

**SINGLE CLICK MULTIPLE SOCIAL MEDIA POSTING
WITH CUSTOMIZED COPYWRITING EACH**

**A PROJECT REPORT SUBMITTED TO
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
AWARD OF THE DEGREE OF
MASTER OF SCIENCE IN APPLIED DATA SCIENCE**

**BY
CAIN ANTONY. T
REG NO. RA2332014010088**

**UNDER THE GUIDANCE OF
Dr.A.THILAKA M.Sc., P.hD.,**



**DEPARTMENT OF COMPUTER APPLICATIONS
FACULTY OF SCIENCE AND HUMANITIES
SRM INSTITUTE OF SCIENCE & TECHNOLOGY**

Kattankulathur – 603 203

Chennai, Tamilnadu

APRIL 2025

BONAFIDE CERTIFICATE

This is to certify that the project report titled “**SINGLE CLICK MULTIPLE SOCIAL MEDIA POSTING WITH CUSTOMIZED COPYWRITING EACH**” is a Bonafide work carried out by **CAIN ANTONY. T (RA2332014010088)**, under my supervision for the award of the Degree of Master of Science – Applied Data Science. To my knowledge the work reported herein is the original work done by these students.

Dr. A.Thilaka
Assistant Professor,
Department of Computer Applications
(GUIDE)

Dr.R.Jayashree
Associate Professor & Head,
Department of Computer Applications

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION OF ASSOCIATION OF RESEARCH PROJECT WITH SUSTAINABLE DEVELOPMENT GOALS

This is to be certify that the research project entitled “**SINGLE CLICK
MULTIPLE SOCIAL MEDIA POSTING WITH CUSTOMIZED
COPYWRITING EACH**” carried out by **Mr. Cain Antony T** under the supervision of **Dr. A. Thilaka** in partial fulfilment of the requirement for the award of Post-Graduation program has been significantly or potentially associated with SDG Goal No **09 (NINE)** titled **INDUSTRY, INNOVATION, AND INFRASTRUCTURE**

This study has clearly shown the extent to which its goals and objectives have been met in terms of filling the research gaps, identifying needs, resolving problems, and developing innovative solutions locally for achieving the above-mentioned SDG on a National and/or on an international level.

SIGNATURE OF THE STUDENT

GUIDE

HEAD OF THE DEPARTMENT

ACKNOWLEDGEMENT

With profound gratitude to the ALMIGHTY, I take this chance to thank the people who helped me to complete this project.

I take this as a right opportunity to say “THANKS” to my parents who are there to stand with me always with the words “YOU CAN”.

I wish to express my sincere gratitude to **Dr. T. R. Paarivendhar**, Chancellor, and **Prof. A. Vinay Kumar**, Pro Vice-Chancellor (SBL), SRM Institute of Science & Technology, who gave us the platform to establish me to reach greater heights.

I earnestly thank **Dr. A. Duraisamy**, Dean, Faculty of Science and Humanities, SRM Institute of Science & Technology, who always encourage us to do novel things.

A great note of gratitude to **Dr. S. Albert Antony Raj**, Deputy Dean, Faculty of Science and Humanities for his valuable guidance and constant support to do this Project.

We express our sincere thanks to **Dr. R. Jayashree**, Associate Professor & Head, for her support to execute all incline in learning.

It is our delight to thank our project guide **Dr. A. Thilaka**, Assistant Professor, Department of Computer Applications, for his help, support, encouragement, suggestions and guidance throughout the development phase of the project.

We convey our gratitude to all the faculty members of the department who extended their support through valuable comments and suggestions during the reviews.

Our gratitude to friends and people who are known and unknown to me who helped in carrying out this project work a successful one.

CAIN ANTONY. T

COMPANY COMPLETION CERTIFICATE



PLAGIARISM CERTIFICATE

Thilaka A

Cain Antony RA2332014010088 PROJECT TITLE & ABSTRACT.docx

aaaa
MCA
SRM Institute of Science & Technology

Document Details

Submission ID
trn:oid::1:3201414737

Submission Date
Apr 1, 2025, 10:55 AM GMT+5:30

Download Date
Apr 1, 2025, 10:55 AM GMT+5:30

File Name
Cain_Antony_RA2332014010088_PROJECT_TITLE_ABSTRACT.docx

File Size
17.7 KB

1 Page
238 Words
1,452 Characters

0% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- 0 Not Cited or Quoted 0%
Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
Matches that are still very similar to source material
- 0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

- 0% Internet sources
- 0% Publications
- 0% Submitted works (Student Papers)

TABLE OF CONTENTS

S. NO	TITLE	PAGE NO.
1.	INTRODUCTION	1
2.	SOFTWARE REQUIREMENT ANALYSIS	3
3.	SYSTEM ANALYSIS	17
4.	MODEL DESIGN	23
5.	SYSTEM IMPLEMENTATION	27
6.	TESTING	38
7.	OUTPUT SCREEN	39
8.	CONCLUSION	43
9.	FUTURE ENHANCEMENTS	45
10.	REFERENCES	45

LIST OF FIGURES

S.NO	TITLE	PAGE NO.
1.	Single Click Multiple Social Media Posting with Customized Copywriting Each	26
2.	VS Code Terminal with Implementation	39
3.	AI Copywriting Agent (Status: Stopped)	39
4.	AI Copywriting Agent (Status: Running)	41
5.	LinkedIn and Tumblr with Copywrite Content	42

ABSTRACT

In the present days, handling more social media platforms effectively is a hard-breaking task in terms of business and for the users too. The project titled "Single Click Multiple Social Media Posting with Customized Copywriting for Each" aims to streamline the process of creating, customizing, and posting content across various social media channels through a unified system. This platform integrates advanced AI capabilities for generating platform-specific content tailored to the unique tone, style, and requirements of each social media site, such as Twitter, Facebook, Instagram, and LinkedIn. By leveraging AI models like OpenAI's GPT, the system ensures that the content aligns with branding guidelines, campaign goals, and audience preferences. Additionally, it incorporates features like Google Sign-In for secure user authentication, a centralized API hub for social media integration, and customizable content-generation agents that cater to diverse tones, keywords, and call-to-actions. The backend handles the heavy lifting of AI content generation and API calls, while the intuitive frontend provides users with an easy-to-navigate interface for managing their campaigns. The project architecture ensures scalability and security, allowing users to save time and maintain a consistent brand voice across platforms. This project not only addresses the inefficiencies of manual social media management but also empowers users to enhance their digital presence with minimal effort, making it a valuable tool for businesses, influencers, and marketers in the digital era.

1. INTRODUCTION

“Single Click Multiple Social Media Posting with Customized Copywriting Each” is a model created to optimize the time of a user to post and write a relative content in social media platforms that are accessible by the user. In this Single Click Multiple social media Posting with Customized Copywriting Each, the user can only utilize the model for the following social medias.

1. LinkedIn
2. Tumblr

There is a plan in executing the model in Instagram, LinkedIn, Twitter X, and Facebook. Since Instagram comes under Facebook and due to lack of usage in Facebook, it finds difficult in accessing their API keys and secrets using their developer platforms. In Twitter X, accessing API key and secret is cost consuming. LinkedIn and Tumblr were the final opinion in accessing their API keys and secrets from their developer platforms. An AI Copywriting Agent is created using HTML to manage and control the process of image recognition of a picture yet to be posted and auto-generation of copywrite content using Python.

1.1 Why AI Copywriting Agent?

The AI Copywriting Agent is created to handle the process of image recognition, label identification from backend, and auto-generation of copywrite content created based on the identified label. The image is clicked twice to screenshot the clicked area to recognize both the image and the label from it. The story-feed section is clicked thrice to write the copywrite content based on the identified label using AI.

1.2 Key Reasons for Using AI Copywriting Agent

1. The model is well-controlled with AI features
2. The copywrite content is auto-generated based on the label identified during the image recognition process where the mouse pointer moves as per the automation
3. The process of image recognition, label identification from backend, and auto-generation of copywrite content were managed along with the content length fixed in the backend python code to avoid repeating of the content.

1.3 Background

1. To optimize the time to post and write content
2. To manage the auto-generation of copywrite content
3. To control the process of image recognition and auto-generation of copywrite content
4. To avoid copyrights issues in posting
5. To avoid any unauthorized activities
6. To increase the creativity in content writing

1.4 Scope

Among other **Social Medias**, **LinkedIn** can be used to share or post informative to the people like ideas, thoughts, learnings through photos/images, and videos. Likewise, **Tumblr** can be used to share or post interesting things to the people that are both entertaining and informative through photos/images, and videos.

In **Online Platforms** **GitHub** is used to create and name a new repository to upload the frontend and backend codes. **Netlify** is used to create a new webpage using log-in through GitHub repository, fill the variables with the suitable values and check if the webpage is working, and if not, refine the backend code to make the webpage functionable. **Hugging Face** is used to collect machine learning models like ViT, GPT-2, and CLIP, import these models in the backend code to begin the process of image recognition, label identification from backend, and auto-generation of copywrite content.

For **User Authorization**, **Google Cloud Credentials**, is used to insert the website URL created using Netlify, mention the type of business such as app, etc., to access the OAuth 2.0 client ID and API key. **Developer Platform of LinkedIn and Tumblr** is used

to create an app to access the API keys and secrets of LinkedIn and Tumblr. The user can get it's **Expected Outcomes** as an **Own Content Creation** where the copywrite content that is auto-generated is like an own story created using AI, **Process Control** where the AI Copywriting Agent controls the process of image recognition, label identification from backend, and auto-generation of label-based copywrite content by clicking the start and stop button indicating the ongoing process and the termination, and **Hacker-like Process**: The auto-generation of the copywrite content is presented as like a hacker accessing another computer from his system.

2. SOFTWARE REQUIREMENT ANALYSIS

2.1 Hardware Requirements

Processor	Minimum x64: 2.4 GHz
RAM	8 GB Recommended
Hard disk	256 GB
Web Server	IIS

2.2 Software Requirements

Operating System	Windows 11 Home / Professional
Tools	Python, HTML, JavaScript
IDE	Visual Studio Code
Browser	Google Chrome

2.3 SOFTWARE SPECIFICATION

Windows 11 is the **Operating System** used to execute the **Programing Languages**: Python, HTML, JavaScript.

In **Libraries and Frameworks** **Pyautogui** is used for Automation of GUI tasks like clicking and typing. **Pynput.mouse** is used for Listening and responding to mouse events (clicks). **Transformers (from Hugging Face)** is used for Loading and using pre-trained AI models (CLIP for vision + GPT-2 for text). **PIL (Python Image Library, via Pillow)** is used for Basic image handling. **Torch (pytorch)** is used for Running and managing deep learning models. **Win32gui (from pywin32)** is used for Interacting with the Windows OS windowing system. **Threading** is used for Running the mouse event listener in the background. **Tkinter** is used for Building a basic graphical user interface (GUI). **Time** is used for Timing click intervals and simulating delays. Visual Studio Code is the **Development Environment via., IDE**.

2.4 ABOUT THE SOFTWARE AND ITS FEATURES

2.4.1 SOFT ENVIRONMENT

Visual Studio Code

Visual Studio Code (VS Code) is a powerful, open-source code editor developed by Microsoft that has become popular among developers for its versatility and ease of use. It supports a wide range of programming languages and frameworks, offering features such as intelligent code completion, debugging, syntax highlighting, and integrated version control. VS Code is highly customizable, allowing users to extend its functionality through a vast library of extensions, themes, and integrations. With a user-friendly interface and built-in support for Git, it is an excellent tool for both beginner and experienced developers working on various types of software projects.

Features of Visual Studio Code

1. **Intelligent Code Completion:** Provides context-aware code suggestions and autocompletion using IntelliSense for various programming languages.
2. **Built-in Debugging:** Allows developers to debug code directly within the editor, with support for breakpoints, call stacks, and an interactive console.
3. **Integrated Version Control:** Comes with built-in Git support, enabling users to commit changes, review diffs, and manage repositories without leaving the editor.
4. **Extensibility:** Supports a vast library of extensions to add functionalities such as language support, themes, code linters, and tools for frameworks and libraries.
5. **Syntax Highlighting:** Highlights code syntax for easier readability across numerous programming languages.
6. **Integrated Terminal:** Features an embedded terminal for running command-line operations within the editor.
7. **Multi-Language Support:** Supports a broad range of languages out of the box, including JavaScript, Python, C++, Java, and more.
8. **Customizability:** Allows users to personalize their coding environment with custom keybindings, themes, and settings.

9. **Live Share:** Facilitates real-time collaboration by enabling multiple developers to work on the same codebase simultaneously.
10. **Code Navigation and Refactoring:** Provides easy navigation tools like "Go to Definition," "Peek Definition," and powerful refactoring options.
11. **Code Snippets:** Supports customizable code snippets to speed up repetitive coding tasks.
12. **Emmet Abbreviations:** Boosts HTML and CSS coding efficiency with Emmet's shorthand abbreviations.

2.4.2 ESSENTIAL LIBRARIES AND FUNCTIONS

1. **Pyautogui** Simulates keyboard and mouse actions. It Automates mouse movement, and scrolling, Types out strings with delays to simulate human typing, Captures screenshots of the screen or a region, and Cross-platform (works on Windows, macOS, Linux).
2. **Pynput.mouse** Listens to mouse events such as clicks and movement. It Detects left/right click, double-click, triple-click, can track mouse position and actions, and Works asynchronously via threading.
3. **Transformers (from Hugging Face)** uses CLIPModel, CLIPProcessor for image and text similarity, and GPT2Tokenizer, GPT2LMHeadModel for story generation. It can access to powerful pre-trained models (CLIP, GPT-2, etc.), easy pipeline for multimodal (text + image) tasks, and hugely popular in AI/NLP/vision communities.
4. **PIL (Python Image Library, via Pillow)** is used for Image processing and manipulation. It can perform functions like open, resize, crop, convert images, and easy integration with other AI libraries like transformers.
5. **Torch (pytorch)** is used for the deep learning framework used by transformers. It is Used to run model inference (no training in your case). Handles tensors and computation on CPU/GPU. Integrates tightly with Hugging Face models.
6. **Win32gui (from pywin32)** Interacts with Windows GUI applications. It gets the title of the currently active window. It helps to identify which app is currently focused.
7. **Threading** Runs code in the background without blocking the main GUI. It allows parallel execution (mouse listener runs while GUI is active), and helps to prevent GUI from freezing.

8. **Tkinter** consist of GUI library that is built into Python. It is the simple way to create windows, buttons, labels, useful for desktop apps with minimal dependencies, and too native and lightweight.
9. **Time** Deals with timestamps and delays. It is a Time-based operations like delays time calculations and useful for debouncing clicks or mimicking human-like typing.

2.4.3 DATA HANDLING AND PREPROCESSING

1. Types of Data Involved

Your application handles two main types of data like **Image Data** with screenshots captured from the screen when the user double-clicks, and **Text Data** with labels predicted by CLIP and stories generated by GPT-2.

2. Image Handling and Preprocessing

Screenshot Capture

The screenshot functionality is initiated using the line `pyautogui.screenshot(screenshot_path, region=(x - 100, y - 100, 200, 200))`, which is typically triggered during a double-click event. This command captures a specific 200x200 pixel region around the cursor, with an offset of 100 pixels to the left and above the current (x, y) mouse coordinates. The resulting image is saved to the specified *screenshot_path* and is used as input for further image processing and classification tasks.

Preprocessing for CLIP

Before feeding the captured screenshot into the CLIP model, it undergoes a preprocessing step defined in the function `preprocess_image(image_path)`. The function loads the image using the PIL library, resizes it to 224x224 pixels, and converts it to the RGB color mode. This resizing matches the input dimension requirements of the CLIP model, while the RGB conversion ensures compatibility with the model's expectations. The function returns the processed PIL image, which is ready to be passed into the CLIP processor

CLIP Text + Image Encoding

To prepare the image and accompanying text for the CLIP model, both are passed to the CLIP processor using the line `inputs = processor(text=text_labels, images=image, return_tensors="pt", padding=True)`. Here, `text_labels` is a predefined list of textual category labels, while `image` is the preprocessed screenshot. The processor converts both modalities into tensor representations that the CLIP model can understand. Padding is applied to ensure that all inputs in the batch have uniform dimensions, allowing for batch processing without errors.

CLIP Prediction

After encoding, the model generates a set of logits representing the similarity between the image and each textual label. These logits are accessed using `logits_per_image = outputs.logits_per_image`. To interpret these logits as probabilities, a softmax function is applied using `probs = logits_per_image.softmax(dim=1)`. The model then identifies the index of the highest probability using `predicted_label_idx = probs.argmax().item()`. This index corresponds to the label that the model predicts as the most relevant match for the given image, based on its learned similarity between visual and textual features.

2. Text Data Handling and Preprocessing

Once the image is processed and passed through CLIP, the next phase involves working with textual data derived from the predicted label. This label—typically a short phrase like “a child” or “a bird”—serves as the seed input for GPT-2, which will generate a relevant and creative continuation. Handling this text requires converting it into a format that GPT-2 can process, beginning with tokenization.

3. Recognized Label (from CLIP) → Used as input for GPT-2

The predicted label output from the CLIP model is first tokenized using the GPT-2 tokenizer. This is done through the line `encoded_tokens = gpt2_tokenizer.encode(label, add_special_tokens=True)`, which converts the text into a series of numerical token IDs. These tokens are then wrapped in a tensor using `input_ids =`

`torch.tensor([encoded_tokens])`, making them ready to be fed into the GPT-2 model. This step ensures the textual input is in the correct format for GPT-2 to process.

4. GPT-2 Story Generation

With the input tensor prepared, the GPT-2 model generates a creative text output using the `generate()` function. Several parameters are tuned to influence the generation process: `max_length=50` limits the length of the story, `num_return_sequences=1` restricts output to a single story, `no_repeat_ngram_size=2` helps avoid repetitive phrases, `top_p=0.9` focuses sampling on the top 90% of likely outcomes, and `temperature=0.7` controls randomness. Together, these settings help strike a balance between creativity and coherence in the generated story.

5. Decoding

After generation, the model returns output in the form of token IDs. These are then converted back to human-readable text using `gpt2_tokenizer.decode(output[0], skip_special_tokens=True)`. This step strips out special tokens and reveals the final output—a short story or description inspired by the original image label. The decoded text can then be displayed, stored, or used further.

6. Data Flow Summary

The complete data flow of the system starts from a simple mouse event. A double-click by the user triggers a screenshot capture using PyAutoGUI. The image is resized and preprocessed with PIL, then fed into CLIP along with a set of textual labels. CLIP processes the visual and textual data and returns the most relevant label. This label is tokenized and passed into GPT-2, which generates a short narrative. Finally, the output can be displayed or even auto-typed into social platforms like LinkedIn or Tumblr, creating a seamless pipeline from visual input to creative textual output.

2.4.4 MODEL BUILDING AND EVALUATION

1. CLIP (Contrastive Language–Image Pretraining)

The model used for image recognition in this pipeline is `openai/clip-vit-large-patch14`, a powerful vision-language model capable of matching visual input to a predefined set of textual labels such as “a man,” “a bird,” and similar descriptive phrases. To interface with the model, the `CLIPProcessor` is employed to convert both the image and the candidate text labels into a format suitable for CLIP’s internal computations. The workflow begins by capturing a screenshot centered around the mouse click area. This image is then resized and preprocessed to meet the model’s input requirements. The processed image, along with the list of textual labels, is passed through the CLIP model. The model computes similarity scores between the image and each text label, which are accessed through `logits_per_image`. These logits are then transformed into probabilities using the softmax function. The label with the highest probability is selected as the most likely match for the image. The accuracy and performance of the model are evaluated based on how well it identifies the visual content. This is manually verified by printing the predicted label using `print(f"Recognized Label: {label}")`. If the label makes visual sense in context, it indicates that CLIP is functioning effectively.

2. GPT-2 (Generative Pre-trained Transformer 2)

The model used for text generation in this workflow is GPT-2, which is responsible for producing a short story or creative copy based on the label recognized by the CLIP model. Before being passed to GPT-2, the label—such as “a bird”—is first tokenized using the GPT-2 tokenizer, converting the text into a format the model can understand. The tokenized input is then used with GPT-2’s `generate()` method to produce a continuation, guided by parameters like `max_length` to limit story length, `top_p` to ensure diversity, `temperature` to control creativity, and `no_repeat_ngram_size` to reduce repetition. Once the model generates the output in the form of token IDs, it is decoded back into readable text using the tokenizer. The effectiveness of GPT-2 is evaluated based on the relevance and fluency of the generated story, which is printed and optionally auto-typed using `print(f"Generated Story:\n{story}")`. If the story aligns well with the original label and reads naturally, GPT-2 is considered to have performed successfully.

3. **Sample Code for Model Evaluation:**

```
import pyautogui

from pynput import mouse

from transformers import CLIPProcessor, CLIPModel, GPT2Tokenizer,
GPT2LMHeadModel

from PIL import Image

import torch

import time

import win32gui # For detecting active window (Windows-specific)


# CLIP model and processor setup

model_name = "openai/clip-vit-large-patch14"

model = CLIPModel.from_pretrained(model_name)

processor = CLIPProcessor.from_pretrained(model_name)


# GPT-2 model and tokenizer setup

gpt2_tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

gpt2_model = GPT2LMHeadModel.from_pretrained("gpt2")


# Predefined labels for recognition

text_labels = ["a human", "a man", "a woman", "a child", "an animal", "a tree",
"a banner", "a bird"]


# Global variable to store the last recognized label

last_recognized_label = None
```

```

linkedin_done = False # Track if LinkedIn post has been made

# Function to preprocess the image

def preprocess_image(image_path):

    image = Image.open(image_path).resize((224, 224)).convert("RGB")

    return image

# Function to recognize the image

def recognize_image(image_path):

    image = preprocess_image(image_path)

    inputs = processor(text=text_labels, images=image, return_tensors="pt",
padding=True)

    with torch.no_grad():

        outputs = model(**inputs)

        probs = outputs.logits_per_image.softmax(dim=1)

        return text_labels[probs.argmax().item()]

# Function to generate a story based on the recognized label

def generate_story(label):

    encoded_tokens = gpt2_tokenizer.encode(label, add_special_tokens=True)

    input_ids = torch.tensor([encoded_tokens])

    output = gpt2_model.generate(

        input_ids, max_length=50, num_return_sequences=1,
no_repeat_ngram_size=2, top_p=0.9, temperature=0.7

    )

```

```

    return gpt2_tokenizer.decode(output[0], skip_special_tokens=True)

# Check if the active window is LinkedIn
def is_active_window_linkedin():
    active_window = win32gui.GetForegroundWindow()
    window_title = win32gui.GetWindowText(active_window)
    return "linkedin" in window_title.lower()

# Function to type the recognized label into LinkedIn
def type_recognized_label_linkedin(label):
    global linkedin_done

    if is_active_window_linkedin():
        print(f"Typing the recognized label on LinkedIn: {label}")
        pyautogui.click(x=300, y=800) # Adjust coordinates to LinkedIn feed input
        box

        time.sleep(0.5)

        for char in label:
            pyautogui.typewrite(char)
            time.sleep(0.1)

        linkedin_done = True # Mark LinkedIn task as completed

# Check if the active window is Tumblr
def is_active_window_tumblr():
    active_window = win32gui.GetForegroundWindow()
    window_title = win32gui.GetWindowText(active_window)

```

```

    return "tumblr" in window_title.lower()

# Function to type the recognized label into Tumblr

def type_recognized_label_tumblr(label):

    if linkedin_done and is_active_window_tumblr(): # Ensure LinkedIn post is
done before Tumblr

        print(f"Typing the recognized label on Tumblr: {label}")

        pyautogui.click(x=400, y=700) # Adjust coordinates to Tumblr feed input
box

        time.sleep(0.5)

        for char in label:

            pyautogui.typewrite(char)

            time.sleep(0.1)


# Mouse event handler

def on_click(x, y, button, pressed):

    global last_recognized_label, linkedin_done

    if pressed:

        current_time = time.time()

        if on_click.last_click_time:

            time_since_last_click = current_time - on_click.last_click_time

            on_click.click_count += 1


# Double-click: Image recognition

```

```

        if on_click.click_count == 2 and time_since_last_click <= 5:

            if is_active_window_linkedin() or (linkedin_done and
is_active_window_tumblr()):

                screenshot_path = "screenshot.png"

                pyautogui.screenshot(screenshot_path, region=(x - 100, y - 100, 200,
200))

                print("Captured a screenshot near the click position.")

                last_recognized_label = recognize_image(screenshot_path)

                print(f"Recognized Label: {last_recognized_label}")

                # Generate and display the story

                story = generate_story(last_recognized_label)

                print(f"Generated Story:\n{story}")

            on_click.click_count = 0 # Reset click count

# Triple-click: Type recognized label

elif on_click.click_count == 3 and time_since_last_click <= 5:

    if is_active_window_linkedin() and last_recognized_label:

        type_recognized_label_linkedin(last_recognized_label)

        elif linkedin_done and is_active_window_tumblr() and
last_recognized_label:

            type_recognized_label_tumblr(last_recognized_label)

    on_click.click_count = 0 # Reset click count

```


else:

on_click.last_click_time = current_time

on_click.click_count = 1

Initialize click tracking

on_click.last_click_time = None

on_click.click_count = 0

Start mouse listener

print("Listening for mouse double-clicks and triple-clicks on LinkedIn or Tumblr feed...")

with mouse.Listener(on_click=on_click) as listener:

listener.join()

3. SYSTEM ANALYSIS

3.1 System Overview

3.1.1 Purpose

The primary objective of the system is to automate image-based content generation and streamline posting workflows on platforms such as LinkedIn and Tumblr. By leveraging AI models like CLIP and GPT-2, the system is capable of recognizing objects within screenshots and generating contextually relevant short stories or copywriting. This reduces the need for manual content creation and allows users to engage audiences with minimal effort.

3.1.2 Core Functionalities

The system incorporates several core features to enhance usability and automation. First, it supports mouse-based interaction, where double or triple mouse clicks act as triggers to initiate specific tasks such as capturing screenshots. Second, it utilizes a combination of CLIP for image recognition and GPT-2 for storytelling, effectively turning visual input into compelling textual output. Third, it includes a platform detection mechanism that allows for automatic input and posting tailored to the detected platform (e.g., LinkedIn or Tumblr). Lastly, the system features a simple graphical user interface (GUI) that enables users to easily start or stop the listener service, offering intuitive control over the automation process.

3.2 System Components

3.2.1 AI Models

1. CLIP Model (OpenAI/clip-vit-large-patch14)

The CLIP model, specifically the openai/clip-vit-large-patch14 variant, is employed for visual recognition within this system. It works by comparing a given image against a predefined list of text labels, such as "a dog," "a building," or "a child." Through this comparison, the model identifies which label best matches the visual content of the image. This enables a meaningful understanding of image context, forming the foundation for subsequent text generation.

2. GPT-2 Model

Once a label is recognized by the CLIP model, it is passed into the GPT-2 model to generate a relevant short story or textual description. GPT-2 first tokenizes the input label and then uses its language generation capabilities to create a coherent narrative or snippet. Various parameters such as temperature and max_length are configured during generation to control the creativity, diversity, and length of the output, ensuring the resulting text aligns well with the context of the original image.

3.2.2 Input Devices

1. Mouse Listener (pynput)

The system uses the pynput library to listen for specific mouse events, enabling intuitive user interaction. A double-click acts as a trigger to capture a screenshot and initiate the image recognition and story generation process using CLIP and GPT-2. Additionally, a triple-click is used to automatically type the recognized label or generated story into a social media text field, such as those on LinkedIn or Tumblr. This seamless interaction allows users to create and share content with minimal manual input.

2. Screenshot Capture (pyautogui)

To identify visual content near the user's point of interest, the pyautogui library is used for capturing screenshots. When a double-click is detected, a screenshot is taken around the mouse cursor position, focusing on a 200x200 pixel area. This targeted capture ensures that the image content most relevant to the user's click is used for recognition, making the system both responsive and context-aware.

3.2.3 GUI (tkinter)

The system includes a basic graphical user interface (GUI) that offers essential controls for user interaction and monitoring. Through this interface, users can easily start or stop the mouse listener service, enabling or disabling the automation features as needed. Additionally, the GUI displays the current system status, clearly indicating whether the listener is Running or Stopped. This simple yet functional design ensures that users can manage the system intuitively without needing to interact directly with the code.

3.3 Workflow Logic

The system features an event-driven design that responds to mouse clicks using time-based logic to distinguish between single, double, and triple clicks. This allows for intuitive user input with minimal UI interaction. When a double-click is detected, the system captures a screenshot near the cursor, processes the image using the CLIP model to recognize the primary object, and then uses GPT-2 to generate a relevant short story or description. The generated story is automatically typed into the currently active social media feed, such as LinkedIn or Tumblr. In contrast, a triple-click bypasses the story generation and simply types the recognized label (e.g., “a cat” or “a plane”) directly into the feed. This dual-action setup offers flexibility based on user intent—whether they want just the label or a full narrative.

3.3.2 Window Detection

To ensure that automation actions are executed only on the intended platforms, the system uses active window detection via the win32gui library. This component continuously monitors the title of the currently active window and checks if it contains specific keywords like “LinkedIn” or “Tumblr.” By doing so, the system ensures that the auto-typing of labels or generated stories only occurs when the user is actively focused on the correct platform. This prevents accidental input into unrelated applications and maintains context-aware automation.

3.3.3 Automation with pyautogui

The system incorporates input automation to streamline the content posting process on platforms like LinkedIn and Tumblr. It begins with mouse click simulation, which is used to automatically select the appropriate input area or text field within the active window. Once the input field is selected, the system proceeds with typing simulation, where the generated text—either a recognized label or a short story—is programmatically entered as if typed by the user. This end-to-end automation enables quick and hands-free content creation, enhancing user productivity and reducing repetitive manual tasks.

3.4 Platform Specifics

3.4.1 LinkedIn

When targeting LinkedIn for automated content posting, the system first performs a check to see if the active window title contains the keyword “LinkedIn.” Once confirmed, it proceeds to simulate a mouse click on the post feed area to activate the text input field. The click coordinates for this action can be customized based on screen resolution or layout preferences. After successfully selecting the input field, the system types either the generated story or the recognized label, depending on whether the user triggered the process via a double-click or triple-click. This allows for smooth, context-aware interaction with the LinkedIn interface.

3.4.2 Tumblr

For Tumblr integration, the system first checks whether the active window title includes the keyword “Tumblr.” Once confirmed, it triggers a mouse click to select the post input area, preparing the field for text entry. Just like with LinkedIn, this ensures the typing occurs in the correct location. Depending on the type of click event received—double-click or triple-click—the system will then type either the full generated story or just the recognized label into the input area. This enables quick and efficient content creation tailored specifically for the Tumblr platform.

3.5 User Interface & Control

3.5.1 tkinter GUI

The system provides a simple yet effective visual status indicator that displays whether the listener service is currently running or stopped. This real-time feedback allows users to easily understand the operational state of the automation process. In addition to visual updates, the interface also includes manual control buttons that let users start or stop the mouse listener as needed. This ensures greater control and flexibility, enabling users to activate or deactivate the system at their convenience without relying solely on background processes.

3.5.2 Threaded Mouse Listener

To maintain a smooth and responsive user experience, the mouse listener is designed to run in a separate thread, independent of the main GUI loop. This threading approach ensures that the graphical interface remains fully interactive, allowing users to start, stop, or monitor the system without any lag or freezing. By offloading the listener to a background thread, the application achieves efficient multitasking, seamlessly managing both user input and automation logic.

3.6 Error Handling & Limitations

3.6.1 Hardcoded Coordinates

The system uses predefined mouse click coordinates to select the input fields on platforms like LinkedIn and Tumblr. These locations are fixed and work well under specific screen resolutions and layout settings. However, since screen dimensions and UI designs can vary across devices, users may need to adjust these coordinates to ensure accurate targeting of the post input areas. This flexibility allows the system to be adaptable to different screen setups while maintaining smooth automation performance.

3.6.2 No multi-platform support at once

The system is designed to check only the currently active window when deciding where to post content. As a result, it can automate posting on only one platform at a time—either LinkedIn or Tumblr—based on which window is active during the mouse click event. This limitation means that simultaneous auto-posting across multiple platforms is not supported in the current implementation. However, it ensures focused and accurate input to the intended platform, minimizing the risk of incorrect content placement.

3.6.3 Minimal Validation

Currently, the system does not include a fallback mechanism in cases where screenshot recognition fails or the story generation produces irrelevant or low-quality results. If the CLIP model is unable to accurately identify the content within the captured image, or if GPT-2 generates text that doesn't align well with the recognized label, the system proceeds without verification. This could lead to incorrect or out-of-context posts. Implementing error handling or confidence-based filtering in future versions could help improve reliability and content quality.

3.7 Possible Enhancements

3.7.1 Dynamic Element Detection

A valuable improvement to the current system would be to replace hardcoded coordinates with computer vision techniques for locating input fields. By using computer vision, the system could dynamically detect and identify the correct areas (such as post boxes on LinkedIn or Tumblr) regardless of screen resolution, layout changes, or user interface updates. This would make the automation process far more robust, adaptable, and user-friendly, eliminating the need for manual adjustment of click positions and enhancing overall accuracy in targeting input fields.

3.7.2 Custom Labels

An important feature that could greatly enhance flexibility is the ability to dynamically add custom recognition labels to the system. Currently, the CLIP model relies on a predefined list of text labels for object recognition. By allowing users to modify or expand this list in real-time, they could tailor the recognition process to suit their unique content needs or niche contexts. This would make the system more personalized and versatile, enabling broader application across different domains or creative styles without modifying the core code.

3.7.3 Richer GUI

To improve usability and transparency, the system can be enhanced by adding a log panel within the GUI. This panel would display real-time logs of actions such as mouse events, screenshot captures, recognized labels, and text generation events, helping users monitor the automation process more clearly. Additionally, introducing a platform toggle option in the interface would let users manually switch between LinkedIn and Tumblr, instead of relying solely on active window detection. This adds a layer of control, especially useful in multi-window environments. Another powerful upgrade would be to visually display the recognized images and their corresponding labels live in the GUI, allowing users to instantly verify what was detected and generated before auto-posting. Together, these features would significantly improve both transparency and control for the end user.

4. MODEL DESIGN

4.1 Overview

This tool is a **contextual AI copywriter assistant** that detects images from mouse interactions (like screenshots), recognizes the content using a **CLIP model**, and then generates a **story/text** using **GPT-2**, typing the content directly into platforms like LinkedIn or Tumblr.

4.2 Model Loading & Initialization

1. CLIP Model (Image Recognition)

The system leverages the CLIP model for image recognition, utilizing the CLIPModel and CLIPProcessor from the Hugging Face Transformers library. Specifically, it employs the "openai/clip-vit-large-patch14" variant, which is designed to compare visual inputs with a set of predefined textual labels. By calculating the similarity between the image and each label, the model selects the one that best represents the image content. This enables the system to accurately recognize elements within screenshots captured during user interaction.

2. GPT-2 Model (Story Generation)

For text generation, the system uses the GPT-2 model, integrating GPT2LMHeadModel and GPT2Tokenizer from Hugging Face. The model used is "gpt2", which is well-suited for generating coherent and creative short-form text. Once the CLIP model identifies a label (e.g., "a mountain" or "a dog"), this label is passed to GPT-2, which generates a short story or description related to the recognized content. This combination of visual recognition and natural language generation creates a seamless flow from image to meaningful content.

4.3 Image Recognition Pipeline

1. Screenshot Capture

The process begins with screenshot capture, which is triggered by a mouse double-click event. When this event occurs, the system uses `pyautogui.screenshot()` to take a snapshot of the region surrounding the click point. This ensures that the captured image is contextually relevant to what the user interacted with on the screen.

2. Preprocessing

Following capture, the image undergoes preprocessing to make it compatible with the CLIP model. Specifically, the screenshot is resized to 224x224 pixels and converted to RGB format, as these are the required input dimensions and color format for the CLIP model to function correctly.

3. Text-Image Similarity

Once preprocessed, the image is passed to the CLIP model, which compares it against a list of predefined textual labels such as “a man,” “a bird,” and others. Using the model’s internal similarity scoring, it calculates how closely the image matches each label. The final prediction is made by applying softmax to the logits and selecting the label with the highest probability using argmax, effectively identifying what the image most likely represents.

4.4 Text Generation Pipeline

Once the CLIP model identifies the most relevant label from the screenshot, that label is encoded and passed into the GPT-2 model. GPT-2 then generates a natural language continuation of the label, typically limited to a maximum of 50 tokens. The result is a short, creative story or caption that relates contextually to the recognized object or scene. This generated story is ultimately what the system prepares to automatically type into the user’s social media feed, providing AI-driven content generation.

4.5 Mouse Listener + Events

To capture user interactions, the system employs `pynput.mouse.Listener`, which actively listens for mouse click events. A double-click triggers the full pipeline: screenshot capture, image recognition via CLIP, and story generation via GPT-2. A triple-click, on the other hand, skips text generation and directly types only the recognized label into the post input field. The listener also tracks timestamps and click counts to accurately distinguish between single, double, and triple-click events, ensuring the correct action is taken.

4.6 Multithreading for Listener

To maintain smooth operation and avoid freezing the interface, the mouse listener runs on a separate background thread, often referred to as `listener_thread`. This multithreading approach ensures that the listener operates independently of the main GUI or program loop, allowing the system to remain responsive while simultaneously monitoring for user actions. It's an essential architectural decision that enhances both performance and user experience.

4.7 Platform Detection & Typing

LinkedIn / Tumblr Window Detection is handled using the `win32gui` library. The functions `GetForegroundWindow()` and `GetWindowText()` are used together to identify the currently active window's title. This mechanism ensures that the automation logic—whether it's typing a label or story—is only executed when the correct platform, such as LinkedIn or Tumblr, is actively selected by the user. This helps maintain platform-specific control and prevents unwanted input into unrelated applications.

For typing automation, the system utilizes `pyautogui`. After confirming the target platform, it clicks on the input area of the social media feed using custom screen coordinates (which may be adjusted based on resolution). Depending on the type of click event, the system will then automatically type either the recognized label (in the case of a triple-click) or the generated story (from a double-click). This allows for seamless and hands-free content posting powered by AI.

4.8 GUI with Tkinter

The system includes a simple graphical user interface (GUI) that serves as the entry point for activating the assistant. The GUI features two main buttons: Start Listener, styled in green, and Stop Listener, styled in red. These buttons allow users to easily start or stop the mouse listener with a single click. Additionally, the GUI provides a status display using a `status_label`, which updates in real-time to indicate whether the assistant is currently running or stopped. This straightforward interface ensures ease of use, giving users full control over when the assistant should be active without needing to interact directly with the code or terminal.

4.9 Labels & Expandability

```
text_labels = ["a human", "a man", "a woman", "a child", "an animal", "a tree",  
"a banner", "a bird"]
```

The assistant uses a predefined list of text labels to help the CLIP model recognize objects within screenshots. These labels include general categories like "a human", "a man", "a woman", "a child", "an animal", "a tree", "a banner", and "a bird". When an image is captured, CLIP compares it against this list to determine the best match. To enhance recognition accuracy and expand its capabilities, users can easily add more labels—such as "a cat", "a book", or "a car"—based on the types of content they expect to encounter. Expanding the label set makes the assistant more intelligent and versatile across different visual contexts.

4.10 MODEL DIAGRAM

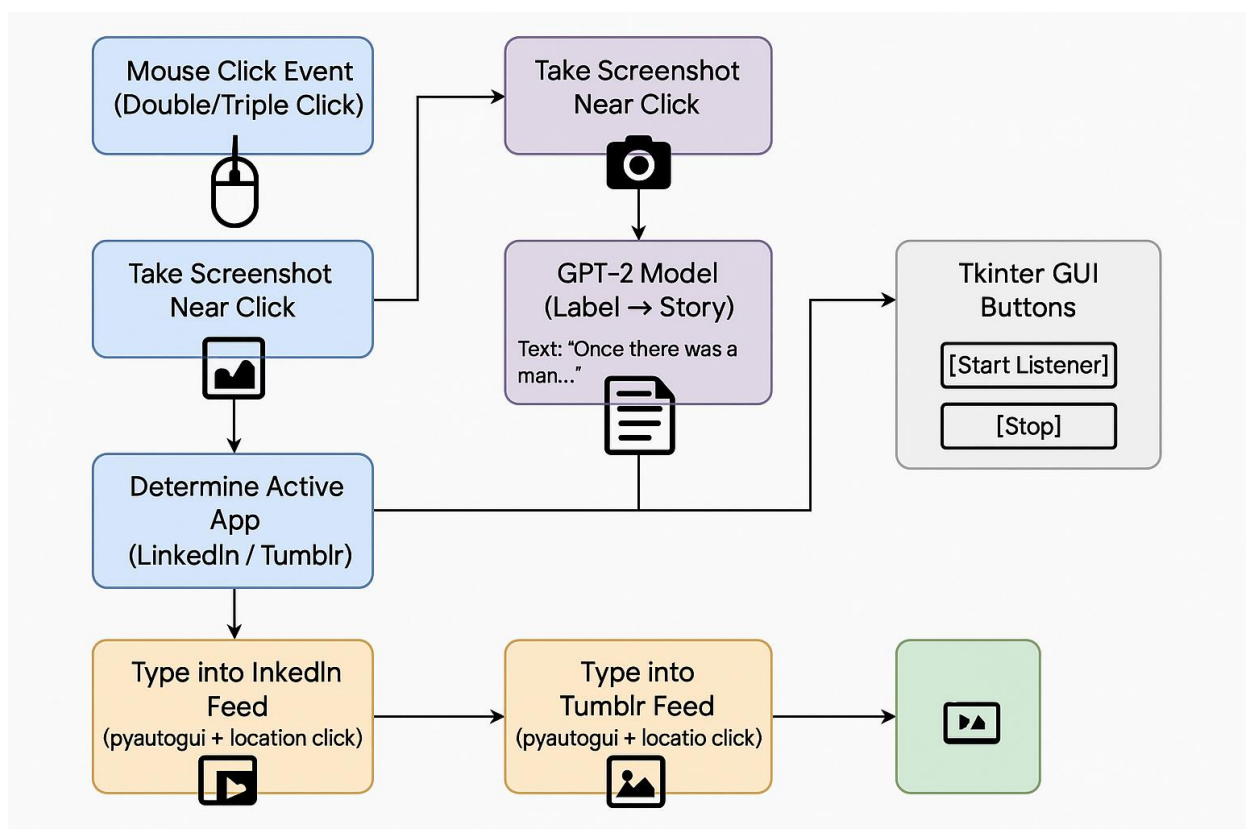


Fig4.1 Single Click Multiple Social Media Posting with Customized Copywriting Each

5. SYSTEM IMPLEMENTATION

5.1 DETAILED OVERVIEW OF SOCIAL MEDIA POSTING WITH CUSTOMIZED COPYWRITING EACH

The main objective of this system is to automate social media posting by generating AI-powered, platform-specific content using nothing more than a single mouse click. The process begins with a mouse event—either a double-click or triple-click—that triggers a screenshot capture of the area surrounding the cursor.

The captured image is analyzed by the CLIP model, which compares it against a set of predefined text labels such as "a bird", "a man", or "a banner", and determines the most relevant match. This recognized label is then passed into the GPT-2 language model, which generates a short and engaging story or caption based on that label, offering context-aware and creative content.

To ensure the generated content is posted in the right place, the system performs platform detection using window title analysis (e.g., checking if the active window is LinkedIn or Tumblr). Depending on the type of mouse click, the assistant takes action accordingly—a double-click initiates full image recognition and story generation, while a triple-click skips generation and simply types the recognized label into the input field.

The assistant includes a simple GUI interface with buttons to start or stop the mouse listener, and it displays real-time status using a label. Behind the scenes, the mouse listener runs in a background thread, ensuring that the GUI remains responsive. The image recognition pipeline uses pyautogui to take a screenshot, and content typing is also handled by pyautogui, simulating human-like typing speeds for a more authentic user experience.

This system is context-aware, capable of distinguishing between platforms by reading the active window title. It also relies on a curated set of predefined labels to improve image matching accuracy. Its modular architecture separates recognition, story generation, and posting into individual components, making it easier to manage and extend.

Future scalability is supported through a cross-platform design, which allows for easy expansion to other social networks like Twitter or Instagram by integrating custom

logic per platform. The use of threading ensures smooth multitasking, while GPT-2 fine-tuning options allow for story personalization, making content generation tailored to brand tone and audience.

In essence, the system brings together computer vision, natural language processing, and UI automation into a single-click solution for effortless, intelligent, and creative social media copywriting.

5.2 LIBRARIES USED IN THE PROGRAM:

PyAutoGUI is a powerful cross-platform GUI automation module in Python that allows for simulating both mouse and keyboard actions. It is primarily used in this assistant for capturing screenshots at the region where a user clicks, simulating mouse clicks on input fields (like LinkedIn or Tumblr), and typing out recognized labels or AI-generated stories. It even mimics human typing speed by introducing realistic delays, enhancing the user experience.

Pynput.mouse is a library that enables monitoring and controlling mouse input. Within the assistant, it listens for mouse events such as clicks and specifically detects double-clicks and triple-clicks. These click types serve as triggers—double-clicks initiate image recognition and story generation, while triple-clicks cause only the label to be typed into a post feed.

The **Transformers** library from Hugging Face is central to AI model handling in this assistant. It provides access to state-of-the-art models like BERT, GPT, and CLIP. Specifically, **CLIPModel** and **CLIPProcessor** are used for image-text matching, while **GPT2Tokenizer** and **GPT2LMHeadModel** are used for natural language generation. **CLIP** helps recognize the content of screenshots by comparing them to a set of text labels, and GPT-2 generates short, engaging stories based on the recognized label.

PIL (Python Imaging Library), particularly its Pillow fork, is employed for image preprocessing tasks. After a screenshot is taken, PIL is used to open, resize, and convert the image to RGB format, preparing it for compatibility with the CLIP model's input requirements.

Torch (PyTorch) serves as the backbone deep learning framework used to run both the CLIP and GPT-2 models. It facilitates efficient inference through the

`torch.no_grad()` context, ensuring predictions are fast and memory-optimized by disabling gradient tracking during evaluation.

The built-in **time** module is leveraged to measure the time between consecutive mouse clicks, which is crucial for distinguishing between double and triple clicks. It also introduces short delays during typing to make the output feel more natural and human-like.

Win32gui, part of the `pywin32` package, is a Windows-specific library that interacts with the Windows GUI through the Windows API. It is used here to detect the currently active window, helping the assistant determine whether to post content to LinkedIn or Tumblr.

Threading, a built-in Python module, plays a key role in running the mouse event listener in a background thread. This allows the GUI to stay responsive and interactive even while the system is actively listening for mouse events in the background.

Lastly, **Tkinter** is Python's standard GUI toolkit used to build the assistant's desktop interface. It provides a window with buttons for starting and stopping the mouse listener and displays the current system status. Additionally, it can help bring focus to a specific window title, supporting seamless integration with user workflows.

5.3 SINGLE CLICK MULTIPLE SOCIAL MEDIA POSTING WITH CUSTOMIZED COPYWRITING EACH MODEL

(I) PYTHON:

(a) app.py

Step 1: Imports and Model Setup

The script starts by importing essential Python libraries like pyautogui for automation, pynput.mouse for detecting mouse events, transformers and torch for using AI models, PIL for image processing, win32gui for checking the active window, tkinter for GUI creation, and threading for background task management.

Step 2: Load Models

The CLIP model (openai/clip-vit-large-patch14) is loaded for image recognition, while GPT-2 is used for generating text based on the recognized label. These models work together to identify objects in screenshots and create short stories.

Step 3: Labels to Classify

A list of predefined labels like “a man”, “a bird”, and “a tree” helps the CLIP model compare and classify images effectively.

Step 4: Image Preprocessing + Recognition

When a double-click occurs, a screenshot is taken and resized to fit the CLIP model's input requirements. The model then identifies the best-matching label for the image.

Step 5: Story Generation

Once a label is recognized, GPT-2 generates a short story using that label as the starting prompt, producing creative and engaging content.

Step 6: Active Window Checks

Functions check whether LinkedIn or Tumblr is the active window, ensuring that typing actions only happen in supported platforms.

Step 7: Display Label + Story

The recognized label and generated story are printed and typed into the active social media input box using pyautogui, simulating human-like typing.

Step 8: Typing into LinkedIn or Tumblr

On triple-click, only the label (not the full story) is typed into the feed. Mouse click coordinates may need adjustment based on screen resolution.

Step 9: Mouse Listener Logic

The script listens for mouse events. A double-click triggers recognition and story generation, while a triple-click types only the label. Timing logic differentiates the clicks.

Step 10: Threaded Listener

The listener runs in a background thread to keep the interface responsive. Start and Stop functions control whether the system is actively listening for clicks.

Step 11: Tkinter GUI

A simple graphical interface is created using Tkinter with a status label and two buttons—Start (green) and Stop (red)—to control the listener easily.

Step 12: Mainloop Starts GUI

Finally, `root.mainloop()` keeps the GUI active and listens for user interactions, making the assistant fully interactive without needing the command line.

(b) uli.py

Step 1: Importing Required Libraries

The program begins by importing key libraries such as pyautogui for automating mouse and keyboard actions, pynput.mouse for detecting mouse events, and transformers to load pre-trained AI models like CLIP and GPT-2. PIL is used for basic image processing, and win32gui helps detect which window is currently active on Windows.

Step 2: Model Initialization

Both the CLIP and GPT-2 models are loaded from Hugging Face. CLIP is responsible for recognizing image content, while GPT-2 generates a short text story based on that recognized label. The processor and tokenizer are also initialized accordingly.

Step 3: Predefined Labels

A list of text labels such as "a human", "a tree", and "a bird" guides the CLIP model by giving it known categories to match an image against. These labels can be expanded to improve recognition accuracy.

Step 4: Image Recognition Function

The image is first resized and converted to RGB for compatibility with CLIP. Then, using the `recognize_image` function, the model predicts which label best matches the captured screenshot region.

Step 5: Story Generation

Once a label is identified, GPT-2 takes it as input and generates a short creative story of up to 50 tokens. Sampling parameters like temperature and top-p are used to enhance creativity.

Step 6: Active Window Detection

Using `win32gui`, the system checks whether the currently focused window is LinkedIn or Tumblr. This ensures that the automated typing happens only in the intended platform's input field.

Step 7: Typing & Output Display

After recognizing the label and generating a story, the assistant displays this output and uses `pyautogui` to type it into the appropriate feed input box. Separate functions handle typing for LinkedIn and Tumblr with predefined cursor coordinates.

Step 8: Mouse Event Handler

The mouse click handler listens for double and triple clicks. A double-click captures a screenshot, processes it, and types the resulting story. A triple-click simply types the last recognized label without generating new content.

Step 9: Initialize Click Variables

Click timing and count are tracked using two variables. These help distinguish between double-click and triple-click actions based on timing.

Step 10: Start Listener

Finally, the mouse listener is activated to monitor user clicks continuously. It runs in real-time and drives the assistant's main interactions.

(c) up.py

Step 1: Import Required Libraries

The script starts by importing essential libraries—pyautogui to automate keyboard typing, pynput.mouse to listen for mouse events, time to handle timing between clicks and delays, and win32gui to detect the currently active window (specific to Windows).

Step 2: Define the Text to Type

A string, "I'm a human bot", is set as the message to be typed when a double-click is detected in the LinkedIn window.

Step 3: Set Target Window Title

The window title "LinkedIn" is used to verify if the user is currently focused on the LinkedIn window before performing any automated typing.

Step 4: Typing Function

A function called `type_text()` is defined to type out the message character by character with a small delay, imitating human typing. This function is only triggered after verifying the double-click and that LinkedIn is active.

Step 5: Active Window Check

Another function, `is_active_window_linkedin()`, uses `win32gui` to get the active window's title and checks if it contains the word "LinkedIn", ignoring case.

Step 6: Mouse Click Handler

The `on_click` function listens for mouse button presses and checks if two clicks happen within 0.3 seconds—indicating a double-click. If a double-click is detected and LinkedIn is the active window, the text is typed automatically.

Step 7: Click Timer Initialization

A timer variable is initialized to help track the interval between clicks and distinguish a double-click from a single one.

Step 8: Start the Listener

Finally, a mouse listener is started using `pynput`, which runs in a loop and listens for mouse activity until the script is manually stopped.

(d) icr.py

Step 1: Import Libraries

The script begins by importing essential libraries. `tkinter` is used to build the graphical user interface (GUI), `PIL` handles image loading and formatting, `torch` runs model computations, and Hugging Face's `transformers` library is used to load and interact with the CLIP and GPT-2 models.

Step 2: Load Pre-trained Models

Both the CLIP and GPT-2 models are loaded using pre-trained versions—CLIP for recognizing image content and GPT-2 for generating a creative story based on that content.

Step 3: Image Preprocessing

A simple preprocessing function ensures the selected image is opened correctly and converted to RGB format, making it compatible with the CLIP model.

Step 4: Image Recognition with CLIP

The recognition function compares the input image against a predefined list of text labels (like “a human”, “a tree”, etc.). CLIP returns the label that best matches the image content.

Step 5: Story Generation with GPT-2

Once a label is identified, GPT-2 takes it as a prompt and generates a short, imaginative story related to it. This story is then decoded into readable text

Step 6: Image Selection and Processing

When a user clicks the “Insert” button, they can choose an image file. The script processes the image, displays it in the GUI, runs it through CLIP, generates a story via GPT-2, and shows both the label and the story on the interface.

Step 7: GUI Interface

The GUI is set up using tkinter, with a window title and fixed size. It includes a button to select an image, a space to display the image, and a label area to show the output.

Step 8: Run the Application

Finally, the script enters the main GUI loop, keeping the interface active and waiting for user interaction.

(II) HTML:

(A) ss.html

Step 1: HTML Basics

The page starts with a standard HTML5 setup and sets the language to English.

Step 2: Head Section

In the <head>, it defines character encoding, makes the page responsive for all devices, and sets a title that appears in the browser tab

Step 3: CSS Styling

A `<style>` block is used to design the layout. The body has a dark background with centered content. A container holds two rectangular panels side by side. Each panel has its own scrollable area and a slightly different gray background. Headings and JSON text are styled for better readability.

Step 4: Body Layout

Inside the body, there's a box that holds two panels—one for each API. Each panel shows a title and an empty space with “Loading...” text, which will later be replaced by real data from the APIs.

Step 5: JavaScript Logic

JavaScript defines two API URLs (which you can change), and a `fetchData` function that fetches data from each API. It displays the result as formatted JSON or an error message if the request fails. When the page loads, both APIs are called and the data is shown in their respective panels.

(B) index.html

Step 1 & 2: HTML Setup & Title

The page begins with standard HTML5 boilerplate code, sets the language to English, and includes meta tags for mobile responsiveness and character encoding. The browser tab title is set to "Run Python Code".

Step 3: Styling the Page

The CSS gives the page a dark theme with white text and centers everything. The main button is large, circular, and has a pink glow effect on hover and click. A `<pre>` tag is styled to look like a terminal output, and the form is centered.

Step 4: Body Content

The body includes a heading titled “Some-po (Social media - copywriting AI)” and a form with a circular “Run” button. Below it, there’s an “Output” section using a `<pre>` tag where results of Python code execution are displayed.

Step 5: JavaScript Setup

JavaScript is used to interact with the form and button. It grabs references to the form, the button, and the output area. Two variables track the running state of the code and a placeholder for any active process.

Step 6: Button Logic

When the “Run” button is clicked, the script prevents the form from submitting the normal way. If the script isn’t already running, it sends a POST request to the `/run_code` endpoint and shows the result in the output area. The button then switches to say “Stop”.

Step 6 Continued: Stopping Execution

If the code is already running and the user clicks the button again, the script attempts to stop the process (though this only works if backend support exists). The output then says “Process terminated” and the button resets to “Run”.

(III) JAVASCRIPT

A. `auth-linkedin.js`

The code starts by importing Node.js’s built-in `querystring` module, which helps convert objects into URL query strings. It then defines a Netlify serverless function that triggers when the related endpoint is hit. Inside the function, it sets up LinkedIn’s OAuth authorization URL, along with required parameters like `client_id`, `redirect_uri`, `response_type`, `scope`, and a state string for security.

These parameters are pulled from environment variables and formatted into a query string. Finally, the function returns a 302 redirect response, pointing the browser to the full LinkedIn OAuth URL so the user can begin the login process.

B. auth-callback.js

The code begins by importing Axios to handle HTTP requests. It sets up a serverless function that runs when the user is redirected back to your app after logging in via Google or LinkedIn. It grabs the code and state from the query parameters — the code is used to get an access token, while the state indicates which provider was used.

If the state is "google", it sends a POST request to Google's OAuth endpoint to exchange the code for an access token. That token is then used to fetch the user's profile info from Google's userinfo API, and the result is sent back as JSON.

If the state is "linkedin", it performs a similar flow but using LinkedIn's OAuth token and profile APIs. The access token is retrieved using a application/x-www-form-urlencoded request, and then the profile info is fetched with a bearer token.

If the state is invalid, it returns a 400 error. Any other errors during the process are caught, logged, and returned as a 500 error response. This structure helps handle login callbacks for both providers in a single clean function.

6. TESTING

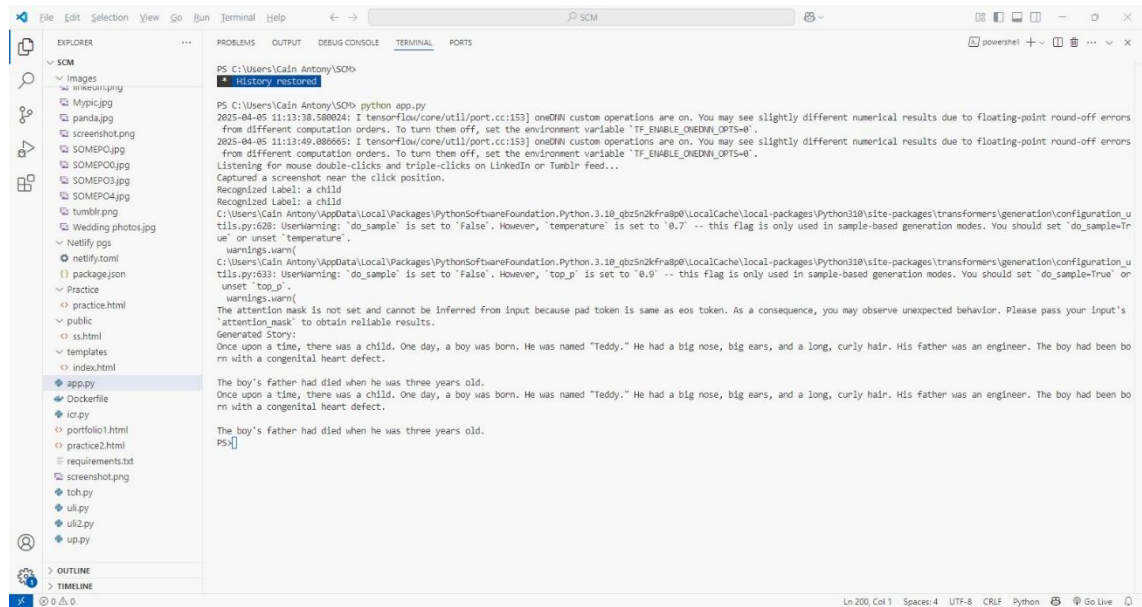
This Python script combines computer vision, natural language generation, and GUI automation to create a smart interactive tool. It listens for mouse clicks and reacts based on how many times you click. On a double-click, it captures a screenshot near the mouse pointer and uses OpenAI's CLIP model to identify what's in the image (like a person, tree, or animal). Based on that recognized label, it then uses GPT-2 to generate a short story. If you triple-click, it checks whether LinkedIn or Tumblr is the active window and types the recognized label automatically into the corresponding textbox.

Behind the scenes, it uses libraries like pyautogui and pynput for screen control and mouse listening, CLIP and GPT-2 models from Hugging Face's transformers for AI processing, and win32gui to detect the active app. It defines helper functions to preprocess images, recognize content, generate stories, and type into websites. Finally, it starts an event listener that continuously monitors your mouse clicks and triggers the appropriate action.

7. OUTPUT SCREEN

7.1 AI FRONTEND:

VS CODE TERMINAL WITH IMPLEMENTAION:



The screenshot shows the VS Code interface with a terminal window open. The terminal displays the output of a Python script. The output includes a message about custom operations being on, a warning about floating-point round-off errors, a message about listening for mouse double-clicks and triple-clicks on LinkedIn or Tumblr feed, a captured screenshot near the click position, a recognized label of a child, a warning about the attention mask not being set, and a generated story about a boy named Teddy. The terminal also shows the file explorer on the left with a list of files and folders.

```
PS C:\Users\Cain Antony\SOB> python app.py
2025-04-05 11:13:38.588024: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-04-05 11:13:49.086665: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
Listening for mouse double-clicks and triple-clicks on LinkedIn or Tumblr feed...
Captured a screenshot near the click position.
Recognized Label: a child
Recognized Label: a child
C:\Users\Cain Antony\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_ghc5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\torch\generation\configuration_utils.py:628: UserWarning: 'do_sample' is set to 'False'. However, 'temperature' is set to '0.7' -- this flag is only used in sample-based generation modes. You should set 'do_sample=True' or unset 'temperature'.
  warnings.warn(
C:\Users\Cain Antony\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_ghc5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\torch\generation\configuration_utils.py:633: UserWarning: 'do_sample' is set to 'False'. However, 'top_p' is set to '0.9' -- this flag is only used in sample-based generation modes. You should set 'do_sample=True' or unset 'top_p'.
  warnings.warn(
The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequence, you may observe unexpected behavior. Please pass your input's 'attention_mask' to obtain reliable results.
Generated Story:
Once upon a time, there was a child. One day, a boy was born. He was named "Teddy." He had a big nose, big ears, and a long, curly hair. His father was an engineer. The boy had been born with a congenital heart defect.

The boy's father had died when he was three years old.
Once upon a time, there was a child. One day, a boy was born. He was named "Teddy." He had a big nose, big ears, and a long, curly hair. His father was an engineer. The boy had been born with a congenital heart defect.

The boy's father had died when he was three years old.
PS C:\Users\Cain Antony\SOB>
```

Fig7.1 VS Code Terminal with Implementation

AI COPYWRITING AGENT (STATUS: STOPPED)

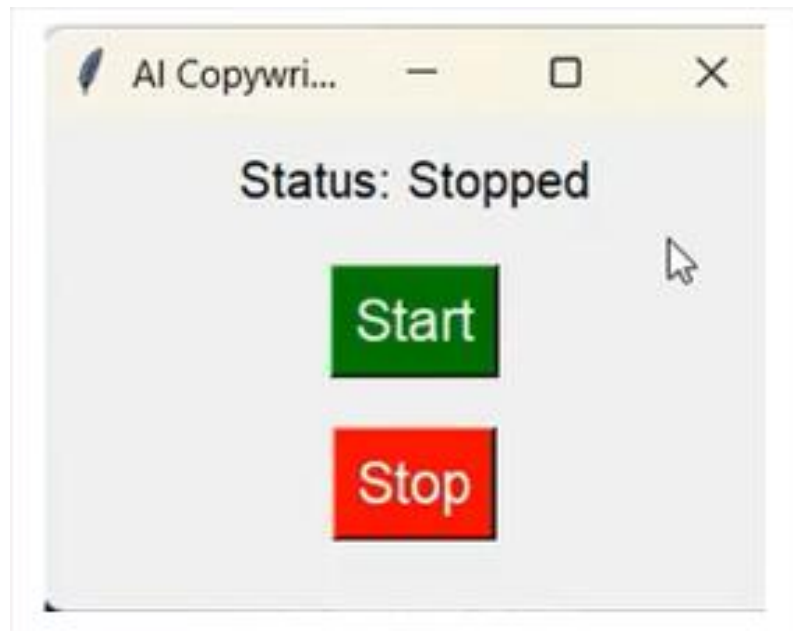


Fig7.2 AI Copywriting Agent (Status: Stopped)

In the above illustration, the AI copywriting agent is an AI frontend model created to manage the process of image recognition, label identification from backend, and the auto-generation of copywrite content. In this setup, HTML acts as the interface for delivering the core functionalities of the AI-powered agent. When the page is loaded, the user can initiate the backend Python code, which powers the AI Copywriting Agent—initially shown in a “stopped” state. The user can then select an image from local storage, and click the **Start** button to begin the workflow. Once started, a double-click on the image triggers the backend to perform image recognition and identify a label using CLIP. A triple-click on the story-feed section will prompt the system to generate copywriting content using GPT-2 based on the recognized label. These processes work seamlessly for both LinkedIn and Tumblr platforms. Once the tasks are complete on both platforms, clicking the **Stop** button will terminate the agent and halt all automated processes. Run the backend python code is mentioned to perform the process of image recognition, label identification from backend, and auto-generation of copywrite content based on identified label from backend. After the python code is executed, an AI copywriting agent would get generated with the status as stopped. Choose a photo/image from the local storage to post it in a social media platform. Before posting, click the start button to start the further processes. Double-click the image/photo for image recognition and label identification from backend where the mouse pointer moves as per the automation. Triple click the story-feed section to auto-generate the copywrite content created based on the label identified from backend. These processes are applied on both LinkedIn and Tumblr. After deriving the processes, click the stop button to terminate the process.

AI COPYWRITING AGENT (STATUS: RUNNING)



Fig7.3 AI Copywriting Agent (Status: Running)

After clicking the start button in the AI copywriting assistant, then it states the status as running indicating the process of image recognition, label identification from backend, and auto-generation of copywrite content based on label identified from backend were started. The user double-clicks the image/photo for image recognition and label identification from backend where the mouse pointer moves as per the automation. Then the user triple-clicks the story-feed section to auto-generate the copywrite content created based on the label identified from backend. These processes are applied on both LinkedIn and Tumblr. After deriving the processes, click the stop button to terminate the process.

COPYWRITE CONTENT IN LINKEDIN AND TUMBLR (OUTPUT)

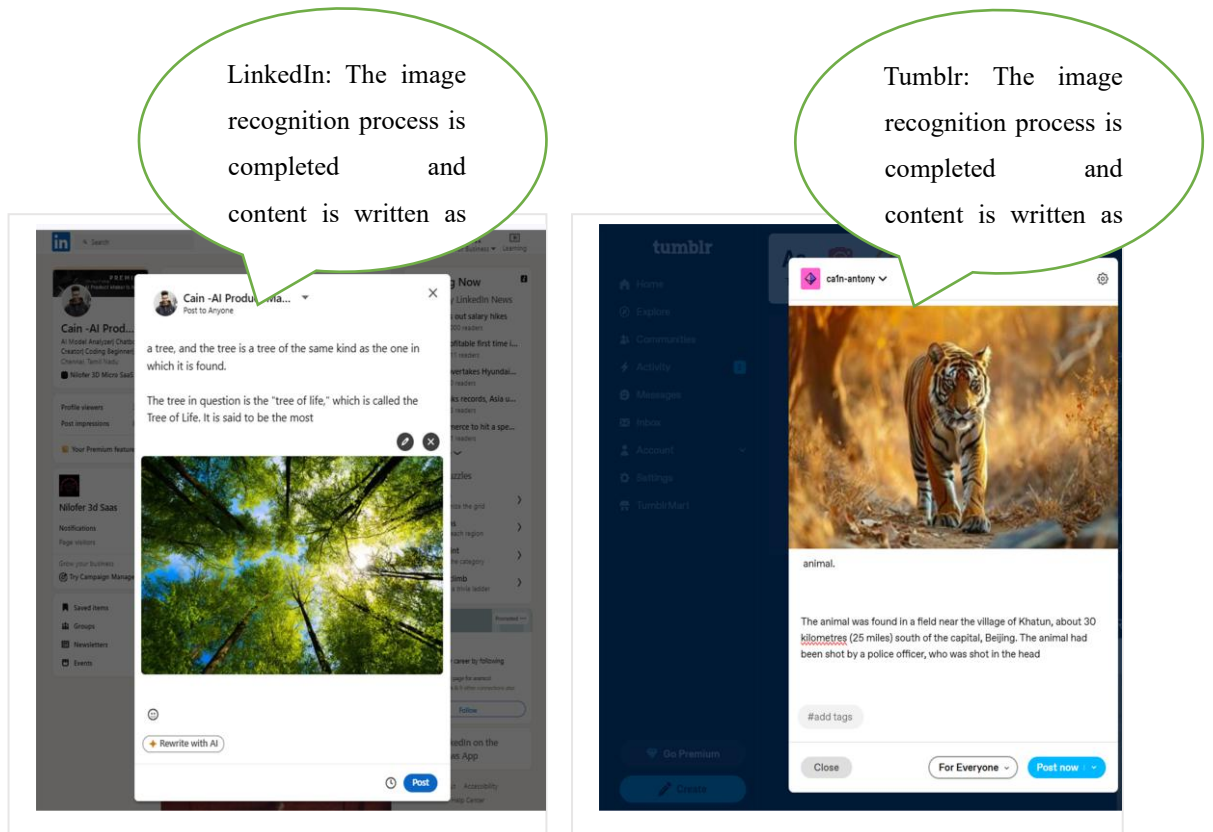


Fig7.4 LinkedIn and Tumblr with Copywrite Content

Finally, the copywrite content is generated on both LinkedIn and Tumblr after double-clicking the photo/image to recognize and identify the label from backend, and triple-clicking the story-feed section to auto-generate the copywrite content based on the label identified from backend.

8. CONCLUSION

The development of a **single-click, multi-platform social media posting system with customized AI copywriting models** marks a significant advancement in digital marketing automation. By combining intelligent content generation, scheduling, optimization, and engagement tracking, this system offers a comprehensive solution for individuals and businesses aiming to streamline their online presence.

The AI-powered copywriting system has achieved several key milestones. It successfully delivers AI-personalized copywriting, tailoring platform-specific tone and language using advanced natural language generation techniques. With multi-platform integration, it enables unified content distribution across LinkedIn, Tumblr, and other social platforms via seamless API connectivity. A smart scheduling feature predicts the best times to post for optimal engagement, leveraging historical data and audience behavior. Hashtag and tag optimization is handled dynamically using NLP to maximize content visibility. Additionally, media auto-enhancement ensures images are resized and optimized automatically for consistency in branding. An analytics dashboard provides real-time tracking of engagement and sentiment, helping refine content strategies over time.

Looking ahead, the system is set to introduce voice-to-post functionality, allowing users to generate content hands-free using voice input. An AI trend predictor will suggest post themes based on real-time trend analysis, while multilingual expansion aims to support content creation in various international and regional languages. The system also plans to implement user behavior learning through reinforcement learning, adapting content suggestions based on feedback. Furthermore, a decentralized content approval flow is in development to support collaborative teams with AI-assisted review and approval processes.

On a broader scale, this solution contributes to **marketing democratization**, empowering small businesses and individual creators to deliver professional-level strategies without needing large teams. It promotes **sustainability in branding** by maintaining a consistent voice and presence across platforms, fostering long-term audience trust. The emphasis on **data-ethical automation** ensures AI is used responsibly, supporting transparency and personalization. Lastly, its modular design offers **enterprise scalability**, making it a powerful content management tool for large organizations operating across multiple departments and global markets.

9. FUTURE ENHANCEMENTS

Now, the model “Single Click Multiple Social Media Posting with Customized Copywriting Each” is an AI simulative type used in LinkedIn and Tumblr to recognize the image and label, and write a story based on the label identified in the backend as per the automation. But in the future, there is a possibility that this can be built executable as either a desktop or an android or apple IOS app where the end user can make use of this app in various social medias. The time with this app in the future can be changed faster, more supportive, stress-free, and more purposeful in using the app. Even, like Chat-GPT, the app can generate the story as per the automation based on the image gonna post it to either of the active social medias. This can bring a drastic change in story creation, typing, analysis, etc. using AI, where the users can experience the new aspects in using the app, increases the curiosity in learning to create an app via., a product using AI. Many clients can view the posts with story using AI, can complement about the app, can share their thought about its usage and experience while using the app. There are more advantages in terms of time consuming, copyrights, own story-creation, etc., and beneficial to the users to share the aspects about the app to use it to be supportive and comfort.

10. REFERENCES

1. <https://github.com/Cain2882/Social-Media-Customized-Posting-with-AI>
2. <https://console.cloud.google.com/apis/credentials?inv=AbmYvA&project=pelagic-region-447113-e2&supportedpurview=project>
3. <https://www.linkedin.com/developers/apps/221470625/products>
4. <https://www.tumblr.com/oauth/apps>
5. <https://huggingface.co/jeffboudier/vision-transformers-spain-or-italy-fan>
6. <https://huggingface.co/openai-community/gpt2>
7. <https://huggingface.co/openai/clip-vit-large-patch14>
8. <https://app.netlify.com/sites/somepo/overview>

GOOGLE SCHOLAR:

1. AI Copywriting Agent - <https://epub.fh-joanneum.at/obvfjhjs/content/titleinfo/10868694/full.pdf>
2. Social Media Copywrite Content - <https://www.academia.edu/download/113977307/2132.pdf>
3. AI auto-story writing - https://www.researchgate.net/profile/Xiaoxuan-Fang/publication/369799144_A_systematic_review_of_artificial_intelligence_technologies_used_for_story_writing/links/64c8ec53b7d5e40f331956ca/A-systematic-review-of-artificial-intelligence-technologies-used-for-story-writing.pdf
4. Story-writing in LinkedIn - <https://dspace.yasar.edu.tr/bitstream/handle/20.500.12742/11474/TEZ-0786.pdf?sequence=1>
5. Story-writing in Tumblr - https://clrn.dmlhub.net/wp-content/uploads/2014/05/Schooling-the-Directioners_Korobkova.pdf
6. Social Media content creation using AI- https://www.theseus.fi/bitstream/handle/10024/854908/Hanninen_Rita.pdf?sequence=2