FASHIONGPT

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

Submitted by

G. Anusha(20B81A1232)

Ch. Gruhitha(20B81A1215)

K. Venu Gopal(20B81A1263)

D. Narayana Reddy(20B81A1222)

Under the Esteemed Guidance of

Sri. G VIHARI

Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

SIR C. R. REDDY COLLEGE OF ENGINEERING

APPROVED BY AICTE

ELURU – 534007

2023 – 2024

SIR C.R. REDDY COLLEGE OF ENGINEERING

DEPARTMENT OF INFORMATION TECHNOLOGY

Affiliated to Jawaharlal Nehru Technological University, Kakinada

Eluru-534007

BONAFIDE CERTIFICATE

This is to certify that the project report entitled "**FASHIONGPT**" being submitted by **G. Anusha(20B81A1232)**, **CH. Gruhitha(20B81A1215)**, **K. Venu Gopal(20B81A1263)**, **D. Narayana Reddy(20B81A1222)** in partial fulfillment for the Bachelor of Technology in Information Technology to the Jawaharlal Nehru Technological University is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree.

Sri. G Vihari
Project Guide & Asst. Prof
Department of IT
Sir C R Reddy College of Engineering

Dr A. Yesu Babu
Professor & HOD
Department of IT
Sir C R Reddy College of Engineering

EXTERNAL EXAMINAR

ACKNOWLEDGEMENT

First and foremost, I give thanks to almighty God for all his blessings that he bestowed upon me without which I won't be where I am today.

We express my sincere thanks to our principal, **Dr K. VENKATESWARA RAO**, for providing me the necessary infrastructure and his guidance in the efficient completion of the project work.

We owe a great deal of gratitude to our beloved Head of Department**, Dr A. YESU BABU**, for his continuous support in the journey of my course, for his patience, motivation, enthusiasm and immense knowledge.

We owe a great deal of gratitude to my project guide **Sri. G VIHARI** for his continuous support in the journey of my course, for his patience, motivation, enthusiasm and immense knowledge. His guidance helped me in all the time of work and completing this project successfully.

Our special thanks to librarian **Smt. D. LAKSHMI KUMARI**, and to the entire library staff Sir C. R. R College of Engineering, for providing the necessary library facilities.

We express our earnest thanks to all the faculty members and non-teaching staff of IT department for extending their valuable support.

PROJECT MEMBERS

G. Anusha	Reg No: 20B81A1232
CH. Gruhitha	Reg No: 20B81A1215
K. Venu Gopal	Reg No: 20B81A1263
D. Narayana Reddy	Reg No: 20B81A1222

DECLARATION

We hereby declare that the dissertation entitled "**FASHIONGPT**" is submitted to the B-Tech degree is my original work and the dissertation has not formed the basis for the award of any degree, fellowship or any other similar titles.

PROJECT MEMBERS

G. Anusha	Reg No: 20B81A1232
CH. Gruhitha	Reg No: 20B81A1215
K. Venu Gopal	Reg No: 20B81A1263
D. Narayana Reddy	Reg No: 20B81A1222

ABSTRACT

The Fashion GPT project aims to leverage Natural Language Processing (NLP) techniques to generate compelling and fashion-forward for various fashion-related image content. With the ever-growing influence of social media, ecommerce, and digital platforms in the fashion industry, there is a rising demand for engaging and creative textual content to accompany visual elements such as images and videos. The proposed Fashion GPT model utilizes Machine learning techniques, particularly the GPT-2 architecture, to understand and generate contextually relevant and stylistically appealing descriptions for fashion-related items, outfits, and trends. The model is trained on a diverse dataset comprising fashion articles, product descriptions, and user-generated content from reputable sources to ensure a comprehensive understanding of the language used in the fashion domain.

CONTENTS

LIST OF FIGURES

LIST OF LIBRARIES

1. PIL-Python Imaging Library
2. StableDiffusionPipeline
3. Diffusers
4. IPython
5. Webcolors
6. Torch

1. INTRODUCTION

1.1 Introduction

For a human mind it is very easily to thin of new content. what if someone asks you to "suggest a dress for a situation". It is very easy for us to do that. but machines process information very differently. Just understanding the structure of the above sentence is a difficult task for them let alone generate something based on that description.

FashionGPT is an AI-powered fashion responsive system designed to provide personalized Style suggestions and fashion advice to users. And the Functionality Using Machine learning algorithms, FashionGPT analyzes user preferences, fashion.

Automatic synthetic content generation is a field that has been explored in the past and was discredited because at that time neither the algorithms existed nor enough processing power that could help solve the problem. However, the advent of deep learning started changing the earlier beliefs. The tremendous power of AI to capture the features even in the humongous of datasets makes them a very viable candidate for automatic content generation. another milestone was achieved when Ian Good Fellow proposed generative adversarial networks in 2014.

GANs are a kind of architecture in Machine learning that can produce content from random noise. What is even more unique about GANs is that the content they create represents the dataset on which they are being trained upon, but it is totally unique in some way or the other.

Generating an image from a text-based description is one aspect of generative adversarial networks that we will focus upon. Since the GANs follow unsupervised learning approach we have modified them to take am input as a condition and generate based on the input condition. This can form base for many things like synthetic audio generation like the ones used in Siri or assistant, video content generation from just scripts. imagine entire movies made from just the script. These are some uses that many companies are researching about. modifying GANs and applying conditions on them isn'tlimited to just generating images, we can use it to create passwords that are veryhard to crack and numerous similar applications like this.

Machine Learning and Content Generation

Machine Learning is a field that utilizes and relies completely on Various Flavors of Neural Networks to Extract Insights from the data and find patterns among that data. While it has been shown to be very successful in things like Image Classification (In some datasets even beating human level accuracy by a large margin) and Time Series Analysis(There are so many factors involved that it even becomes difficult for a human to take all those into account), A completelydifferent Aspect of it has been started to explore.

The big Question Being

"Can We use Machine Learning to Generate Content?"

As we know Neural Networks can extract features of a dataset that they have been trained upon, the goal becomes using those features to create new data points that do not belong in the dataset itself.

Generative AI models can generate stunning visuals, including graphics, images, art forms and videos. Marketers can leverage these AI-generated visuals to enhance their storytelling, create eye-catching social media posts and produce visually engaging presentations. machine learning content offer a variety of functions. Skills may include a range of actions like creating bulks of blog posts, subject lines, landing pages, emails, eBook’s, etc.

Generative Adversarial Networks

Generative Adversarial Networks (GAN’s) were created by Ian Good Fellow in 2014 to generate content instead of just representing it in a compact form and they are the most successful kind of Deep Learning Models that are even remotely close to the task.

What does GAN do?

Basically, it can be trained to generate data from a scratch or random noise.

It consists of two building blocks:

1) **Generator:**

The task of the Generator is to take in some input and generate something out of it. In cases of Images it might take in some noise as input and generate an image which might not mean anything Initially. It is simply the reverse of what a standard Convolutional Neural Network (CNN) is. A CNN takes in input as an image and down samples it along the height and width dimensions while increasing it along the channel dimension which acts as our features essentially. What a Generator Does is it takes in a down sampled input and through various Up sampling Operations Generates an Image.

By comparing the real images and the images that is generated by generator, GAN builds a discriminator that helps us to learn the differences that makes that image real and then after it will provide feedback to generator about the image that is to be generated next.

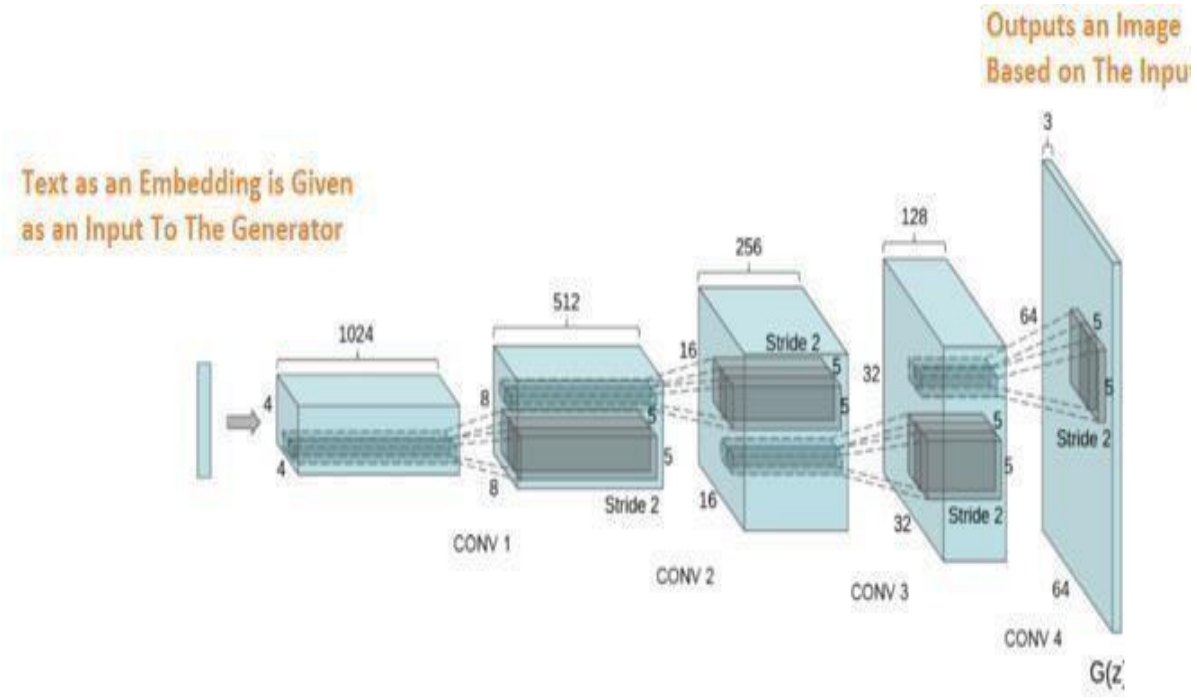


Figure 1: A standard Generator Architecture for our task

2) Discriminator:

Generator alone will just generate something random, so basically discriminator will give guidance to generator on what images it should create. Discriminator is nothing more than a simple convolutional neural network that takes in an image as an input and determines whether the image came from the original dataset or is it an image generated by the generator. Simply taking in an image as an input it determines whether it is real or fake (Synthetically Generated by Generator).

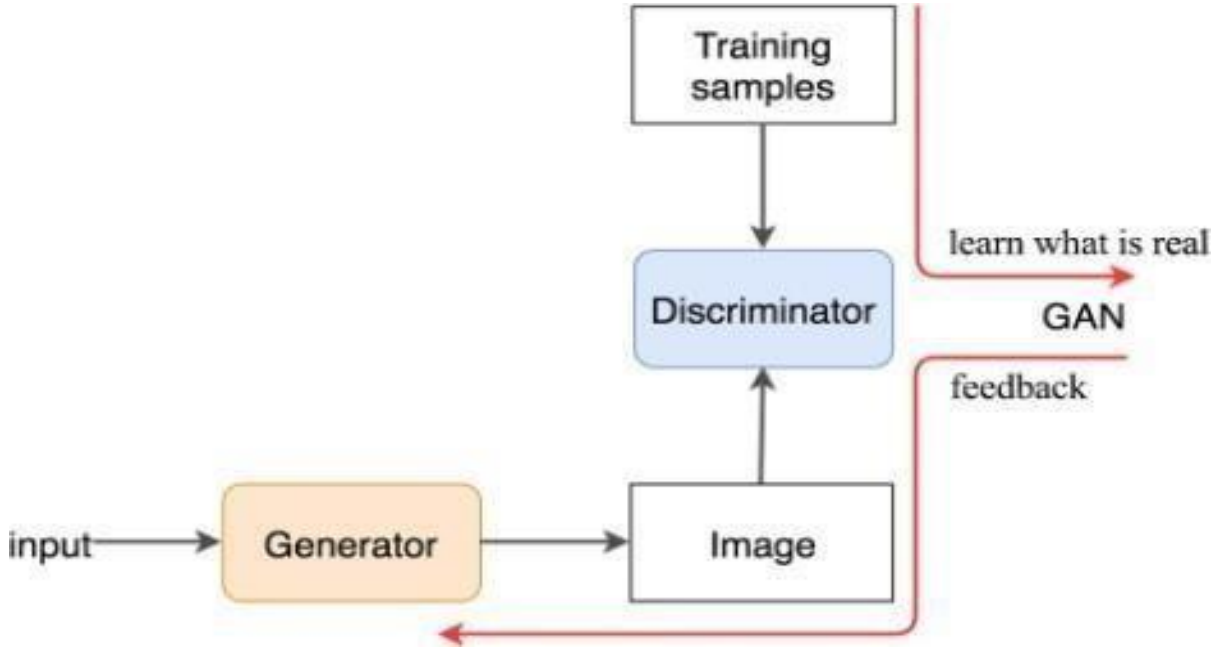


Figure 2

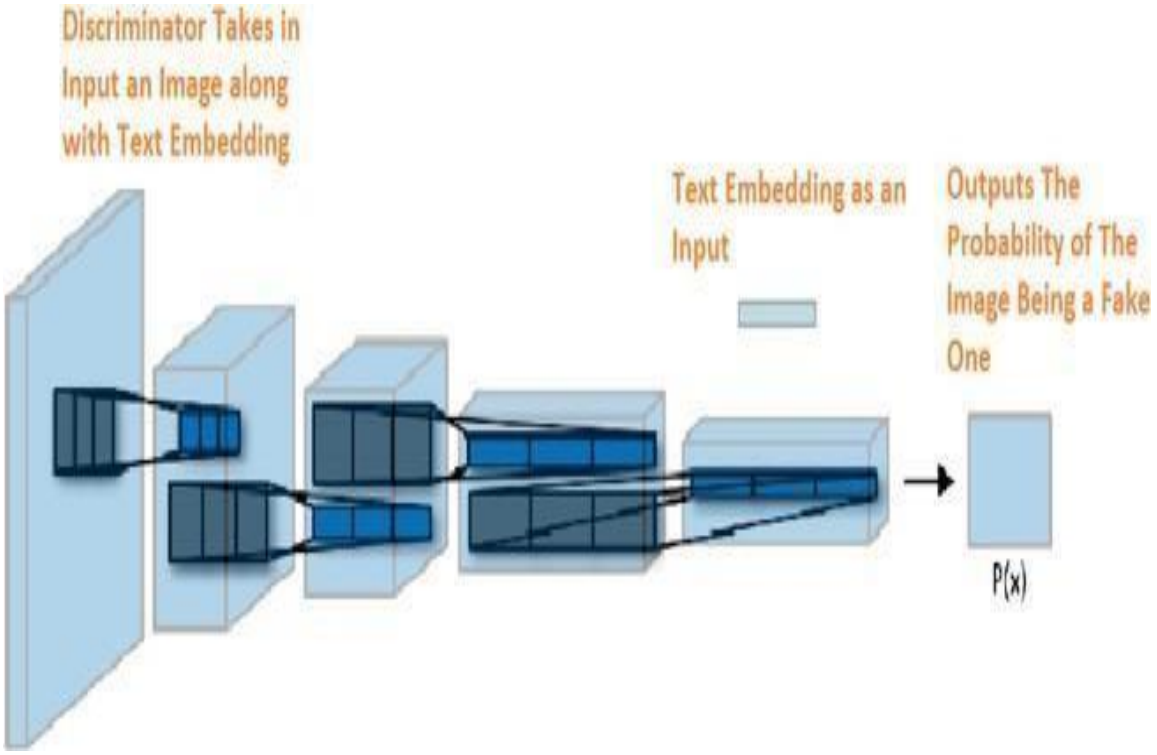


Figure 3: A Standard Discriminator for Our Task

Uses

1. Personalized Response

FashionGPT utilizes Machine learning algorithms to analyze user preferences and browsing behavior, allowing it to generate personalized clothing responsive tailored to everyone.

2. Improved Shopping Experience

By providing accurate and relevant clothing suggestions, FashionGPT enhances the overall shopping experience for customers, making it easier for them to discover new styles and find clothing items that match their unique tastes.

3. Virtual Shopping Experience

A virtual shopping experience is an online retail experience that simulates the feeling of shopping in a physical store. Through immersive technologies like virtual reality (VR) or augmented reality (AR), users can browse products, interact with virtual environments, and make purchases without leaving their homes. Virtual shopping offers convenience, accessibility, and a more engaging way to shop, bridging the gap between online and offline retail.

4. Variety of product

It means having lots of different things to choose from when you're shopping. When a store has a wide variety, it means they offer a range of items, which is great because it gives customers more options. So, whether you're looking for something specific or just browsing, having a variety of products means there's something for everyone. It makes shopping more fun and enjoyable.

5. Recommendation of application

It is an AI application designed for the fashion industry, offering personalized styling recommend. Virtual try-on capabilities, trend forecasting, style inspiration, and assistance with fashion content creation. It enhances the shopping experience, provides valuable insights for professionals, and helps users discover new trends and outfits.

6. Txt to Image generation

Text-to-image applications utilize artificial intelligence to generate images based on textual descriptions. By inputting descriptive phrases or sentences, users can prompt the system to create corresponding images, ranging from simple objects to complex scenes.

7. Text to Speech Synthesis

Generation of Realistic Synthesized Audio from Text is another very important application of this kind of model where the user enters a text and a waveform is generated automatically. This is useful in applications like Assistant.

1.2 Problem Statement

The Fashion GPT project seeks to address this issue by developing a sophisticated NLP model to enhance and streamline the content creation process in the dynamic and ever-evolving landscape of the fashion industry.

Generating Images from Text is a very difficult problem that can be approached by using Generative Adversarial Networks and will be extremely useful for content creators wherein they can type a description and have the type of content generated automatically saving them a lot of money and work. Imagine Thinking about a Description and having to draw something that matches the description in a meaningful way. It's even a difficult task for humans. But Machine Learning Can Understand the Underlying Structure of The Content and might beable to generate that automatically. Thereby eliminating the need of domain expertise.

GANs despite having all the upside for content generation are very difficult to train and Take a lot of time to converge and are unstable doing the training process and in this project, we also try to tackle these problems by modifyingthe Underlying Structure of the GAN Model.

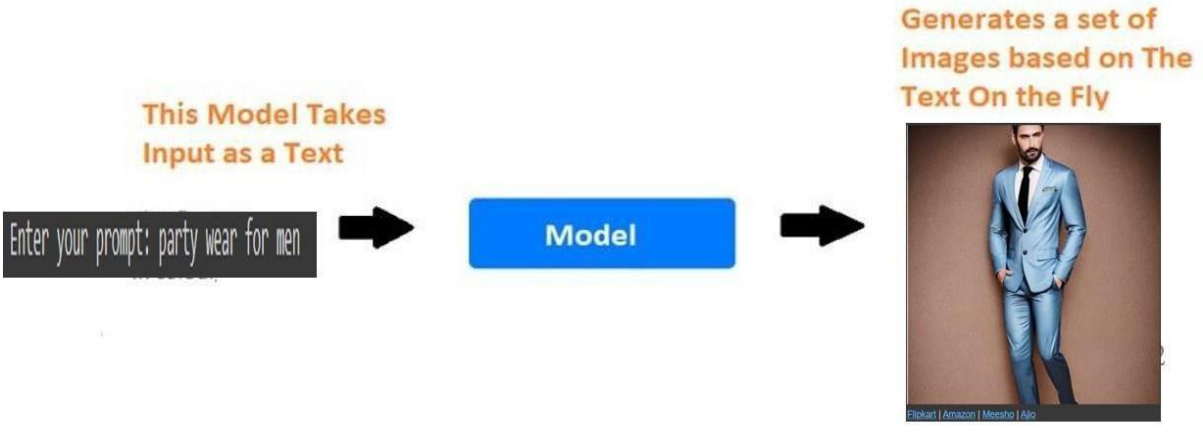


Figure 4: Gan Model

1.3 Objective

The main objective of this project is to generate the outfit according to the occasion in which a text can be inputted, and it outputs an image matching the description of the text and in doing so try to improve upon the generator architecture of the Generative Adversarial Networks. By modifying the input to a generator and applying conditions on the input we can create a model that generates images not from noise but from a controlled input.

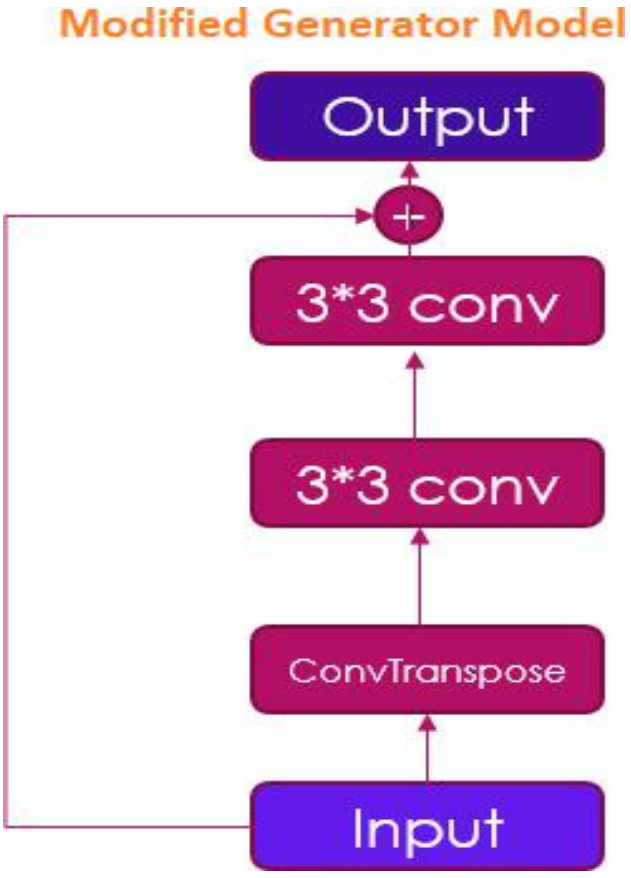


Figure 5

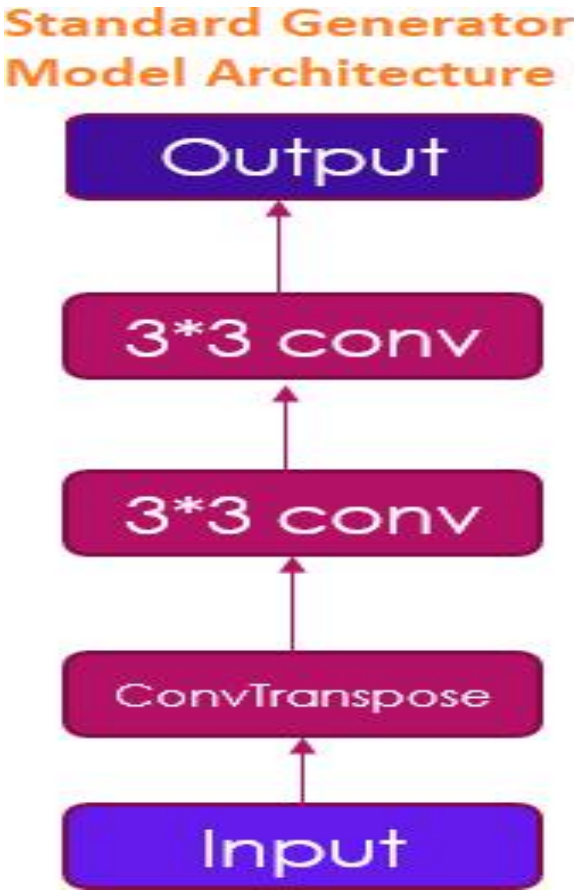

Standard Generator
Model Architecture
Output
3*3 conv
3*3 conv
ConvTranspose
Input

Figure 6

1.4 Methodology

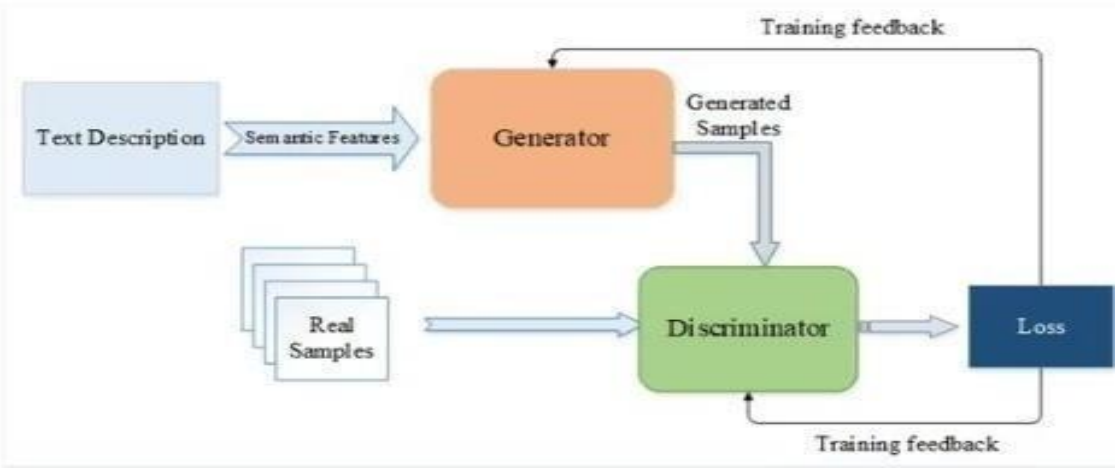


Figure 7: Methodology

We first start by integrating our code with AI and contains annotation for each image in the form of a text Description.

```
Enter your prompt: party wear for men
```

Figure: 8

Next, We Begin importing all the packages and the sub packages and splitting itinto the training, Validation and testing set. The following packages and libraries are being used to process The Dataset and build the architectures:

- NumPy
- Pytorch
- OpenCV

We first start by downloading and pre-processing the pre-trained dataset. During the pre-processing phase we convert text into embedding and normalize the images, so they are ready to be passed onto respective models We then start to build our Customized Generator Model and use a standard Pre-trained Model as the Discriminator After the model Creation we create a training script and take income best practices in the field of Deep Learning to train the model with stability using us customize Pytorch Trainer. The Final task is to redirect the final trained model into a Flask Web App so that Testing becomes easy.

1.5 Organization

The steps required for the project is:

Chapter 1

In this we provide a brief introduction to all the work that we did in the project and what the result looks like. Apart from that we also describe what arethe exact steps followed to reach the results.

Chapter 2

In this chapter we did a literature survey of things already existing whether in academia or in various conferences. We got to know our basic Understanding from this.

Chapter 3

In this chapter we go through all the system development process that we didand describe the algorithms used in detail. From LSTM models to generators everything involved in the project has been described under this.

Chapter 4

In this chapter we presented the results of the working model with accuracy of the discriminator and various screenshots of the working model.

Chapter 5

In this we present whatever conclusions that came out of the project and future works that can be done.

2. LITERATURE SURVEY

2.1 Books and Publication

To understand how to generate content required an extensive study of Deep Learning and Unsupervised learning and to do that we used various Books and Publication along with Some Blogs, Talks and Conferences.

To finally be able to generate content we needed to study how to properly do the following Steps.

- Pre-process the words into embedding. Taking in input as a word and doproper tokenization

- We need to convert the long sentence description into a word embedding to pass it into the generator.

- Pre-processing the image and using proper techniques to normalize them. Normalizing and augmenting the images using own methods in NumPy.

- Studying various Machine Learning Architectures and pros and cons of each. Studying about different architectures like auto encoders and GANs that already exist.

- Studying various methods and already existing architectures like the resnet.

Following books were studied to achieve the following:

1) **Natural Language Processing with Python by Steven Bird**

In this book I got familiarized with a lot of tools and techniques to process words and the Overall field of Natural Language Processing and the best practices to properly process my Input

2) **Neural Networks and Deep Learning by Michael Nielsen**

In this book I studied various architectures of Deep Learning that are most commonly used today along with proper techniques to train the models andsteps to avoid overfitting and underfitting

3) **Generative Adversarial Networks by Ian Good Fellow**

Ian Good Fellow is the creator of Generative Adversarial Networks and this publication helped me understand what GANs are, how they function and more importantly how can they even do what they do. Also Introduced me to some Problems like Mode Collapse.

3. SYSTEM ANALYSIS

3.1 Existed System

The existing text-to-text generation system is adept at comprehending and producing human-like textual responses across a broad spectrum of topics and contexts. This system employs sophisticate natural language processing (NLP) techniques, including recurrent neural networks (RNNs) or transformer models like GPT (Generative Pre-trained Transformer), to understand input text and generate coherent and contextually relevant output.

However, despite its strengths, this system faces certain limitations. One significant constraint is its reliance solely on textual data, which restricts its ability to comprehend and generate responses in other modalities such as images, audio, or video. This unimodal nature hampers its effectiveness in scenarios where, multi-modal communication is crucial for conveying information effectively.

For instance, in customer service applications, incorporating visual elements like diagrams alongside textual responses could enhance clarity and understanding. Similarly, in educational settings, the ability to provide not just text but also interactive visual aids or audio explanations could greatly enrich the learning experience. Moreover, in creative applications like storytelling or content generation, the exclusion of multimedia elements limits the system's expressive capabilities.

Overcoming this limitation would entail developing a multi-modal text-to-text generation system capable of understanding and generating responses across various modalities. By integrating text with other forms of media, such as images, audio, or video, this enhanced system could provide more comprehensive and engaging interactions, expanding its utility across diverse domains and applications.

3.2 Proposed System

Text Description to Embedding

Converting natural language text descriptions into images is an amazing demonstration of Machine Learning. Text classification tasks such as sentiment analysis have been successful with Machine Recurrent Neural Networks that are able to learn discriminative vector representations from text. In another domain, Deep Convolutional GANs are able to synthesize images such as interiors of bedrooms from a random noise vector sampled from a normal distribution. The focus of Reed et al. [1] is to connect advances in text embeddings and image synthesis with GANs, inspired by the idea of Conditional-GANs.

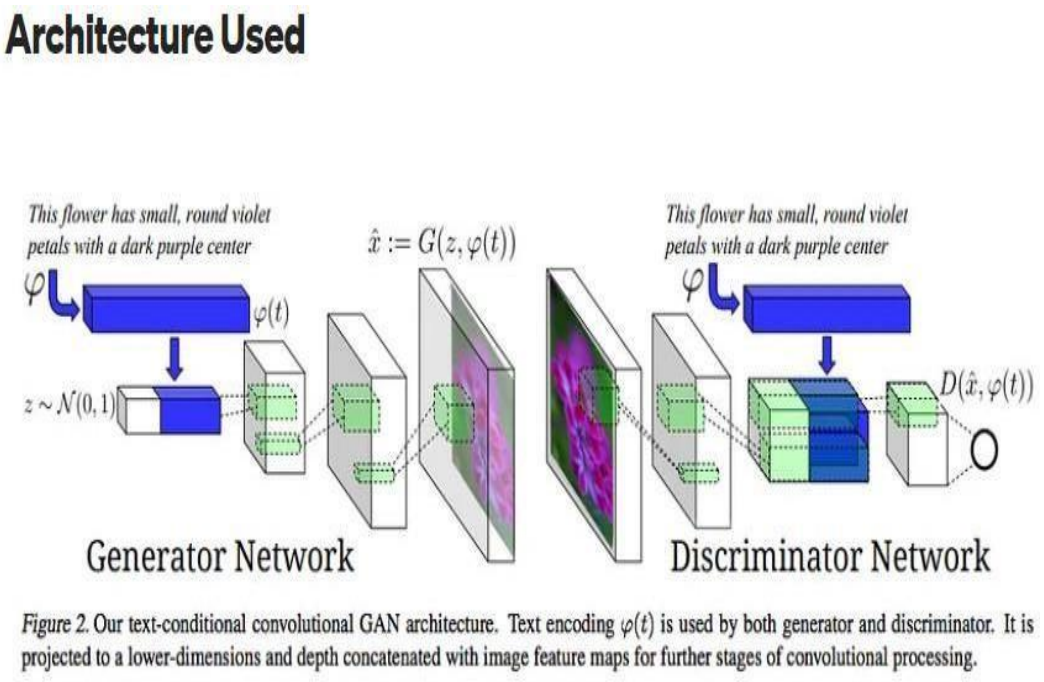


Figure 2. Our text-conditional convolutional GAN architecture. Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

Figure 9: Processing Sentences

Why Word Embedding

Why exactly do we need to convert our sentence into an Embedding and not justa one hot Encoded vector.

To Understand that let us take a very simple Example where in once we represent the Words as one hot encoded Vectors and in the other, we Use an Embedding Matrix.

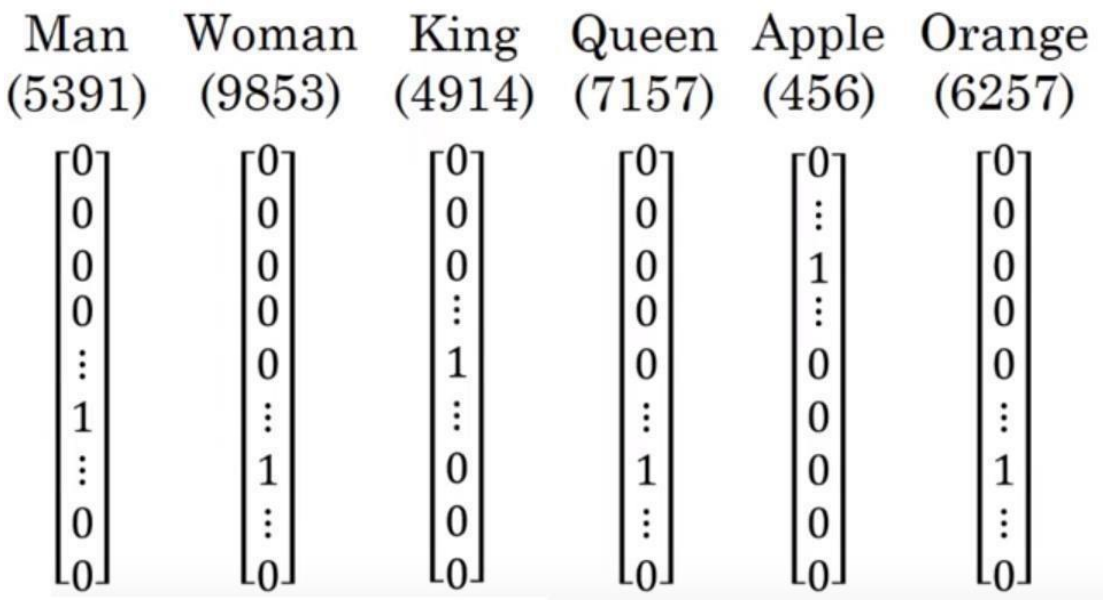


Figure 10: One Hot Encoded Vectors

The issue with representing words like this is:

1. Each word is a very high dimensional vector
2. Those Vectors do not have any kind of relation among them that a model can Learn and it becomes very difficult for it to learn when it cannot even understand the relation between words

Now let us Represent them in an Embedding

King (4914)	Queen (7157)	Apple (456)	Orange (6257)
-0.95	0.97	0.00	0.01
0.93	0.95	-0.01	0.00
0.7	0.69	0.03	-0.02
0.02	0.01	0.95	0.97

Figure 11: Embedding

When Represented like this the embedding for each vector has a meaning. When Representing these in Euclidean Space we will see that The Two Fruits are closer to each other while the King and Queen are very similar to each other in many respects except one which could be Gender.

It is not pre-decided on what features the model should learn but during the process model Itself decides the best values that reduce the loss and in process it learns the embedding That makes more sense to it.

Conditional-GANs work by inputting a one-hot class label vector as input to the generator and discriminator in addition to the randomly sampled noise vector. This results in higher training stability, more visually appealing results, as well as controllable generator outputs. The difference between traditional Conditional-GANs and the Text-to-Image model presented is in the conditioning input. Instead of trying to construct a sparse visual attribute descriptor to condition GANs, the GANs are conditioned on a text embedding learned with a Deep Neural Network. A sparse visual attribute descriptor might describe “a small bird with an orange beak” as something like:

```
[ 0 0 0 1 . . . 0 0 . . . 1 . . . 0 0 0 . . . 0 0 1 . . .0 0 0]
```

Pre-Processing the Images

Data Augmentation

Data Augmentation will help us to create more data to feed in to the model and help it to generalize well by letting it see the data in various orientations. We create our own transformation using NumPy. Here are some of the Augmentation that we will be implementing.

- Random Flip (Horizontal and Vertical)
- Random 90-degree Rotations
- Adding Lightening to the visual channels

Combining the random flip and random rotation we have come up with the 8 dihedral transformations that could be applied to any number of channels and onany kind of dataset as could be seen in the code snippet we first start by creating a function which takes in an input x as a tensor (Matrix Representation of Our Image) and a mode. We do not want to apply these image augmentations when we are in validation mode and testing the entire thing out in training mode, we need to randomly apply these transforms. We use the python's default random number generator to determine what kind of transformations would be randomlyapplied to the image.

```python
def detect_colors(image_path):
    try:
        image = Image.open(image_path)
        # Convert image to RGB mode
        image = image.convert("RGB")
        # Resize image for faster processing
        image = image.resize((100, 100))
        # Get colors using K-means clustering
        colors = image.getcolors(10000)  # Increase the parameter if needed
        sorted_colors = sorted(colors, key=lambda x: -x[0])  # Sort colors by frequency
        # Get the dominant colors
        dominant_colors = [color[1] for color in sorted_colors[:3]]
        for i in range(len(dominant_colors)):
          if dominant_colors[i]=="Unknown":
            dominant_colors[i]="Black"
        return dominant_colors
    except Exception as e:
        print(f"Error detecting colors: {e}")
        return []
```

To flip the image horizontally we first convert our tensor into a numpy array and then use the numpy fliplr function to flip the array horizontally and flipup to flip the array vertically. To rotate the image, we generate a random number k between 0 and 3 which determines how many 90-degree rotations of the arraywe will do. The following dihedral transformations could be formed after this step.

- Horizontal Flip + Any of three 90-degree Rotations
- Horizontal Flip with No Rotations
- Vertical Flip + Any of three 90-degree Rotations
- Vertical Flip with No Rotations

3.3 Creating Generator Model

The way a standard Generator Model Works is that it takes in some input and bya series of Up sampling or Deconvolution operations, it creates the Image. The only issue with that is while generating the final output it takes into account is the Information from the previous layer which are very ideal for tasks like Classification and Bounding Box Regression. But when dealing with Image Generation we should also keep into account the original input constraintswithout much processing along with the Information in the last layer as it will not only help the gradient flow better but also help converge the model faster.

```
def construct_flipkart_link(keywords):
    # Construct Flipkart search query based on keywords
    query = '%20'.join(keywords)
    flipkart_link = f"<a href='https://www.flipkart.com/search?q={query}' target='_blank'>Flipkart</a>"
    return flipkart_link

def construct_amazon_link(keywords):
    # Construct Amazon search query based on keywords
    query = '+'.join(keywords)
    amazon_link = f"<a href='https://www.amazon.in/s?k={query}' target='_blank'>Amazon</a>"
    return amazon_link

def construct_meesho_link(keywords):
    # Construct Meesho search query based on keywords
    query = '+'.join(keywords)
    meesho_link = f"<a href='https://meesho.com/search?q={query}' target='_blank'>Meesho</a>"
    return meesho_link

def construct_ajio_link(keywords):
    # Construct Ajio search query based on keywords
    query = '+'.join(keywords)
    ajio_link = f"<a href='https://www.ajio.com/search/?text={query}' target='_blank'>Ajio</a>"
    return ajio_link

# Get input from the user
prompt = input("Enter your prompt: ")
# Generate images based on the input prompt
generated_images = generate_images(prompt)
```

In the code snippet above we create our customized generator model from scratch using pytorch. We start off by declaring the class and then initializing the architecture within it.to properly use of pytorch inbuilt neural network layers we need to use super to inherit the properties of the base class we start off by declaring a convtranspose2d which essentially takes in the input embedding and starts by doubling along the height and width and reducing along the channel direction we add a dropout to increase regularization which not only deals with overfitting the model on the training but also helps the model generalize on the input features well this is followed by two convolutional blocks, one doubling along the channel dimension and the other one taking in that input and again reducing it back to original channel dimensions without any change in any other dimensions. This was done as in our practical implementations this trick worked out well now comes the major step of producing the final image. As we stated earlier that we also need to add in theoriginal embedding directly. But the issue with that is embedding has different dimensions altogether.

3.4 Training the Model

The training process of a Generative Adversarial Network is a bit complicated than training a normal Neural Network as it involves pre-training the discriminator and the generator in an alternating fashion.

```
# Display the generated images with links to buy similar dresses online
if generated_images:
    for i, (img, img_name) in enumerate(generated_images):
        display(img)  # Display the image
        # Detect colors in the image
        colors = detect_colors(img_name)
        # Convert RGB colors to color names
        color_names = [rgb_to_name(color) for color in colors]
        # Parse the prompt and extract relevant keywords
        keywords = parse_prompt(prompt)
        # Construct links for different platforms based on keywords
        flipkart_link = construct_flipkart_link(keywords)
        amazon_link = construct_amazon_link(keywords)
        meesho_link = construct_meesho_link(keywords)
        ajio_link = construct_ajio_link(keywords)
        # Display the links
        display(HTML(f"<p>{flipkart_link} | {amazon_link} | {meesho_link} | {ajio_link}</p>"))
else:
    print("No images were generated due to an error.")
```

We can further Optimize our Training By following Some Simple Steps

1) Proper Learning Rate Scheduler:

A constant learning rate isn't ideal when training our model as after some epochs we should decrease the learning rate by some amount to get closerto our desired value and fine tune the model better. This can be done with the help of a learning rate scheduler which takes in some input function and varies the learning rate by that. now decreasing the learning rate is a very common trend in machine learning. But initially we can increase the learning rate for a few epochs before starting to reduce it since it allows the model to jump high losses very quickly and then slowly fine tune as we can see in the graph it starts with a low learning rate and rises for afew iterations in an epoch and then it starts to fall down.

Figure 12: Learning Rate Scheduler

2) **Using Different Learning Rates for different Layers of the Discriminator:**

Training all the layers with same learning Rate can affect the Training since the earlier Layers of the model (closer to the Input) are likely to have learned more general features which we might not want to change that much as compared to later Layers so for earlier layers even a lower learning rate works while higher learning rate are suitable for later Layers

3) **Using Weight Decay and Dropout to deal with overfitting:**

Weight decay and dropout can be used to deal with overfitting. For our sample dataset a weight decay of 1e-4 and a dropout between 0.10 and 0.23 is working the best.

4) **Finding an optimal Learning Rate for The Discriminator:**

Instead of randomly starting with any learning rate we can train a subset of data between a learning rate range say 1e-7 to 1e-2 and observe the effect of each learning rate on the loss curve will help us determine the optimum rate. A graph could be plotted for learning rate vs training loss which could be used to determine the best learning rate to begin with. Going even further grouping up few layers we can find out the optimal learning rate for each layer groups and use that to further optimize the discriminator.

3.5 Redirecting to the Web-App

We Develop a Web App using flask that presents the user. The user with a web app and an option to input the text and choose a model from various inference Models that we have trained. On clicking generate Image, the request is processed in the backend in python and a resultant image using the model is created and we hit another flask endpoint where the generated Image is displayed. The following end points have been created in our existing flask application**:**

1) Home Route:

At the home endpoint or route, we redirect to the page where the user can provide with an input if the model has been loaded successfully without any error. If the chosen model cannot be loaded properly, we redirect to a route describing the error.

2) Generate Route:

After the user successfully enters a text it is pre-processed into a vector and passed on to our LSTM model that generates the word embedding. The embedded vector is then passed to the loaded generator model and is saved onto a location using timestamp as the file name.

3) Result Route:

After the Image has been successfully generated, we redirect the application to apage which displays the generated image.

4) Error Route:

The default route in case any error exists.

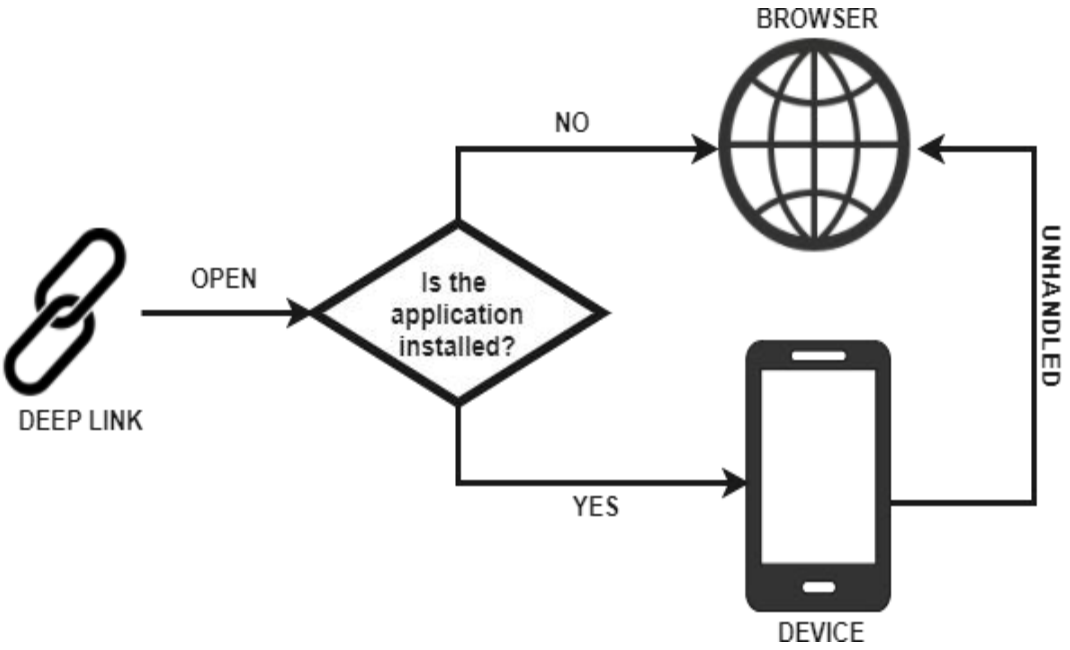


Figure 13: Flow graph for Redirection

A sample of the WebApp Could be seen in the Images Below:

Figure 14: Sample Redirecting to web app

4.SYSTEM REQIUREMENTS

4.1 Software Requirements

- Operating system: Windows 7 or above
- Technology: Machine Learning, python

4.2 Hardware Requirements

- RAM: 8GB and above
- Storage: 512 SSD (min)

5. PERFORMANCE ANALYSIS

5.1 Evaluation

It is not easy to evaluate the performance of a generative model based on any metric and mostly humans at the end must decide whether the generated content is good or not and whether or not it holds any particular meaning. However, we can judge the discriminator model in its ability to distinguish real images from fake ones. now compared to ordinary convolutional models where high accuracy means better results it isn't true in the case of generative models. If the discriminator has very high accuracy in distinguishing real images from fake ones, then that implies generator hasn't done a very good job in creating images that represent the dataset well. In the situation of a perfect equilibrium the discriminator should have an accuracy of 50% that is it has to take a random guess to determine whether the generated image is fake or not implying the generator has created images so good that are indistinguishable from the originalimages. The closer an accuracy is to 50% the better task the generator has done in creating images. The above graph shows us the loss that is the discriminator loss for the real images in blue color and discriminator loss for fake images in orange color and generator loss for the generated images in the green color.

This is an expected loss during this training and it will stabilize after around 35 epochs. The discriminator loss for the real and the fake images is approximately 50% and the generator loss for the generated images is between to 50% to 70 %.

This GAN model will get stabilized in 35 epochs and after that it will give us an accuracy approximately in between 70% to 80% and it will remain stabilized after that.

5.2 Training Phase Snapshots

In the starting as we get those images that have more or less noise but withoutany meaning.

Figure 15

Traditional wear for women in blue saree:

Figure 16

Traditional wear for women in sky blue saree:

Figure 17

Traditional wear for women in Green saree:

Figure 18

Traditional Wear For women in Pink Saree

Figure 19

6. DESIGN AND METHODOLOGY

6.1 Design

1. **Data Collection:** Gather a diverse dataset of fashion-related text, including descriptions of clothing items, fashion trends, style guides, and fashion articles. Additionally, collect images of fashion items paired with their descriptions.

2. **Preprocessing:** Clean and preprocess the text data by removing noise, irrelevant information, and standardizing the format.

3. **Model Architecture:** Design a transformer-based architecture tailored for fashion-related tasks. This architecture should include multiple layers of transformers with self- attention mechanisms for capturing long-range dependencies in the text data.

4. **Training:** Train the Fashion GPT model on the preprocessed data using techniques like transfer learning. Fine-tune the model on fashion-specific tasks such as generating clothing descriptions, suggesting outfit combinations, or predicting fashion trends.

5. **Evaluation:** Evaluate the performance of the Fashion GPT model using metrics like perplexity, BLEU score, or human evaluation to assess the quality of generated fashion-related text.

6. **Deployment:** Deploy the trained model as an API or integrate it into fashion-related applications, websites, or virtual styling platforms to provide users with personalized fashion recommendations, style advice, and trend forecasts.

7. **Continuous Improvement:** Continuously update and refine the Fashion GPT model by incorporating new data, fine-tuning the architecture, and addressing any biases or limitations in the generated output.

6.2 Modules

Module 1-Text Input Module: This module is responsible for processing textual inputs, such as fashion-related queries or prompts, and converting them into a format suitable for the model's input layer. It may involve tokenization, encoding, and embedding of the input text.

Module 2-Transformer Architecture: The core of the Fashion GPT model is based on a transformer architecture. This module comprises multiple layers of transformers with self-attention mechanisms for capturing textual dependencies and generating coherent fashion-related text.

Module 3 - Decoder Module: In the context of text generation tasks, the decoder module takes the output of the transformer layers and generates sequences of text. It may include techniques like beam search or nucleus sampling to improve the quality and diversity of the generated text.

Module 4 - Fine-tuning Module: To adapt the pre-trained Fashion GPT model to specific fashion related tasks, a fine-tuning module is used. This module fine-tunes the parameters of the model on a task-specific dataset, adjusting them to better fit the task requirements.

Module 5 - Deployment Module: Once trained and evaluated, the Fashion GPT model needs to be deployed for practical use. This module involves packaging the model into an accessible format,such as an API or a deployable package, and integrating it into fashion-related applications or

platforms.

Hugging Face Text to Image

The Stable Diffusion model was created by researchers and engineers from Comp Vis, Stability AI, Runway, and LAION. The Stable Diffusion Pipeline can generate photorealistic images given any text input. It’s trained on 512x512 images from a subset of the LAION-5B dataset.

This model uses a frozen CLIP ViT-L/14 text encoder to condition the model on text prompts. With its 860M UNet and 123M text encoder, the model is relatively lightweight and can run on consumer GPUs. Latent diffusion is the research on top of which Stable Diffusion was built. It was proposed in High-Resolution Image Synthesis with Latent Diffusion Models by Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer.

By decomposing the image formation process into a sequential application of denoising autoencoders, diffusion models (DMs) achieve state-of-the-art synthesis results on image data and beyond. Additionally, their formulation allows for a guiding mechanism to control the image generation process without retraining. However, since these models typically operate directly in pixel space, optimization of powerful DMs often consumes hundreds of GPU days and inference is expensive due to sequential evaluations. To enable DM training on limited computational resources while retaining their quality and flexibility, we apply them in the latent space of powerful pretrained autoencoders. In contrast to previous work, training diffusion models on such a representation allows for the first time to reach a near-optimal point between complexity reduction and detail preservation, greatly boosting visual fidelity. By introducing cross-attention layers into the model architecture, we turn diffusion models into powerful and flexible generators for general conditioning inputs such as text or bounding boxes and high-resolution synthesis becomes possible in a convolutional manner. Our latent diffusion models (LDMs) achieve a new state of the art for image inpainting and highly competitive performance on various tasks, including unconditional image generation, semantic scene synthesis, and super-resolution, while significantly reducing computational requirements compared to pixel-based DMs.

GPT-2

GPT-2 was created as a "direct scale-up" of GPT-1[8] with a ten-fold increase in both its parameter count and the size of its training dataset.[7] It is a general-purpose learner and its ability to perform the various tasks was a consequence of its general ability to accurately predict the next item in a sequence,[2][9] which enabled it to translate texts, answer questions about a topic from a text, summarize passages from a larger text,[9] and generate text output on a level sometimes indistinguishable from that of humans,[10] however it could become repetitive or nonsensical when generating long passages.[11] It was superseded by GPT-3

and GPT-4 models, which are not open source anymore.

GPT-2 has, like its predecessor GPT-1 and its successors GPT-3 and GPT-4, a generative pre-trained transformer architecture, implementing a deep neural network, specifically a transformer model,[8] which uses attention instead of older recurrence- and convolution-based architectures.[12][13] Attention mechanisms allow the model to selectively focus on segments of input text it predicts to be the most relevant.[14][15] This model allows for greatly increased parallelization, and outperforms previous benchmarks for RNN/CNN/LSTM-based models.[8]The Stable Diffusion model was created by researchers and engineers from CompVis, Stability AI, Runway, and LAION. The Stable Diffusion Pipeline can generate photorealistic images given any text input. It's trained on 512x512 images from a subset of the LAION-5B dataset. This model uses a frozen CLIP ViT-L/14 text encoder to condition the model on text prompts. With its 860M UNet and 123M text encoder, the model is relatively lightweight and can run on consumer GPUs. Latent diffusion is the research on top of which Stable Diffusion was built. It was proposed in High-Resolution Image Synthesis with Latent Diffusion Models by Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, Björn Ommer. By decomposing the image formation process into a sequential application of denoising autoencoders, diffusion models (DMs) achieve state-of-the-art synthesis results on image data and beyond. Additionally, their formulation allows for a guiding mechanism to control the image generation process without retraining. However, since these models typically operate directly in pixel space, optimization of powerful DMs often consumes hundreds of GPU days and inference is expensive due to sequential evaluations. To enable DM training on limited computational resources while retaining their quality and flexibility, we apply them in the latent space of powerful pretrained autoencoders. In contrast to previous work, training diffusion models on such a representation allows for the first time to reach a near-optimal point between complexity reduction and detail preservation, greatly boosting visual fidelity. By introducing cross-attention layers into the model architecture, we turn diffusion models into powerful and flexible generators for general conditioning inputs such as text or bounding boxes and high-resolution synthesis becomes possible in a convolutional manner. Our latent diffusion models (LDMs) achieve a new state of the art for image inpainting and highly competitive performance on various tasks, including unconditional image generation, semantic scene synthesis, and super-resolution, while significantly reducing computational requirements compared to pixel-based DMs.

Stable diffusion ai

Stable Diffusion is a generative artificial intelligence (generative AI) model that produces unique photorealistic images from text and image prompts. It originally launched in 2022. Besides images, you can also use the model to create videos and animations. The model is based on diffusion technology and uses latent space. This significantly reduces processing requirements, and you can run the model on desktops or laptops equipped with GPUs. Stable Diffusion can be fine-tuned to meet your specific needs with as little as five images through transfer learning. Stable Diffusion is available to everyone under a permissive license. This differentiates Stable Diffusion from its predecessors.

Stable Diffusion is important because it's accessible and easy to use. It can run on consumer-grade graphics cards. For the first time, anyone can download the model and generate their images. You also have control over key hyperparameters, such as the number of denoising steps and the degree of noise applied.

Stable Diffusion is user-friendly, and you don't need additional information to create images. It has an active community, so Stable Diffusion has ample documentation and how-to tutorials. The software release is under the Creative ML Open RAIL-M license, which lets you use, change and redistribute modified software. If you release derivative software, you have to release it under the same license and include a copy of the original Stable Diffusion license.

As a diffusion model, Stable Diffusion differs from many other image generation models. In principle, diffusion models use Gaussian noise to encode an image. Then, they use a noise predictor together with a reverse diffusion process to recreate the image. Apart from having the technical differences of a diffusion model, Stable Diffusion is unique in that it doesn't use the pixel space of the image. Instead, it uses a reduced-definition latent space.

The reason for this is that a colour image with 512x512 resolution has 786,432 possible values. By comparison, Stable Diffusion uses a compressed image that is 48 times smaller at 16,384 values.

This significantly reduces processing requirements. And it’s why you can use Stable Diffusion on a desktop with an NVIDIA GPU with 8 GB of RAM. The smaller latent space works because natural images aren't random. Stable Diffusion uses variational autoencoder (VAE) files in the decoder to paint fine details like eyes. Stable Diffusion V1 was trained using three datasets collected by LAION through the Common Crawl. This includes the LAION-Aesthetics v2.6 dataset of images with an aesthetic rating of 6 or higher.

What can Stable Diffusion do?

Stable Diffusion represents a notable improvement in text-to-image model generation. It’s broadly available and needs significantly less processing power than many other text-to-image models. Its capabilities include text-to-image, image-to-image, graphic artwork, image editing, and video creation.

Text-to-image generation

This is the most common way people use Stable Diffusion. Stable Diffusion generates an image using a textual prompt. You can create different images by adjusting the seed number for the random generator or changing the denoising schedule for different effects.

Image-to-image generation

Using an input image and text prompt, you can create images based on an input image. A typical case would be to use a sketch and a suitable prompt.

Creation of graphics, artwork and logos

Using a selection of prompts, it’s possible to create artwork, graphics and logos in a wide variety of styles. Naturally, it's not possible to predetermine the output, although you can guide logo creation using a sketch.

7. IMPLEMENTATION

7.1 Coding

```
from PIL import Image
import torch
from diffusers import StableDiffusionPipeline
from IPython.display import display, HTML
from webcolors import rgb_to_name as webcolors_rgb_to_name


class CFG:
    device = "cuda" if torch.cuda.is_available() else "cpu"
    seed = 42
    generator = torch.Generator(device).manual_seed(seed)
    image_gen_steps = 35
    image_gen_model_id = "stabilityai/stable-diffusion-2"
    image_gen_size = (400, 400)
    image_gen_guidance_scale = 9
    prompt_gen_model_id = "gpt2"
    prompt_dataset_size = 6
    prompt_max_length = 12


image_gen_model = StableDiffusionPipeline.from_pretrained(
CFG.image_gen_model_id, torch_dtype=torch.float32,revision="fp16",
use_auth_token='your_hugging_face_auth_token',
guidance_scale=CFG.image_gen_guidance_scale)
image_gen_model = image_gen_model.to(CFG.device)


def generate_images(prompt):
    try:
```

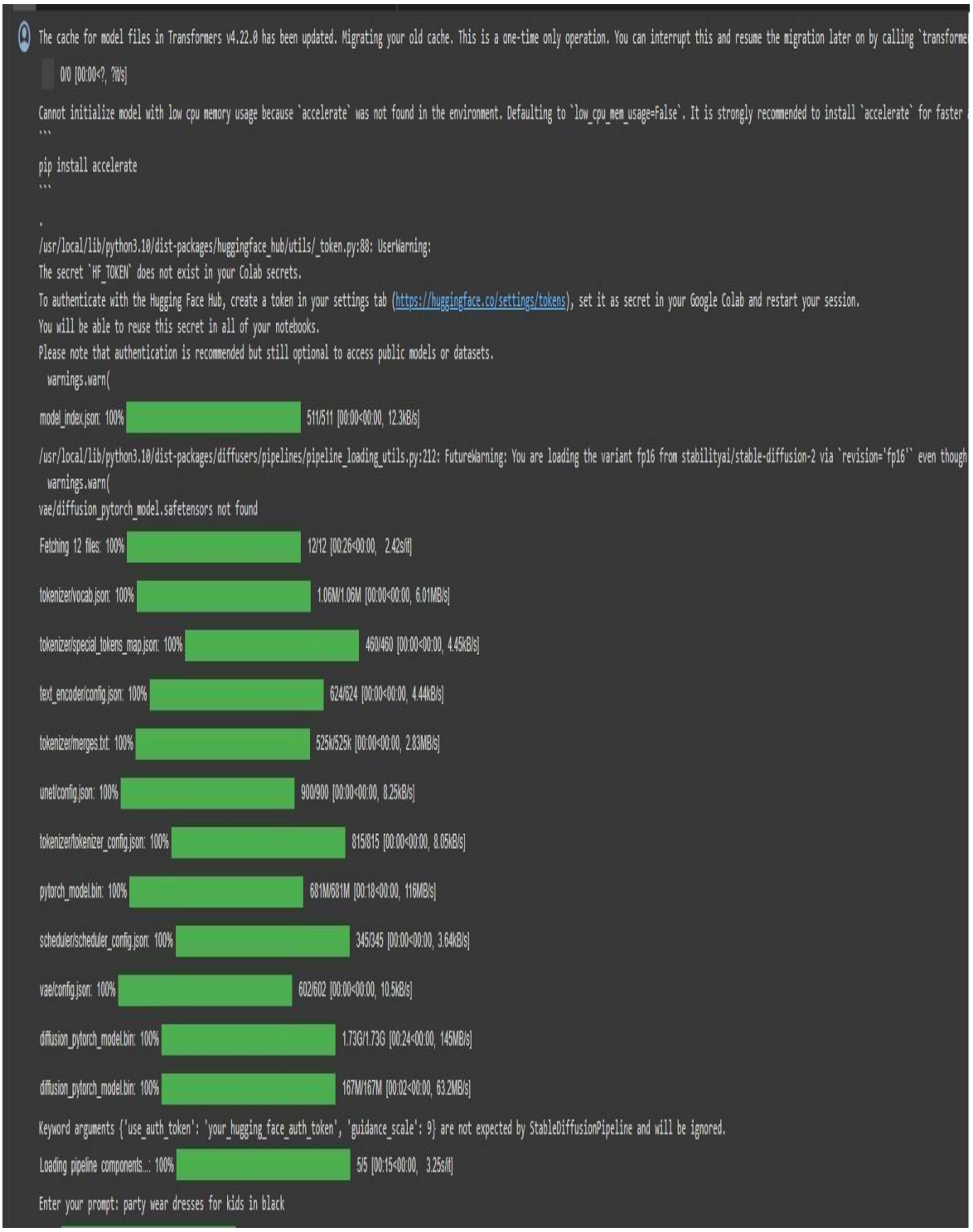
```
        images_with_links = []
        for i in range(5): # Generate 10 images based on the entered prompt
            image = image_gen_model(
                prompt, num_inference_steps=CFG.image_gen_steps,
                generator=CFG.generator,
                guidance_scale=CFG.image_gen_guidance_scale ).images[0]
            image = image.resize(CFG.image_gen_size)
            img_name = f"generated_image_{i}.png"
            image.save(img_name)  # Save the generated image
            images_with_links.append((image, img_name))
            # Append tuple of image and its file name
        return images_with_links
    except Exception as e:
        print(f"Error during image generation: {e}")
        return []


def detect_colors(image_path):
    try:
        image = Image.open(image_path)
        # Convert image to RGB mode
        image = image.convert("RGB")
        # Resize image for faster processing
        image = image.resize((100, 100))
        # Get colors using K-means clustering
        colors = image.getcolors(10000) # Increase the parameter if needed
        sorted_colors = sorted(colors, key=lambda x: -x[0])  # Sort colors by frequency
        # Get the dominant colors
        dominant_colors = [color[1] for color in sorted_colors[:3]]
```

```
                for i in range(len(dominant_colors)):
                 if dominant_colors[i]=="Unknown":
                   dominant_colors[i]="Black"
               return dominant_colors
            except Exception as e:
               print(f"Error detecting colors: {e}")
               return []

        def rgb_to_name(rgb_tuple):try:
               color_name = webcolors_rgb_to_name(rgb_tuple)
               return color_name
            except ValueError:
               return "Unknown"

        def parse_prompt(prompt):
            # Split the prompt into individual words
            keywords = prompt.lower().split()
            # Extract relevant keywords
            relevant_keywords = [keyword for keyword in keywords if keyword not in ['a', 'an',
'the',
            'to', 'for', 'in', 'on', 'at', 'of', 'with', 'about', 'from', 'by']]
            return relevant_keywords

        def construct_flipkart_link(keywords):
            # Construct Flipkart search query based on keywords
            query = '%20'.join(keywords)
```

```
        flipkart_link=f"<a href='https://www.flipkart.com/search?q={query}'
target='_blank'>Flipkart</a>"
            return flipkart_link


        def construct_amazon_link(keywords):
            # Construct Amazon search query based on keywords
            query = '+'.join(keywords)
            amazon_link=f"<a   href='https://www.amazon.in/s?k={query}'
target='_blank'>Amazon</a>"
            return amazon_link


        def construct_meesho_link(keywords):
            # Construct Meesho search query based on keywords
            query = '+'.join(keywords)
            meesho_link=f"<a href='https://meesho.com/search?q={query}'
target='_blank'>Meesho</a>"
            return meesho_link


        def construct_ajio_link(keywords):
            # Construct Ajio search query based on keywords
            query = '+'.join(keywords)
            ajio_link=f"<a href='https://www.ajio.com/search/?text={query}'
target='_blank'>Ajio</a>"
            return ajio_link


        # Get input from the user
        prompt = input ("Enter your prompt: ")


        # Generate images based on the input prompt
        generated_images = generate_images(prompt)
```

```
# Display the generated images with links to buy similar dresses online
if generated_images:
    for i, (img, img_name) in enumerate(generated_images):
        display(img)  # Display the image
        # Detect colors in the image
        colors = detect_colors(img_name)
        # Convert RGB colors to color names
        color_names = [rgb_to_name(color) for color in colors]
        # Parse the prompt and extract relevant keywords
        keywords = parse_prompt(prompt)
        # Construct links for different platforms based on keywords
        flipkart_link = construct_flipkart_link(keywords)
        amazon_link = construct_amazon_link(keywords)
        meesho_link  =  construct_meesho_link(keywords)
        ajio_link = construct_ajio_link(keywords)
        # Display the links
        display(HTML(f"<p>{flipkart_link}|{amazon_link}|{meesho_link}|
        {ajio_link}</p>"))
        else:
    print("No images were generated due to an error.")
```

7.2 Result


The cache for model files in Transformers v4.22.0 has been updated. Migrating your old cache. This is a one-time only operation. You can interrupt this and resume the migration later on by calling `transforme
0/0 [00:00<?, ?it/s]
Cannot initialize model with low cpu memory usage because `accelerate` was not found in the environment. Defaulting to `low_cpu_mem_usage=False`. It is strongly recommended to install `accelerate` for faster
```
pip install accelerate
```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
model_index.json: 100% 511/511 [00:00<00:00, 12.3kB/s]
/usr/local/lib/python3.10/dist-packages/diffusers/pipelines/pipeline_loading_utils.py:212: FutureWarning: You are loading the variant fp16 from stabilityai/stable-diffusion-2 via `revision='fp16'` even though
warnings.warn(
vae/diffusion_pytorch_model.safetensors not found
Fetching 12 files: 100% 12/12 [00:26<00:00, 2.42s/it]
tokenizer/vocab.json: 100% 1.06M/1.06M [00:00<00:00, 6.01MB/s]
tokenizer/special_tokens_map.json: 100% 460/460 [00:00<00:00, 4.45kB/s]
text_encoder/config.json: 100% 624/624 [00:00<00:00, 4.44kB/s]
tokenizer/merges.txt: 100% 525k/525k [00:00<00:00, 2.83MB/s]
unet/config.json: 100% 900/900 [00:00<00:00, 8.25kB/s]
tokenizer/tokenizer_config.json: 100% 815/815 [00:00<00:00, 8.05kB/s]
pytorch_model.bin: 100% 681M/681M [00:18<00:00, 116MB/s]
scheduler/scheduler_config.json: 100% 345/345 [00:00<00:00, 3.64kB/s]
vae/config.json: 100% 602/602 [00:00<00:00, 10.5kB/s]
diffusion_pytorch_model.bin: 100% 1.73G/1.73G [00:24<00:00, 145MB/s]
diffusion_pytorch_model.bin: 100% 167M/167M [00:02<00:00, 63.2MB/s]
Keyword arguments {'use_auth_token': 'your_hugging_face_auth_token', 'guidance_scale': 9} are not expected by StableDiffusionPipeline and will be ignored.
Loading pipeline components...: 100% 5/5 [00:15<00:00, 3.25s/it]
Enter your prompt: party wear dresses for kids in black

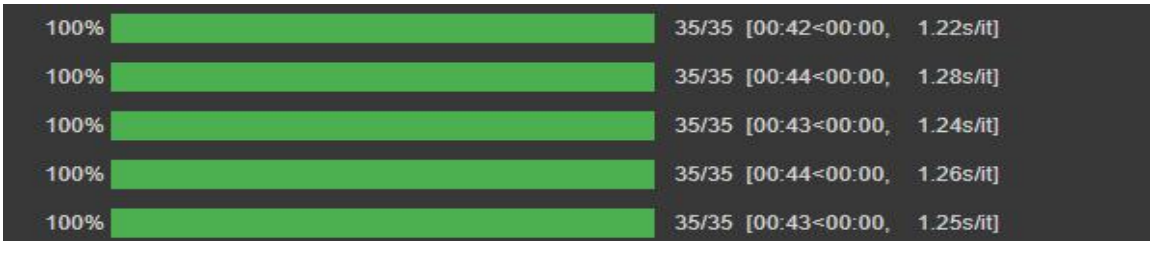

100% 35/35 [00:42<00:00, 1.22s/it]
100% 35/35 [00:44<00:00, 1.28s/it]
100% 35/35 [00:43<00:00, 1.24s/it]
100% 35/35 [00:44<00:00, 1.26s/it]
100% 35/35 [00:43<00:00, 1.25s/it]

Figure 20

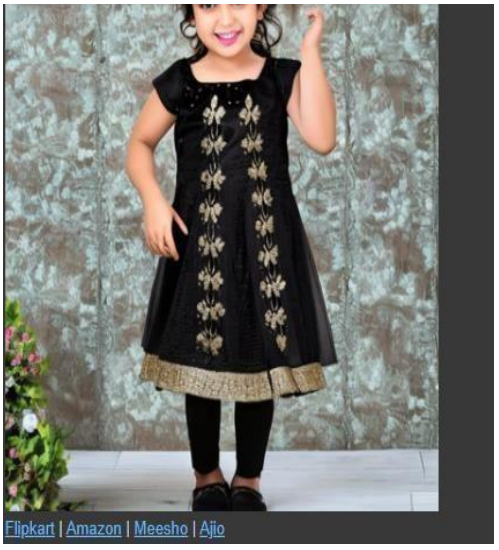


Figure 21

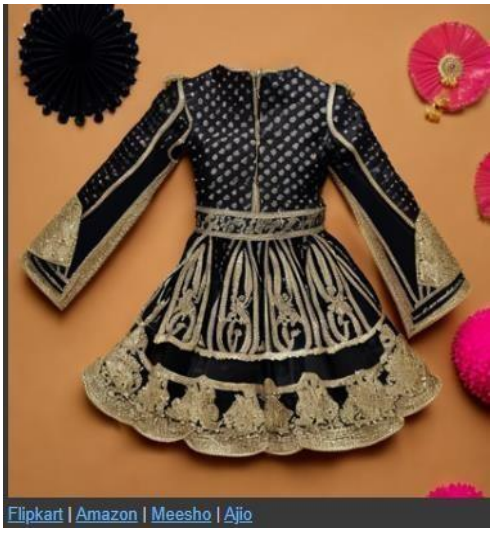


Figure 22

Figure 23

Figure 24

7.3 Testing

7.3.1 Introduction

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property function as a unit.

Test Cases

We now know, test cases are integral part of testing. So, we need to know more about test cases and how these test cases are designed. The most desired or obvious expectation from a test case is that it should be able to find most errors with the least amount of time and effort

Testing Objectives:

- To ensure that during operation the system will perform as per specification.
- To make sure that system meets the user requirements during operation
- To verify that the controls incorporated in the same system as intended
- Testing is a process of executing a program with the intent of finding an error
- A good test case is one that has a high probability of finding an as yet undiscovered error

7.3.2 Test Case Design:

White box testing

White box testing is a testing case design method that uses the control structure of the procedure design to derive test cases. All independents path in a module are exercised at least once, all logical decisions are exercised at once, execute all loops at boundaries and within their operational bounds exercise internal data structure to ensure their validity.

Black Box Testing

Black Box Testing attempts to find errors in following areas or categories, incorrect or missing functions, interface error, errors in data structures, performance error and initialization and termination error.

System Testing

Testing has become an integral part of any system or project especially in the field of information technology. The importance of testing is a method of justifying, if one is ready to move further, be it to be check if one is capable to with stand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical

Module Testing

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects,

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test ID	Test Case	Pass	Fail
1	Dhoti and kurta for men	Passed	Failed
2	Western wear for a lady	Passed	Failed

Test Results

All the test cases mentioned above passed successfully. No defects encountered Choose or create a text input that you want to use to generate the image. Select any keywords or themes that you want the AI to use to generate the image. Use the text to image generator to generate the image based on your input and selected keywords.AI-generated content can't be copyrighted because it isn't considered to be the work of a human creator.

Testing result 1

Testing result 2

8. CONCLUSION

8.1 Conclusion

This project enhances visual content creation by generating images related to given text inputs. The system effectively bridges the gap between textual descriptions and visual representations, offering a seamless and innovative approach to content generation.

In this project we have created a web app that can take in text description and generate images based on that. And while doing that we have modified the generator architecture in such a way that we have reduced thetraining time of GAN.

8.2 Future Scopes

Personalized Response

FashionGPT utilizes Machine learning algorithms to analyze user preferences and browsing behavior, allowing it to generate personalized clothing responsive tailored to everyone.

Improved Shopping Experience

By providing accurate and relevant clothing suggestions, FashionGPT enhances the overall shopping experience for customers, making it easier for them to discover new styles and find clothing items that match their unique tastes.

For the Future Work we can take the 64*64 image that we have obtained through our results and using Super Resolution convert it into a 256*256 image which can again be accomplished with the help of GAN. We can make two stages of the GAN that is the first stage will give us image that is 64*64 and second stage will give us 256*256 image.

8.3 References

1. https://github.com/CompVis/latent-diffusion
2. https://huggingface.co/docs/diffusers/en/api/pipelines/stable_diffusion/text2img
3. https://www.mdpi.com/2673-4591/20/1/16