, return_tensors="pt")
17-    # Summary of encoded text using pretrained model
18-    summary_ids = model.generate(inputs, max_length=512, num_beams=5,
19-                                early_stopping=True)
20-    # Decode the summary output using tokenizer
21-    summary = tokenizer.decode(summary_ids[0],
22-                               skip_special_tokens=True)
23-    return summary
```


Reference

Sentiment Analysis for Course Recommendation and Improvement" by E. B. S. Almeida, L. A. V. Nunes, and A. A. F. Loureiro:

https://link.springer.com/chapter/10.1007/978-3-319-45234-0_2

Shinde, P. L., & Patil, S. B. (2020). A novel approach for course recommendation system using sentiment analysis and summarization techniques. *International Journal of Computer Science and Mobile Computing*, 9(5), 52-59.

"Sentiment Analysis on MOOCs Reviews: A Hybrid Approach"

Zhang, Y., Chen, Y., & Lu, L. (2018). A Sentiment Analysis and Summarization for MOOC Reviews. In *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* (pp. 198-203). IEEE.

Lee, S. H., & Lee, J. Y. (2019). Sentiment analysis and summarization of student feedback for course improvement. *International Journal of Emerging Technologies in Learning (iJET)*, 14(14), 178-194.

Kim, K. K., Kang, Y. S., & Lee, S. H. (2018). Automated Course Evaluation Based on Semantic Analysis and Summarization Techniques. *Journal of Digital Convergence*, 16(11), 33-43.

Park, S. S., Kim, S. H., & Kim, J. J. (2017). Course Evaluation Analysis Using Sentiment Analysis and Visualization Techniques. *Journal of Digital Convergence*, 15(9), 13-23.

Kaur, A., & Kaur, A. (2021). Sentiment Analysis of Online Course Reviews: A Study on Coursera. *International Journal of Emerging Technologies in Learning (iJET)*, 16(1), 111-125.