Mini Project Report

on

GENAI CONTENT HUB USING TRANSFORMERS

(CSE VI Semester Project report)

2023-2024

Submitted to:

Dr. Piyush Bagla

(CC- B.Tech. Section-A-VI-sem)

Submitted by:

Akanksha

Roll no:2118160

B.Tech. Section-A-VI-Sem

Session:2023-2024

Bachelor of Technology(B.Tech.) GRAPHIC ERA HILL UNIVERSITY,DEHRADUN

CERTIFICATE

(from Internal Co-ordinator of mini project i.e. Class Coordinator)

Certified that Ms AKANKSHA (Roll No.-2118160) have completed Mini project on the topic "**GENAI CONTENT HUB USING TRANSFORMERS**" for fulfillment of CSE VI Semester Mini Project in Graphic Era Hill University, Dehradun.The student has successfully completed this course to the best of my knowledge.

Date:13 July 2024

Dr. PIYUSH BAGLA

Class Coordinator

B.Tech. Section-A-VI-Semester

GEHU,Dehradun

ACKNOWLEDGEMENT

I would like to express our gratitude to the Almighty ,the most beneficent and the most merciful, for successful completion of Mini Project.

I wish to thank my parent for their continuing support and encouragement .I also wish to thank them for providing me with the opportunity to reach this far in our studies.

I also acknowledge to my class coordinator and subject teacher Dr. PIYUSH BAGLA sir who helped me to understand this course.

At last but not the least ,I greatly indebted to others who directly or indirectly helped me during this course.

Ms. AKANKSHA

Roll No-2118160

B.Tech. Section-A-VI-Semester

Session: 2023-2024

GEHU, Dehradun

TABLE OF CONTENTS

1. INTRODUCTION
2. METHODOLOGY
3. FUNCTIONALITY OVERVIEW
4. CLIENT INTERFACE SYPNOSIS
5. CODING SYPNOSIS
6. RESULT AND CONCLUSION
7. REFERENCES

1. INTRODUCTION

In today's digital age, the demand for intelligent systems capable of processing and generating vast amounts of content has grown significantly. The AI Content Hub project emerges as a response to this demand, aiming to leverage advanced artificial intelligence (AI) techniques to facilitate text summarization, content generation, and specialized document analysis. This report details the development, implementation, and evaluation of the AI Content Hub, highlighting its innovative use of state-of-the-art AI models to enhance productivity and decision-making across various domains.

Project Overview
AI Content Hub integrates sophisticated AI algorithms and models to address diverse user needs through a unified platform. Key functionalities include:

- Text Summarization: Utilizing PEGASUS, an advanced transformer model, for abstractive text summarization, allowing users to distill comprehensive texts into concise summaries while preserving essential information.
- Content Generation: Leveraging Gemini-pro from Google's Generative AI capabilities to generate contextually relevant textual and visual content based on user prompts, enhancing creativity and efficiency in content creation tasks.
- Image and PDF Analysis: Enabling users to extract insights and answers from uploaded images and PDF documents using Gemini-pro, facilitating rapid information retrieval and analysis.
- Medical Query Resolution: Providing accurate and detailed medical advice through natural language processing and AI-driven reasoning, ensuring adherence to medical standards and guidelines.

Objectives
The primary objectives of AI Content Hub are to:

- Simplify and accelerate content creation processes for textual and visual content.
- Enhance information accessibility and decision-making through efficient document analysis.
- Provide reliable and contextually appropriate medical advice to users seeking health-related information.

Scope
The scope of the project encompasses the development of a robust web application using Flask, Python's micro web framework, integrated with powerful AI models. The application is designed to be scalable and adaptable, catering to both individual users and organizational needs across various sectors, including education, healthcare, and digital marketing.
This report comprehensively documents the methodologies, implementation details, results, and future directions of the AI Content Hub, offering insights into its innovative approach towards leveraging AI for content generation and specialized document analysis.

2. METHODOLOGY

Technology used

- **Flask Framework**: AI Content Hub is built on the Flask framework, a lightweight and flexible Python web framework. Flask provides the foundation for handling HTTP requests, routing, and rendering HTML templates, facilitating the development of a responsive and scalable web application.

- **Transformers Library**: The project utilizes the Transformers library from Hugging Face, which offers a comprehensive collection of pre-trained state-of-the-art models for natural language processing tasks. Specifically, models like PEGASUS for text summarization .

- **Google Generative AI (Gemini-pro)**: For image and PDF analysis and content generation tasks, AI Content Hub leverages Google's Generative AI capabilities through Gemini-pro. This includes querying uploaded images and PDF documents to extract relevant information and generate responses based on user queries, thereby expanding the application's utility beyond textual content. Gemini-pro utilizes transformer-based architectures, which have revolutionized the field of NLP. Transformers employ self-attention mechanisms to weigh the importance of different words in a sentence, enabling the model to capture dependencies and long-range dependencies effectively. This architecture facilitates both the understanding of input queries and the generation of coherent responses.

- **Python Libraries**: Various Python libraries such as Spacy for natural language processing tasks (used in the text summarization function), AutoTokenizer and AutoModelForSeq2SeqLM from Hugging Face Transformers for integrating and deploying AI models, and dotenv for managing environment variables are employed to streamline development and ensure efficient handling of data and models.

Model Architecture

1.spaCy

spaCy is a robust and versatile natural language processing (NLP) library designed for efficient text processing in Python. It offers a comprehensive suite of tools and functionalities that enable developers and researchers to perform various NLP tasks with ease. At its core, spaCy provides capabilities such as tokenization, part-of-speech tagging, named entity recognition (NER), dependency parsing, and more.1

One of spaCy's key strengths lies in its efficiency and speed, making it suitable for processing large volumes of text data in production environments. The library is optimized for performance and memory usage, allowing developers to build scalable NLP applications without compromising on processing speed.

Another notable feature of spaCy is its built-in support for pre-trained models and word embeddings. These models enable tasks such as entity recognition across different domains, syntactic parsing for understanding sentence structure, and semantic analysis through word vectors. Moreover, spaCy integrates seamlessly with other popular libraries and frameworks like TensorFlow and PyTorch, facilitating the incorporation of advanced machine learning models into NLP pipelines.

Furthermore, spaCy supports multiple languages and provides language-specific models, enabling developers to apply NLP techniques across diverse linguistic contexts. This multilingual capability makes spaCy versatile for global applications where text processing needs to accommodate various languages.

2.Transformers

Transformer architecture, introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017, revolutionized natural language processing and many other fields. The key innovation is the self-attention mechanism, which allows the model to weigh the importance of different words in a sentence irrespective of their position. Here's a high-level overview of the transformer architecture:

1. Input Embeddings

- Token Embeddings: Each word or token in the input sequence is converted into a dense vector of fixed size.
- Positional Encodings: Since the transformer does not inherently understand the order of words, positional encodings are added to the token embeddings to provide information about the position of each word in the sequence.

2. Encoder

The encoder is a stack of identical layers, each consisting of two main components:

- Multi-Head Self-Attention Mechanism: This allows the model to focus on different parts of the input sequence simultaneously. It computes a set of attention scores for each word pair in the sequence and creates multiple attention heads to capture different types of relationships.
- Feed-Forward Neural Network: A fully connected network applied to each position separately and identically. It consists of two linear transformations with a ReLU activation in between.

Each encoder layer also includes:

- Layer Normalization: Applied before the attention and feed-forward layers to stabilize and speed up training.
- Residual Connections: These connections add the input of each sub-layer to its output, helping to prevent vanishing gradients.

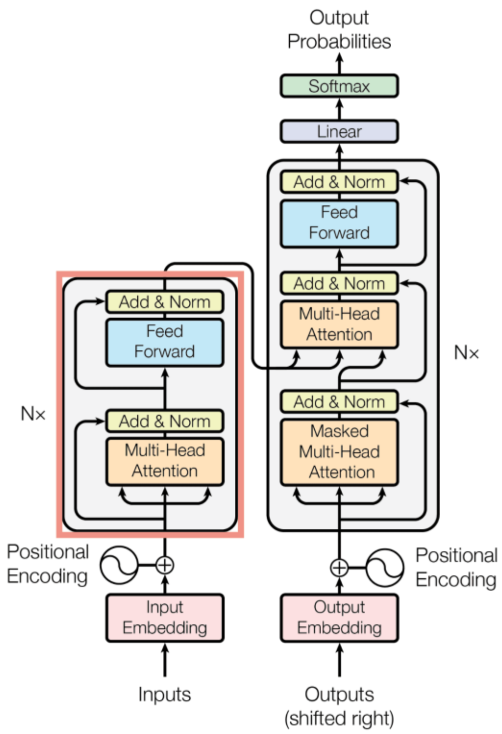
3. Decoder

The decoder is also a stack of identical layers, with some key differences:

- Masked Multi-Head Self-Attention Mechanism: Similar to the encoder’s self-attention but prevents positions from attending to subsequent positions, ensuring that the prediction for a position depends only on the known outputs at previous positions.
- Encoder-Decoder Attention Mechanism: Allows each position in the decoder to attend to all positions in the input sequence, combining information from both the encoder and the current state of the decoder.
- Feed-Forward Neural Network: Similar to the encoder, each position in the sequence is processed through the same feed-forward network.

4. Output

Linear and Softmax Layers: The decoder's final output is passed through a linear layer followed by a softmax function to produce a probability distribution over the vocabulary for each position.



Self Attention Mechanism

The self-attention mechanism is a pivotal component in transformer-based architectures, widely used in natural language processing (NLP) tasks like machine translation, text generation, and summarization. It fundamentally enhances the model's ability to capture relationships between words in a sequence, enabling it to understand context and dependencies more effectively than previous architectures like recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

At its core, self-attention allows the model to weigh the significance of each word in relation to every other word in the input sequence. This is achieved through a mechanism where the model computes attention scores that indicate how much focus each word should receive when processing the sequence. These attention scores are derived from comparing each word to every other word in the sequence, learning which words are most relevant for understanding the meaning of a given input.

3. FUNCTIONALITY OVERVIEW

Text Summarization

Models Used: PEGASUS and NLP with spaCy
Description: The text summarization functionality includes both abstractive and extractive summarization. The PEGASUS model is used for abstractive summarization, while spaCy is used for extractive summarization.
Abstractive Summarization: Generates concise summaries by creating new sentences that capture the essence of the original text.
Extractive Summarization: Identifies and extracts key sentences from the original text using NLP techniques with spaCy.
How It Works: Users input text, and based on the chosen summarization type (abstractive or extractive), the appropriate model processes the input to generate a summary.

Text Generation

Model Used: Gemini-pro
Description: The text generation functionality leverages the Gemini-pro model to produce high-quality content based on given prompts. This can be used for generating articles, creative writing, and more.
How It Works: Users provide a prompt or a question, and the Gemini-pro model generates relevant and coherent text in response, drawing on its extensive training data.

Medical Advice

Model Used: Gemini-pro
Description: This functionality provides users with medical advice based on their input queries. It aims to deliver accurate and helpful information in response to health-related questions.
How It Works: Users ask a medical question, and the Gemini-pro model processes the query to generate a well-informed answer based on medical knowledge and data.

Image Question Answering

Model Used: Gemini-pro
Description: This feature allows users to upload images and ask questions about the content of those images. The model analyzes the image and provides relevant answers.
How It Works: Users upload an image and ask a question related to it. The Gemini-pro model analyzes the image to understand its content and generates an appropriate response.

PDF Question Answering

Model Used: Gemini-pro
Description: This functionality enables users to upload PDF documents and ask questions about their content. The model reads and interprets the PDF to provide accurate answers.
How It Works: Users upload a PDF and pose a question about its content. The Gemini-pro model processes the document, extracts the necessary information, and generates a response.

4. CLIENT INTERFACE SYPNOSIS

GENAI TOOLS Page

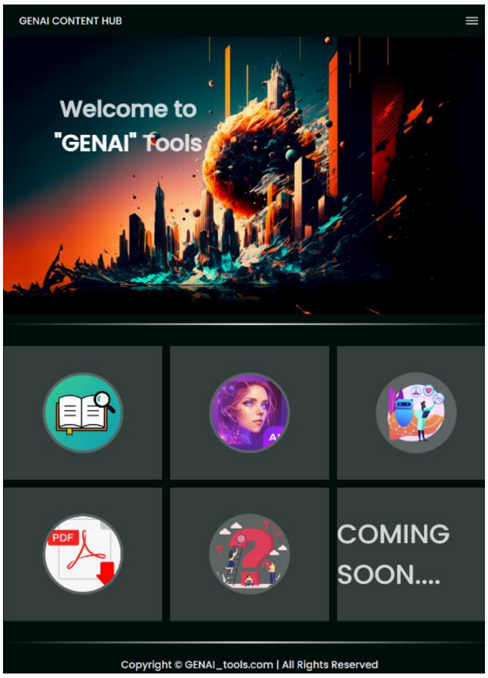
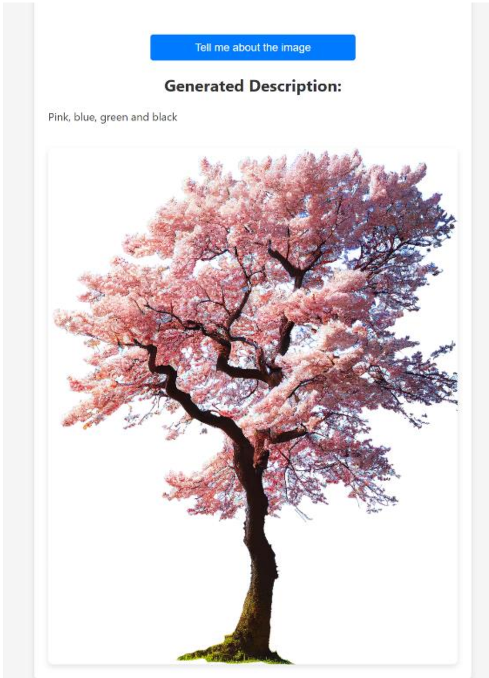
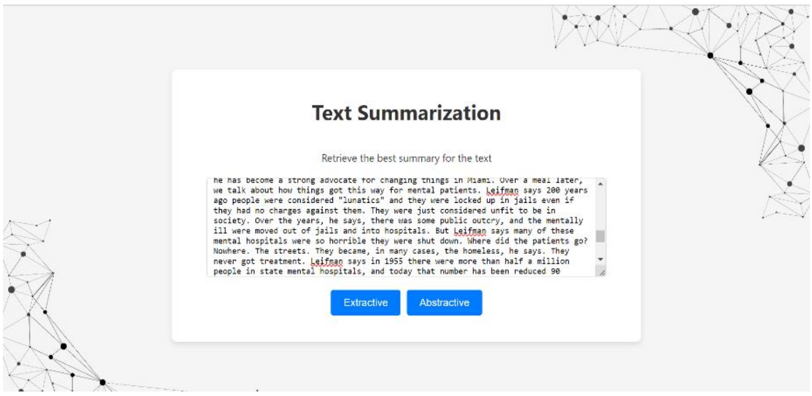
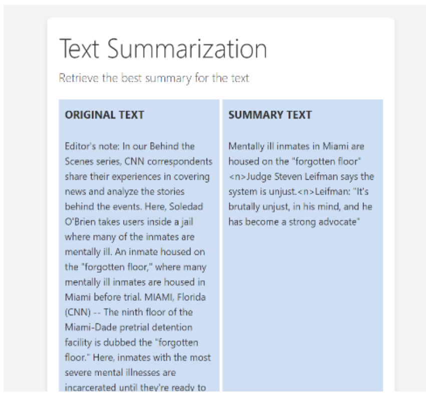


Image Question Answering

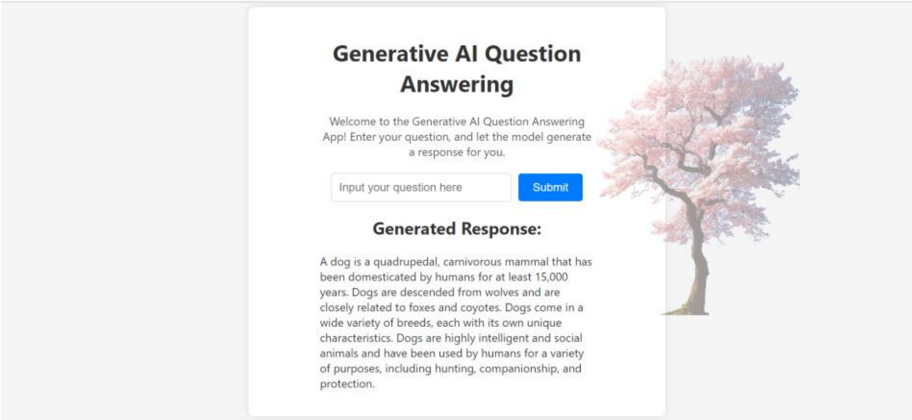


Text Summarization



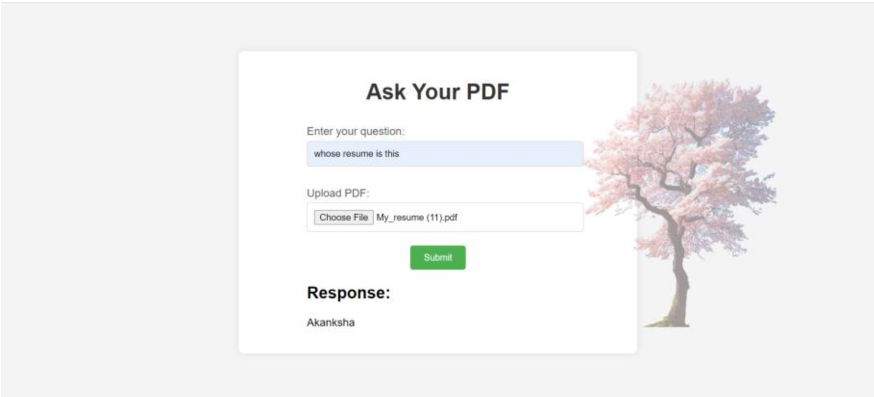


Text Generation



Medical Advice

PDF Question Answering



5. CODING SYPNOSIS

Code for Image question answering tool

```python
# textgen.py
from dotenv import load_dotenv
from PIL import Image
import google.generativeai as genai
import os


load_dotenv()


genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
model = genai.GenerativeModel('gemini-pro-vision')

# Function to generate content
def gen(input_text, image_path):
    image = Image.open(image_path)
    if input_text:
        response = model.generate_content([image, input_text])
    else:
        response = model.generate_content(image)
    return response.text
```

Code for Abstractive Summarization

```python
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM


def summarizer2(rawtext):
    # Load PEGASUS model and tokenizer
    try:
        tokenizer = AutoTokenizer.from_pretrained('models/pegasus_tokenizer')
        model = AutoModelForSeq2SeqLM.from_pretrained('models/pegasus_model')

    except:
        model_name = 'google/pegasus-cnn_dailymail'
        tokenizer = AutoTokenizer.from_pretrained(model_name)
        model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

        # Save the model and tokenizer
        tokenizer.save_pretrained('models/pegasus_tokenizer')
        model.save_pretrained('models/pegasus_model')
    # Tokenize and encode data
    inputs = tokenizer.encode(rawtext, return_tensors='pt', max_length=1024, truncation=True)
    summary_ids = model.generate(inputs, num_beams=4, max_length=150, early_stopping=True)
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    return summary,rawtext,len(rawtext.split()),len(summary.split())
```

Code for PDF Question Answering

```python
import os
from PyPDF2 import PdfReader
from langchain.text_splitter import CharacterTextSplitter
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import google.generativeai as genai
from dotenv import load_dotenv

load_dotenv()

# Configure GenAI with  API key
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))


def get_gemini_pro_response(prompt):
    model = genai.GenerativeModel('gemini-pro')
    response = model.generate_content(prompt)
    return response.text


def process_pdf(file, query, model_path):
    if file:
        pdf_reader = PdfReader(file)
```

```
    text = ""
    for page in pdf_reader.pages:
        text += page.extract_text()


    text_splitter = CharacterTextSplitter(
        separator="\n",
        chunk_size=1000,
        chunk_overlap=200,
        length_function=len
    )
    chunks = text_splitter.split_text(text)
    # Load SentenceTransformer model -used to encode sentences into numerical embeddings
    model = SentenceTransformer(model_path)
    # Encode chunks
    embeddings = model.encode(chunks)
    # Encode query
    query_embedding = model.encode([query])
    similarities = cosine_similarity(query_embedding, embeddings)
    top_indices = similarities.argsort(axis=1).flatten()[-5:][::-1]
    retrieved_chunks = [chunks[i] for i in top_indices]
    # Prepare context for Gemini Pro model
    context = " ".join(retrieved_chunks)
    prompt = f"""Based on the following context from the PDF, please answer the question:

        Context: {context}

        Question: {query}

        Answer the question accurately and concisely."""


        gemini_response = get_gemini_pro_response(prompt)

        return gemini_response

    return None
```

Code for Extractive Summarization

```
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
from heapq import nlargest


def summarizer(rawdocs):
    stopwords=list(STOP_WORDS)
    #print(stopwords)

    nlp=spacy.load('en_core_web_sm')
    doc=nlp(rawdocs)

    tokens = [token.text for token in doc]
    #print(tokens)
    word_freq={}
    for word in doc:
        if word.text.lower() not in stopwords and word.text.lower() not in punctuation:
            if word.text not in word_freq.keys():
                word_freq[word.text]=1
            else:
                word_freq[word.text]+=1


    max_freq=max(word_freq.values())

    for word in word_freq.keys():
        word_freq[word]=word_freq[word]/max_freq


    sent_tokens=[sent for sent in doc.sents]
    #print(sent_tokens)
    sent_scores={}
    for sent in sent_tokens:
        for word in sent:
            if word.text in word_freq.keys():
                if sent not in sent_scores.keys():
                    sent_scores[sent]=word_freq[word.text]
                else:
                    sent_scores[sent]+=word_freq[word.text]
    select_len=int(len(sent_tokens)*0.3)
    summary=nlargest(select_len,sent_scores,key=sent_scores.get)
    final_summary=[word.text for word in summary]
    summary=' '.join(final_summary)
    return summary, doc, len(rawdocs.split(' ')), len(summary.split(' '))
```

6. RESULT AND CONCLUSION

The AI Content Hub project illustrates the powerful capabilities of modern AI models in various domains, including text summarization, content generation, medical advice, and visual question answering. The integration of these functionalities into a single web application offers users a versatile tool for different tasks.

Key Takeaways:

- Effectiveness of AI Models: The project highlights the effectiveness of models like PEGASUS and Gemini-pro in producing high-quality outputs for different types of input data.
- Versatility: The diverse range of functionalities demonstrates the versatility of AI models in addressing various use cases, from text processing to image and PDF analysis.
- User-Friendly Interface: The web application provides an intuitive and user-friendly interface, making advanced AI capabilities accessible to a broader audience.

Future Work:

- Model Improvements: Further training and fine-tuning of the models could enhance their performance, especially in domain-specific tasks.
- Feature Expansion: Additional features, such as voice input for queries and real-time processing, could be integrated to make the application more comprehensive.
- User Feedback: Incorporating user feedback mechanisms would help in continuously improving the system's accuracy and usability.

Overall, the AI Content Hub project serves as a robust platform showcasing the potential of AI in transforming various aspects of information processing and content generation.

7. REFERENCES

[1]. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention Is All You Need.(Nips), 2017. *arXiv preprint arXiv:1706.03762*, *10*, S0140525X16001837.

[2]. Lee, G. G., Latif, E., Shi, L., & Zhai, X. (2023). Gemini pro defeated by gpt-4v: Evidence from education. *arXiv preprint arXiv:2401.08660*.

[3]. García-Peñalvo, F., & Vázquez-Ingelmo, A. (2023). What do we mean by GenAI? A systematic mapping of the evolution, trends, and techniques involved in Generative AI.

[4]. Zhang, J., Zhao, Y., Saleh, M., & Liu, P. (2020, November). Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International conference on machine learning* (pp. 11328-11339). PMLR.

[5]. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020, October). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (pp. 38-45).

[6]. Giarelis, N., Mastrokostas, C., & Karacapilidis, N. (2023). Abstractive vs. extractive summarization: An experimental review. *Applied Sciences*, *13*(13), 7620.

[7]. Cheung, J. C. (2008). Comparing abstractive and extractive summarization of evaluative text: controversiality and content selection. *B. Sc.(Hons.) Thesis in the Department of Computer Science of the Faculty of Science, UnSiversity of British Columbia*, *47*.