

ABSTRACT

Images play a pivotal role in conveying information, emotions, and ideas. Being able to generate images from textual descriptions is crucial because it enables us to bridge the gap between language and visuals, making communication more effective and engaging. This project presents advancements in text-to-image generation facilitated by diffusion-based generative models, particularly focusing on the Stable Diffusion model. Leveraging principles of diffusion models, this project aims to address the limitations of traditional methods such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) in producing high-quality, diverse images from textual descriptions. The Stable Diffusion model operates by iteratively diffusing noise throughout the image space, gradually refining the generated image while conditioning on the given text prompt. This diffusion process allows for the gradual incorporation of textual information into the image generation process, resulting in images that accurately reflect the specified descriptions while maintaining visual fidelity and diversity. Experimental results demonstrate the effectiveness of our approach in generating high-quality images that closely resemble the textual descriptions. Our proposed method offers a promising solution for synthesizing realistic images from textual input, with potential applications in various domains such as computer vision, content creation, and virtual reality. We have also used the Hugging face models in the generation of images from the textual descriptions.

Keywords: Textual descriptions, Generation of Images, Variational autoencoders, Stable Diffusion, Hugging Face Model, Encoding & Decoding, Diffusion process.

LIST OF FIGURES

S. No.	Figure No.	Figure Name	Page No.
1.	Fig. 2.1:	AlignDraw Architecture	18
2.	Fig. 2.2:	Proposed Architecture	19
3.	Fig. 3.1:	Content diagram of the project	24
4.	Fig. 3.2:	Flowchart of the project	26
5.	Fig. 3.3:	U-Net Architecture	27
6.	Fig. 4.1:	Class diagram	33
7.	Fig. 4.2:	Use-case diagram	34
8.	Fig. 4.3:	Sequence diagram	35
9.	Fig. 4.4:	Activity Diagram	36

LIST OF TABLES

S. No.	Table No.	Table Name	Page No.
1.	Table 1	A comparative Study of various related works	10
2.	Table 2	Software Requirements	21
3.	Table 3	Python Packages	22
4.	Table 4	Frontend packages	22
5.	Table 5	Minimum Hardware Requirements	23
6.	Table 6	Test cases and it's validations	64

LIST OF SCREENS

S. No.	Screen No.	Screen Name	Page No.
1.	Screen 1	Project source code directory	40
2.	Screen 2	Home Page	54
3.	Screen 3	About Model	55
4.	Screen 4	Comparison between Models	55
5.	Screen 5	Sample Prompts and images generated by the model	56
6.	Screen 6	Page to enter the prompt to generate image	56
7.	Screen 7	Image generated by the Stable Diffusion model	57
8.	Screen 8	Hugging face model to generate images, text and audio	57
9.	Screen 9	Text to image generation page implemented using Hugging face model	58
10.	Screen 10	Image generated for the prompt car in a full traffic	58
11.	Screen 11	Text to Anime generation page implemented using Hugging face model	59
12.	Screen 12	Anime-Image generated for the prompt car in a full traffic	59
13.	Screen 13	Text to Speech Convertor Screen	60
14.	Screen 14	Voice to Text Converter screen	60

LIST OF ACRONYMS

U-NET	U-shaped network
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
VAE	Variational Autoencoder
DDPM	Denoising Diffusion Probabilistic Model
CLIP	Contrastive Language-Image Pre-Training
UML	Unified Modeling Language
CKPT	Checkpoint file
OS	Operating System
RAM	Random Access Memory

1. INTRODUCTION

Text-to-image generation using stable diffusion represents a cutting-edge convergence of natural language processing and advanced image synthesis techniques. In this project, we explore the exciting intersection of these domains to develop a novel approach for generating highly realistic and diverse images from textual descriptions.

Traditional methods of text-to-image generation have often struggled to capture the richness and complexity of textual descriptions, resulting in limitations such as poor image quality and lack of diversity. However, with the emergence of stable diffusion-based methods, we have witnessed a significant shift in the landscape of image synthesis.

By harnessing the diffusion techniques in a controlled manner, we can transform latent representations into detailed and coherent images over multiple iterations. The process from text to image begins with the encoding of textual descriptions into a latent space. This latent space serves as a bridge between the textual and visual domains, capturing the semantic essence of the input text in a structured format suitable for image synthesis. Through a series of diffusion steps, the latent representation undergoes gradual refinement, progressively revealing the visual content encoded within the text.

Stable diffusion-based text-to-image generation represents a transformative approach that bridges the gap between natural language processing and image synthesis. With its ability to produce diverse and realistic imagery across a wide range of textual descriptions. Whether describing concrete objects, abstract concepts, or fantastical scenes, the model can effectively translate textual descriptions into visually compelling representations with remarkable accuracy. In this project we have also used the Hugging face models in the generation of images from the textual descriptions.

1.1 MOTIVATION

The primary aim of text-to-image generation using stable diffusion is to bridge the semantic gap between textual descriptions and corresponding photorealistic images. Stable diffusion-based generative models leverage advanced AI concepts to interpret textual prompts and translate them into high-quality visual representations. By

harnessing diffusion techniques, the model navigates the complex process of synthesizing images from text, ensuring stability and coherence throughout the generation process. Ultimately, the goal is to empower users across various fields such as design, entertainment, and education to effortlessly create realistic images from textual descriptions, thereby enhancing creativity, productivity, and the overall user experience.

1.2 PROBLEM DEFINITION

The challenge of creating high-quality, photorealistic images from textual descriptions in a coherent and interpretable manner. This task encompasses multiple facets, including understanding the semantics and context embedded within the textual input, translating this information into visual features and structures, and ensuring stability and consistency throughout the generation process. Stable diffusion-based generative models aim to address these challenges by leveraging advanced AI techniques to interpret textual prompts, navigate the complex space of image generation, and produce visually compelling outputs that align with the provided descriptions. Key objectives include minimizing the semantic gap between text and image representations, preserving the fidelity and coherence of generated images, and facilitating intuitive and efficient interaction between users and the generative system. Through this approach, text-to-image generation using stable diffusion seeks to unlock new avenues for creativity, expression, and communication across diverse domains and applications.

1.3 OBJECTIVE OF THE PROJECT

The objective of the project is to develop a robust text-to-image generation system using stable diffusion-based generative models. This involves creating an AI-powered tool capable of accurately interpreting textual descriptions and generating corresponding photorealistic images with high fidelity and coherence. Key goals include minimizing the semantic gap between text and image representations, ensuring stability and consistency in the generation process, and facilitating intuitive user interaction. Ultimately, the objective is to empower users across various fields with a

powerful and accessible tool for seamlessly translating textual ideas into visual creations.

1.4 LIMITATIONS OF THE PROJECT

Limitations of the project include challenges in generating truly realistic images that accurately reflect the details of the input text. Despite advancements, the system may struggle with complex linguistic details, resulting in inaccuracies or inconsistencies in image generation. Additionally, the model's performance may vary across different languages, posing difficulties in achieving consistent results in multilingual settings. The system's ability to understand and interpret subtle cultural references or context-specific language may be limited, impacting the quality and relevance of generated images. Furthermore, computational constraints may restrict the system's capacity to handle large-scale datasets or process intricate textual inputs effectively, leading to potential limitations in scalability and performance.

1.5 ORGANIZATION OF THE PROJECT

Concise overview of rest of the documentation work is explained below.

Chapter 1: Introduction describes the motivation and objective of this project.

Chapter 2: Literature survey describe the primary terms involved in the development of this project.

Chapter 3: Analysis deals with the detail analysis of the project. Software Requirement Specification further contains software requirements, hardware requirements.

Chapter 4: Contains Methodology. It includes the system overview, modules, architecture diagram and algorithms used.

Chapter 5: Contains the implementation and results of the project.

Chapter 6: Contains project conclusion.

Chapter 7: Contains future enhancement.

Chapter 8: Contains references.

2. LITERATURE SURVEY

2.1 Survey based on ML and DL

These are the few research papers regarding text-to-image generation.

Sadia Ramzan, Muhammad Munwar, Iqbal and Tehmina Kalsum [1] Text-to-image generation is a pivotal technique with wide-ranging applications, yet current methods often struggle to produce realistic images based on textual descriptions. This study introduces RC-GAN, a deep learning architecture designed to address this challenge by seamlessly translating textual concepts into visually coherent images. Trained on the Oxford-102 flowers dataset, RC-GAN demonstrates superior performance, yielding more lifelike flower images with an inception score of 4.15 and a PSNR value of 30.12 dB. Future research aims to expand the model's capabilities by training it on diverse datasets.

Chenshuang Zhang, Chaoning Zhang, Mengchun Zhang, In So Kweon [2] This reviews text-to-image diffusion models in the context that diffusion models have emerged to be popular for a wide range of generative tasks. As a self-contained work, this survey starts with a brief introduction of how a basic diffusion model works for image synthesis, followed by how condition or guidance improves learning. Based on that, we present a review of methods on text-conditioned image synthesis, i.e. text-to-image. We further summarize applications beyond text-to-image generation: text-guided creative generation and text-guided image editing. Beyond the progress made so far, we discuss existing challenges and promising future directions.

Scott Reed, Zeynep Akata, Xinchun Yan, Lajanugen Logeswaran Bernt Schiele, Honglak Lee [3] Highlights the challenge of synthesizing realistic images from text, noting the current limitations of AI systems in achieving this goal. They propose a novel approach that combines recurrent neural network architectures for text feature representation and deep convolutional generative adversarial networks (GANs) for image generation. Their model effectively bridges advancements in text and image modeling, translating textual descriptions into plausible images of birds and flowers.

Sanyam Lakhanpal, Shivang Chopra, Vinija Jain, Aman Chadha, and Man Luo [4] Text-to-image generation using diffusion models has gained attention, but spelling inaccuracies in generated text remain a challenge. To improve accuracy, advanced techniques utilize glyph-controlled image generation, comprising a text layout generator followed by an image generator. However, these models face challenges, leading to the development of the LenCom-Eval benchmark for testing lengthy and complex visual text generation. Additionally, a training-free framework enhances two-stage generation approaches, demonstrating notable improvements in OCR word F1 scores on LenCom-Eval and MARIO-Eval benchmarks.

Seongmin Lee, Benjamin Hoover, Hendrik Strobelt, Zijie J. Wang, ShengYun Peng, Austin Wright, Kevin Li, Haekyu Park, Haoyang Yang, Duen Horng (Polo) Chau [5] the remarkable ability of diffusion-based generative models to generate images has drawn attention from all across the world. For non-experts, nevertheless, their complexity presents difficulties. In order to tackle this, we present Diffusion Explainer, an interactive visualization tool that streamlines the functions of Stable Diffusion. Through animations and interactive features, Diffusion Explainer provides a thorough explanation of the model's elements and functions, allowing users to comprehend how it functions. Users can compare the ways in which various text prompts affect the creation of images, which improves their comprehension of the process. Diffusion Explainer, which is free to use and can be accessed through web browsers, seeks to make teaching contemporary AI methods more accessible to anyone.

Yutong He, Alexander Robey, Naoki Murata, Yiding Jiang, Joshua Williams, George J. Pappas, Hamed Hassani, Yuki Mitsufuji, Ruslan Salakhutdinov, and J. Zico Kolter [6] Prompt engineering for text-to-image (T2I) models is effective but laborious, requiring manual crafting. To address this, PRISM introduces an algorithm for automated prompt generation, leveraging large language model (LLM) capabilities. PRISM identifies human-interpretable and transferable prompts for T2I models, even with black-box access. Inspired by LLM jailbreaking, PRISM iteratively refines prompts for desired concepts.

Yanyu Li, Huan Wang, Qing Jin¹, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, Jian Ren [7] Text-to-image diffusion models are powerful but

computationally expensive, requiring high-end GPUs or cloud-based services. To address this, a generic approach is introduced to run these models on mobile devices in under 2 seconds. This is achieved by optimizing network architecture and step distillation, resulting in an efficient UNet and enhanced training strategies. Experiments on MS-COCO dataset demonstrates improved FID and CLIP scores compared to Stable Diffusion v1.5 with fewer denoising steps.

Agneet Chatterjee, Gabriela Ben Melech, Stan, Estelle Aflalo, Sayak Paul, Dhruba Ghosh, Tejas Gokhale, Ludwig Schmidt, Hannaneh Hajishirzi, Vasudev Lal, Chitta Baral, Yezhou Yang [8] Text-to-image (T2I) models often struggle with faithfully representing spatial relationships from text prompts. To address this, SPRIGHT, a spatially-focused dataset, is introduced by re-captioning 6 million images from existing datasets. Evaluation shows SPRIGHT's effectiveness in capturing spatial relationships, leading to a 22% improvement in spatially accurate image generation with minimal data usage. Training on images with numerous objects also enhances spatial consistency, resulting in state-of-the-art performance on T2I-CompBench. Controlled experiments provide insights into factors influencing spatial consistency in T2I models.

Chunyang Bi, Xin Luo, Sheng Shen, Mengxi Zhang, Huanjing Yue, and Jingyu Yang [9] Diffusion models excel in generative tasks but struggle with global degradation issues in super-resolution tasks. To tackle this, a two-stage degradation-aware framework is introduced. Firstly, unsupervised contrastive learning is employed to capture degradation representations. Secondly, a degradation-aware module is integrated into a simplified Control-Net, allowing adaptation to various degradations. Additionally, degradation-aware features are decomposed into global semantics and local details branches, enhancing the diffusion denoising module for improved target generation.

Divyanshu Mataghare, Shailendra S. Aote, Ramchand Hablani [10] Diffusion models (DMs) excel in generating high-quality images but require extensive computational resources due to their pixel-based optimization process. To address this, latent diffusion models (LDMs) leverage potent pre-trained autoencoders to train DMs efficiently in latent space, achieving a balance between detail preservation and

complexity reduction. Additionally, cross-attention layers are introduced to enhance DMs' flexibility, enabling them to generate high-resolution images from common conditioning inputs like text or bounding boxes. LDMs demonstrate superior performance in tasks such as image inpainting, class-restrictive image blending, text-to-image synthesis, and super-resolution, while requiring less processing power compared to pixel-based DMs.

Syed sha ala, Jeyamurugan, Mohamed faiz ali, Veerasundari [11] proposed a text-to-image technology called Stable Diffusion that will enable billions of individuals to produce beautiful works of art in a few seconds. It can run on GPUs available in consumer gadgets and is an improvement in both performance and quality. It will advance democratic. image generation by enabling both researchers and the general public to use it under various conditions. We anticipate the open ecosystem that will grow up around it as well as new models to really probe the limits of latent space.

Chitwan Saharia, William Chan, Saurabh Saxena [12] text-to-image diffusion model that achieves unprecedented photorealism and language understanding. Leveraging large transformer language models like T5, Imagen encodes text effectively for image synthesis, outperforming models solely based on image diffusion. Imagen sets a new state-of-the-art FID score on the COCO dataset without training on it and exhibits superior image-text alignment. The project also introduces DrawBench, a benchmark for text-to-image models, demonstrating Imagen's superiority over recent methods in sample quality and alignment.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala [13] introduce ControlNet, a neural network architecture designed to incorporate spatial conditioning controls into large, pretrained text-to-image diffusion models. ControlNet utilizes the robust encoding layers of pretrained diffusion models as a backbone to learn various conditional controls, such as edges, depth, segmentation, and human pose. The architecture employs "zero convolutions" to gradually grow parameters from zero, ensuring robust fine-tuning without harmful noise interference. Through extensive testing with Stable Diffusion and various conditioning controls, ControlNet demonstrates robust training with both small and large datasets, potentially expanding the applications of image diffusion models.

Zhiqiu Lin, Deepak Pathak Baiqi Li, Jiayao Li [14] address the challenge of comprehensive evaluation in generative AI due to the lack of effective metrics and standardized benchmarks. They introduce VQAScore, which uses a visual-question-answering (VQA) model to measure alignment between images and complex text prompts. VQAScore outperforms existing metrics across multiple image-text alignment benchmarks, even when computed with off-the-shelf models. Additionally, their in-house model, CLIP-FlanT5, achieves superior performance compared to strong baselines, including those utilizing proprietary models. The authors introduce GenAI-Bench, a challenging benchmark with 1,600 compositional text prompts, enabling researchers evaluate text-to-visual generation models on complex real-world scenarios.

Jun Cheng, Fuxiang Wu, Yanling Tian, Lei Wang, Dapeng Tao [15], the task of text-to-image synthesis poses challenges due to the limited information in textual descriptions, leading to ambiguous and abstract representations. To overcome this, RiFeGAN is proposed, a novel approach for enriching text descriptions to provide additional visual details for image synthesis. RiFeGAN utilizes an attention-based caption matching model to refine compatible candidate captions and self-attentional embedding mixtures to extract features effectively. It then employs multi-captions attentional generative adversarial networks to synthesize images from the enriched captions. Experimental results demonstrate that RiFeGAN effectively generates images from enriched captions and significantly improves synthesis outcomes.

Mr. R. Nanda Kumar, Manoj Kumar M, Hari Hara Sudhan V, Santhosh R [16], the project aims to generate realistic images from textual descriptions using DALL-E, offering a fun and creative way to visualize ideas and concepts. Developed as an Android application using Kotlin and Java, it utilizes natural language prompts to create images, suitable for various purposes like presentations. The application allows users to seamlessly translate imaginative ideas into tangible images of their choice, without compromising on quality. With options to customize the image size based on network capabilities, it provides a user-friendly experience for generating high-quality images with ease.

Akanksha Singh, Ritika Shenoy, Sonam Anekar, Prof. Sainath Patil [17] proposed that Text to image synthesis refers to the method of generating images from

the input text automatically. Deciphering data between picture and text is a major issue in artificial intelligence. Automatic image synthesis is highly beneficial in many ways. Generation of the image is one of the applications of conditional generative models. For generating images, GAN (Generative Adversarial Models) are used. Recent progress has been made using Generative Adversarial Networks (GAN). The conversion of the text to image is an extremely appropriate example of deep learning.

Sean Liu [18] proposed that Text-to-image generation is a field with great potential. It is particularly interesting to us because it programmatically synthesizes one data type into another, generating a photorealist image based on a phrase. One field of application for our model is to aid language learning since children understand the meaning of a phrase better if they see images that correspond to text phrases. In our project, after exploration of several models, we decided the Attentional Generative Adversarial Network model (AttnGAN) would give us the best performance. Then we trained and validated Google captions dataset, fine-tuned parameters, and applied our final model to the learning scenario where we test the quality of images generated from text caption.

Table 1: A Comparative Study of various related works

S.no	Title & Authors	Published journal and Year	Dataset used	Approach	Outcome	Limitations
1.	“eDiff-I: Text-to Image Diffusion Models with an Ensemble of Expert Denoisers”, Yogesh Balaji, Seungjun Nah, Xun Huang	arXiv [2023]	COCO and Visual Genome datasets.	DALL-E 2 and Stable Diffusion, paint with words (enables users to specify the spatial locations of objects)	eDiff-I can serve as a powerful tool for digital artists for content creation and to express their creativity freely	While images generated in the CLIP-text-only setting often contain correct foreground objects, they tend to miss fine-grain details.
2.	“Brush Your Text: Synthesize Any Scene Text on Images via Diffusion Model”, Xinyuan Chen, Yaohui Wang, Yue Lu, Yu Qiao	Proceedings of the AAAI Conference on Artificial Intelligence, 2024 ojs.aaa.org [2023]	Notably and SynthText	It introduces a training-free framework, named Diff-Text. This framework is designed to apply to scene text generation in any language.	By using Diff text it generates the text in different languages by using images taken as input and scanning the text on images to identify the language on it.	It is observed that the model still struggles to generate small scale scene text and achieve precise text color control. Moreover, the generated scene text still occasionally includes unintended textual elements.

S.no	Title & Authors	Published journal and Year	Dataset used	Approach	Outcome	Limitations
3.	“Adding Conditional Control to Text-to-Image Diffusion Models”, Lvmin Zhang, Anyi Rao, and Maneesh Agrawala.	IEEE [2023]	LAION-5B dataset.	This introduces ControlNet, a neural network architecture to add spatial conditioning controls to large, pretrained text to-image diffusion models.	Diverse conditioning datasets show that the ControlNet structure is likely to be applicable to a wider range of conditions.	The training data used is limited in size and diversity. Transferability to new conditioning tasks is uncertain.
4.	“Text-to-Image Generation Using Deep Learning”, Sadia Ramzan, Muhammad Munwar Iqbal and Tehmina Kalsum	MDPI [2022]	Oxford-102 flowers	Conditional GANs were used with recurrent neural networks (RNNs) and convolutional neural networks	The RC-GAN model combines computer vision and natural language processing to generate high-quality images from text descriptions	The model's performance is solely evaluated on the Oxford-102 flowers dataset, and there's a need to test it on a broader range of datasets
5.	“Stable Diffusion Text-Image Generation”, Syed Sha Alam, Jeyamurun, Mohamed Faiz Ali, Veerasundari	IJSREM [2023]	Diffusion model	Diffusion models (DMs) achieve cutting-edge synthesis results on image data and beyond by breaking down the creation of images into a series of denoising autoencoder applications	Latent diffusion models are a quick and easy technique to boost the training and sampling effectiveness of de-noising diffusion models without sacrificing their quality	Diffusion models can be thought of as low-dimensional learning manifolds, generally images, that map points in a high-dimensional latent space(interpretability and accuracy)

S.no	Title & Authors	Published journal and Year	Dataset used	Approach	Outcome	Limitations
6.	“Text To Image Generation Using AI”,Mr. R. Nanda Kumar, Manoj Kumar M, Hari Hara Sudhan V, Santhosh R	IJCRT [2023]	Developed by OpenAI	Utilizes DALL-E for generating images from text prompts, employing a dataset of diverse visual concepts	Highly advanced AI application that uses cutting-edge technology to generate high-quality images from text descriptions	The text doesn't specify how exactly the improvement in image quality will be achieved or what specific techniques will be developed
7.	“Text-to-image Diffusion Models in Generative AI: A Survey” Chenshuang Zhang, Chaoning Zhang, Mengchun Zhang, In So Kweon	ReasearchGate [2015]	Various Datasets	(DDPMs) for generating images from noise through a parameterized Markov chain.aiming to optimize the noise added at each step to reconstruct the original image.	Evolution of text-to-image generation methods, from early attempts like AlignDRAW to recent advancements in diffusion models.	Addressing dataset biases is difficult, resource requirements for training are high, and evaluating model performance remains challenging due to diverse criteria and human subjectivity.
8.	“Text-to-Image Diffusion Model on Mobile Devices within Two Seconds” Yanyu Li, Huan Wang, Qing Jin, Ju Hu	arXiv [2023]	Various Datasets	Latent diffusion model reduces the inference computation and steps by performing the denoising process in the latent space, which is encoded from VAE	Text-to-image model that runs denoising in 1.84 seconds with image quality on par with Stable Diffusion.	Reduce the model size to make it more compatible with various edge devices

S.no	Title & Authors	Published journal and Year	Dataset used	Approach	Outcome	Limitations
9.	“Improving Spatial Consistency in Text-Image Models”, Agneet Chatterjee, Gabriela Ben Melech Stan, Estelle Aflalo, Sayak Paul”.	arXiv [2024]	SPRIGHT dataset	Implemented efficient training techniques leveraging SPRIGHT to achieve state-of-the-art spatially accurate image generation.	Achieved state-of-the-art performance in generating spatially accurate images through efficient training techniques.	Dependency on the quality and diversity of the SPRIGHT dataset, which may limit generalization to unseen data.
10.	“Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, Chitwan Saharia, William Chan, Saurabh Saxena,	arXiv [2022]	COCO dataset and the DrawBench benchmark	Leveraging large transformer language models, such as T5, pretrained on text-only corpora, to encode text for image synthesis.	Using frozen large pretrained language models as text encoders for text-to-image generation, achieving unprecedented photorealism and alignment with text	There has been comparatively less work on social bias evaluation methods
11.	“Text-Image Generation-Stable Diffusion” Unnati kolte, Samprada Kale, Priti Yamkar, Sakshi Deshpande	IJSREM [2023]	MS COCO	User prompts are processed by Flask, which communicates with Stable Diffusion for image creation. Regular model stays current with advancements in text-to image techniques	An interactive system capable of generating high-quality images from textual prompts. between user input and image generation	Reliance on the quality of the trained Stable Diffusion model and the availability of suitable training data.

S.no	Title & Authors	Published journal and Year	Dataset used	Approach	Outcome	Limitations
12.	“Generative Adversarial Text to Image Synthesis”, Scott Reed, Zeynep Akata, Xincheng Yan, Lajanugen Logeswaran	arXiv [2016]	MS COCO and CUB 200-2011 datasets	Leverages generic recurrent neural network architectures for learning discriminative text features and deep convolutional GANs for image generation.	Effectively generates diverse images from detailed text descriptions, showcasing improved synthesis through manifold interpolation regularization. Additionally, the model demonstrates robust generalizability across datasets.	There is a lack of coherence in complex scenes, suggesting avenues for future research to address these challenges and further improve the model's performance.
13.	“Text To Image Generation Using Stable Diffusion“, Divyanshu Mataghare, Shailendra S. Aote, Ramchand Hablani	ECB [2023]	DC-GAN, StackGAN, AttnGAN, and WGAN, without specifying the datasets	Enables improved training and sampling effectiveness without compromising image quality, making it suitable for a range of conditional image synthesis tasks.	Explores text-to-image generation by integrating de-noising diffusion models with latent diffusion models and a cross-attention conditioning mechanism	The sequential sampling process with latent diffusion models is slower compared to GANs, and fine-grained precision in pixel space might be challenging to achieve.

S.no	Title & Authors	Published journal and Year	Dataset used	Approach	Outcome	Limitations
14.	“Automated Black-box Prompt Engineering for Personalized Text-to Image Generation”, Yutong He, Alexander Robey, Naoki Murata, Yiding Jiang	arXiv [2024]	DreamBooth dataset	PRISM algorithm automatically generates human-interpretable and accurate text prompts for text-to-image generative models	By leveraging visual concepts and iterative refinement through LLM in-context learning, it enables the generation of prompts transferable to diverse T2I models	Despite its effectiveness, PRISM may be susceptible to adversarial manipulation or malicious intent, similar to vulnerabilities observed in LLMs.
15	“Diffusion Explainer: Visual Explanation for Text-to-image Stable Diffusion”, Seongmin Lee, Benjamin Hoover, Hendrik Strobelt Zijie J. Wang Sheng, Yun Peng	arXiv [2023]	open source (Diffusion Explainer)	Incorporates animation of random noise refinement and allows users to explore each refinement timestep.	It integrates visual overviews of complex components with detailed explanations, allowing users to explore the impacts of prompts on image generation	Potentially limiting its applicability to understanding other generative AI models or tasks.
16.	Denoising Diffusion Probabilistic Models”, Jonathan Ho, Ajay Jai, Pieter Abbeel	arXiv [2020]	CIFAR10 dataset	Diffusion probabilistic model, which is essentially a parameterized Markov chain pre- trained model	Generates high-quality images and extracting meaningful representations from the data distribution.	It is observed that it lacks competitive log likelihoods compared to other models, potentially indicating a trade-off between capturing

S.no	Title & Authors	Published journal and Year	Dataset used	Approach	Outcome	Limitations
17.	High-Resolution Image Synthesis with Latent Diffusion Models. Robin Rombach ¹ , Andreas Blattmann ¹ , Dominik Lorenz ¹ , Patrick Esse, Bjorn Ommer	arXiv [2022]	MS COCO dataset	Training an autoencoder to create a lower-dimensional and designing an architecture connecting transformers to the DM's UNet backbone for arbitrary token-based conditioning mechanisms	It has the ability to render large, consistent images of 1024*1024 px for densely conditioned tasks such as super-resolution, inpainting, and semantic synthesis.	The sequential sampling process is slower than that of Generative Adversarial Networks (GANs), and they may struggle with tasks requiring ultra-fine precision.
18.	“Towards Real-World Image Super-Resolution via Degradation-Aware Stable Diffusion”, Chunyang Bi,Xin Luo, Sheng Shen,Mengxi Zhang,Huanjing Yue and Jingyu Yang	arXiv [2024]	DIV2K, DIV8K, Flickr2K, FFHQ	Two-stage pipeline: training the Degradation Learner via contrastive learning on low-resolution patches and fine-tuning a Degradation-Aware Adapter	Effectively captures global degradation representations and ensures semantic accuracy through a two-stage process	Provide a high-level overview of the DeeDSR approach without delving into specific technical details.

2.2 EXISTING SYSTEM

The alignDRAW model, a combination of a recurrent variational autoencoder and an alignment model over words, showcased its prowess in generating images based on given textual descriptions. Employing a rich array of attentional mechanisms, this model builds upon the foundation laid by the Deep Recurrent Attention Writer (DRAW), enhancing its capabilities by iteratively drawing patches on a canvas while focusing on the pertinent words within the accompanying description. This iterative process enables the model to gradually refine the image, aligning it more closely with the semantics of the input text. Despite producing images that were characterized by a degree of blurriness and lacked photorealism, alignDRAW demonstrated remarkable generalization abilities. In essence, alignDRAW represents a significant leap forward in bridging the semantic gap between text and images, offering a promising avenue for further exploration in the realm of multimodal learning.

2.3 DISADVANTAGES OF EXISTING SYSTEM

Align draw, while a powerful technique for generating images with diffusion models, does come with some drawbacks. One significant limitation is its computational cost and complexity. Since align draw requires aligning each individual sample drawn from the diffusion process to a reference image, it can be quite resource-intensive, especially when dealing with high-resolution images or large datasets. This process involves iteratively refining the generated images until they closely resemble the reference image, which can require a substantial amount of computational power and time. Moreover, align draw may not perform optimally when dealing with complex datasets with multiple objects or intricate details, as aligning these elements accurately can be challenging. Overall, while align draw can produce high-quality images, its computational demands and limitations in generating diverse outputs should be taken into consideration when applying this technique in practice.

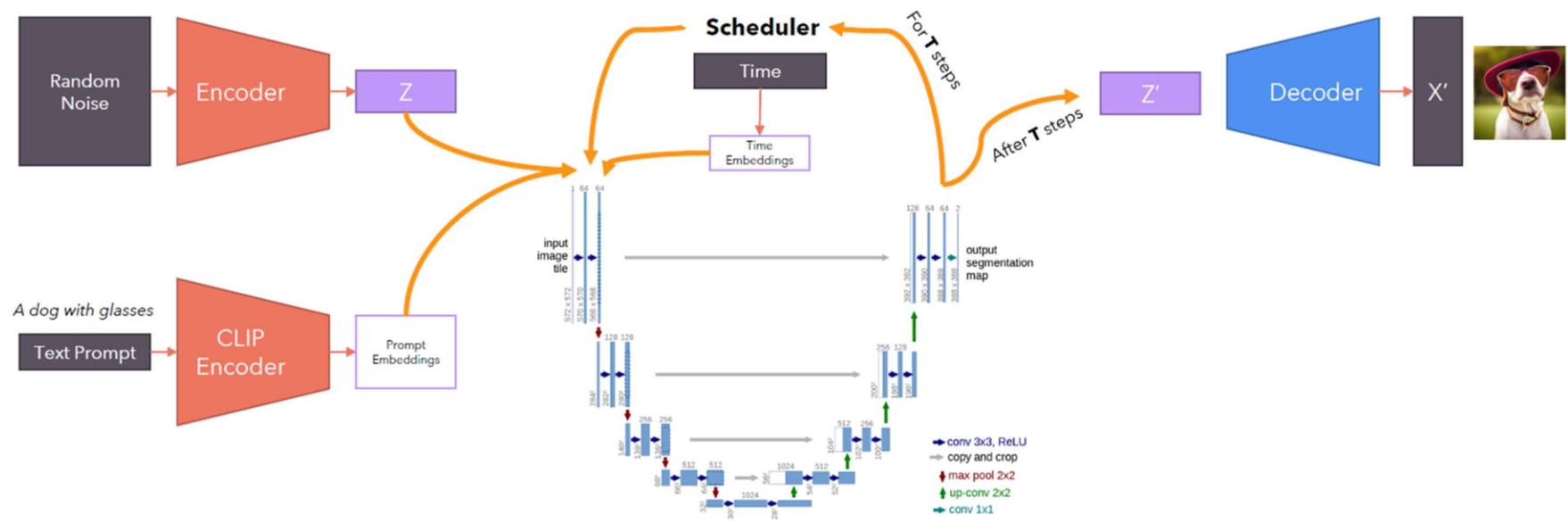


Fig. 2.2: Proposed Architecture

2.5 CONCLUSION

Different papers along with knowledge resources are researched and thus the proposed system is made by incorporating different features from the survey of research papers. It generates the images based on the input text prompt given by the user. The Model has the ability to generate images across various domain.

3. REQUIREMENT ANALYSIS

3.1 INTRODUCTION

Our project focuses on text-to-image generation using stable diffusion, an innovative approach that combines natural language processing and advanced image synthesis techniques. By interpreting text and gradually refining latent representations, our system generates visually compelling images that faithfully reflect the semantics of the input text. Through the integration of insights from various research papers and knowledge resources, we aim to push the boundaries of text-to-image synthesis, offering a novel solution with the potential to revolutionize creative expression and communication in artificial intelligence.

3.2 SOFTWARE REQUIREMENT SPECIFICATION

3.2.1 Software Requirement

The software requirements include the languages, libraries, packages and operating system, and different tools for developing the project. We used python for coding Refer table2 and table 3 for software requirements.

Minimum Software Requirements

Table 2. Software Requirements

Software	Version
Google Colab	3.7
Visual Studio Code Editor	1.76.0
Operating System	64-bit OS, Windows 11, 21H2
Python	3.9.12

Python packages

Python packages we used in our project includes the following table 3. We used the command **pip install “package_name==version”** to download the packages.

Table 3. Python packages

S. No.	Packages used	Description
1.	pytorch	PyTorch is an open-source machine learning library used for text-to-image generation.
2.	numpy	NumPy is a fundamental package for scientific computing, crucial for array operations.
3.	tqdm	tqdm is a fast, extensible progress bar for Python and CLI, useful for tracking tasks' progress.
4.	transformers	Transformers is a library for natural language understanding and generation tasks.
5.	lightning	PyTorch Lightning is a lightweight PyTorch wrapper for high-performance AI research.
6.	pillow	Pillow is a Python Imaging Library (PIL) fork, providing image processing capabilities.
7.	diffusers	The diffusers package facilitates Stable Diffusion model implementations for progressive text-to-image generation. It offers tools for probabilistic generative processes, enhancing image synthesis quality.

Front end packages used:

Table 4. Frontend packages

S. No.	LibrariesUsed	Description
1.	flask	flask is a web framework that promotes rapid development and simple, practical design.
2.	os	In Python, the OS module has functions for dealing with the operating system.

3.2.2 Hardware Requirement

The below Table 5 shows the minimum hardware requirements of our project.

Minimum Hardware Requirements

Table 5. Minimum Hardware requirements

Hardware Requirements	Configuration
Processor	Intel core with i5 configuration or equivalent processors.
RAM	8GB
Hard Disk	256 GB or greater

3.3 NON-FUNCTIONAL REQUIREMENTS

- **Compatibility:** According to the definition, Compatibility is the capacity for two systems to work together without having to be altered to do so. This project is compatible to run on Windows 10,11
- **Capacity:** It can be stated as the capacity of a system refers to the amount of storage it utilizes. This project work i5 processor of 8GB RAM
- **Environment:** It can be stated as all the external and internal forces that exert on your project. This project works on python 3.9.12 environment and higher versions
- **Performance:** The performance of this project is analyzed based on the accurate image it has been generated.
- **Reliability:** It can be stated as the extent to which the software system consistently performs the specified function without failure. In this project, output is analyzed is based on the output image that has been generated.

3.4 CONTENT DIAGRAM OF OUR PROJECT

We divided the whole process into several modules and explained how each module works in this section as shown in figure 3. Firstly it takes the text input, which serves as the starting point for image generation. The text input undergoes preprocessing to extract relevant features and convert it into a format suitable for processing by the model.

Next, the preprocessed text is fed into the stable diffusion-based generative model, where it interacts with latent representations and diffusion processes to generate an initial image. The generated image is then refined iteratively through multiple diffusion steps, gradually incorporating more details and enhancing visual fidelity. Finally, the output of the generative model is the synthesized image, which closely aligns with the semantics of the input text.

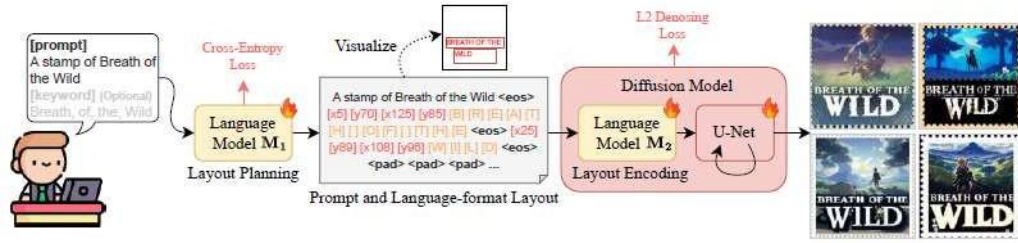


Fig. 3.1: Content diagram of our project

Extracting Text Descriptions: Utilizing text datasets and preprocessing techniques, we retrieve textual descriptions required for image generation.

Data Preprocessing: The extracted text descriptions undergo preprocessing steps, including the removal of special characters, stop words, and extra spaces.

Semantic Labeling: Using advanced techniques like stable diffusion-based generative models, text descriptions are labeled to indicate their semantic content, facilitating subsequent image synthesis.

Extracting Text Descriptions: Utilizing text datasets and preprocessing techniques, we retrieve textual descriptions required for image generation.

Data Preprocessing: The extracted text descriptions undergo preprocessing steps, including the removal of special characters, stop words, and extra spaces.

Semantic Labeling: Using advanced techniques like stable diffusion-based generative models, text descriptions are labeled to indicate their semantic content, facilitating image synthesis.

Feature Representation: Word and sentence embedding techniques are applied to convert the preprocessed text descriptions into numerical vectors, enabling easier model training.

Image Synthesis Model Development: We develop image synthesis models utilizing stable diffusion-based techniques, leveraging the preprocessed text vectors to generate high-quality and diverse images that correspond to the textual descriptions.

Website Integration: The trained image synthesis models are integrated into a website frontend. Users can input text descriptions, and the models generate corresponding images.

3.5 FLOWCHARTS OF OUR PROJECT

The figure 4 describes the flow of the project begins with input text descriptions provided by the user. These descriptions undergo preprocessing to standardize the data, including steps such as removing special characters and stop words. Following preprocessing, the text descriptions are semantically labeled to guide the image generation process. Next, the labeled text descriptions are converted into numerical vectors using word and sentence embedding techniques, facilitating easier processing by the image generation model. The heart of the flowchart involves the application of stable diffusion-based generative models to synthesize images from the numerical representations of the text descriptions. Finally, it generates the output, which consists of the synthesized images corresponding to the input text descriptions.

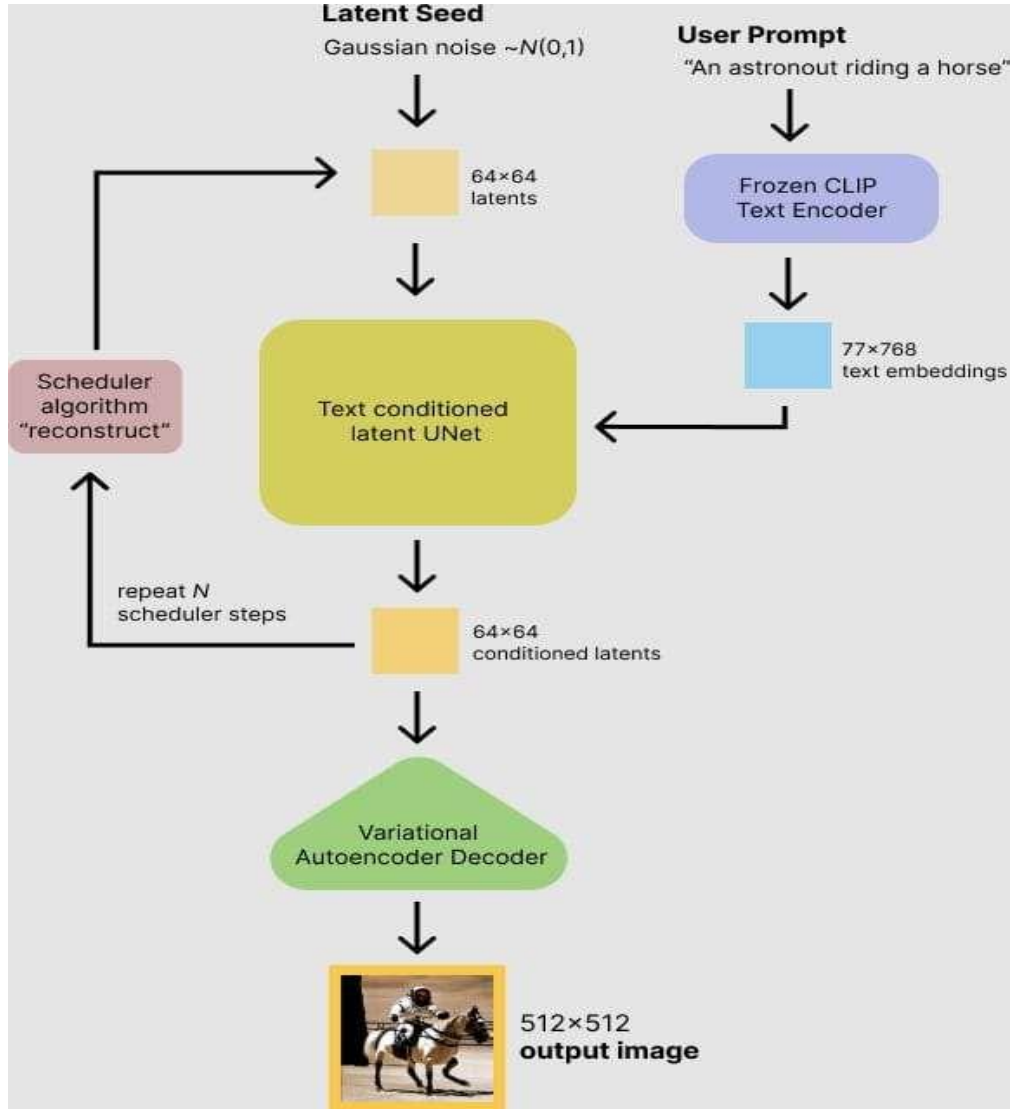


Fig. 3.2: Flowchart of the project

3.6 ALGORITHMS

The stable diffusion process involves the use of U-Net architecture, a convolutional neural network (CNN) commonly employed in image processing tasks. U-Net is utilized to facilitate diffusion process, which simulates the information across the image space.

The architecture of U-Net is characterized by its U-shaped design, which consists of an encoder pathway followed by a decoder pathway. The encoder pathway progressively down samples the input image to capture high-level features, while the

decoder pathway up samples the features to generate the final segmentation mask.

At each level of the encoder pathway, the input image is convolved with multiple filters to extract feature maps. Between each convolutional layer, rectified linear unit (ReLU) activation functions are applied to introduce non-linearity, and max-pooling. In the decoder pathway, the feature maps from the encoder pathway are upsampled through transpose convolutional layers or upsampling operations to reconstruct the spatial resolution of the original image

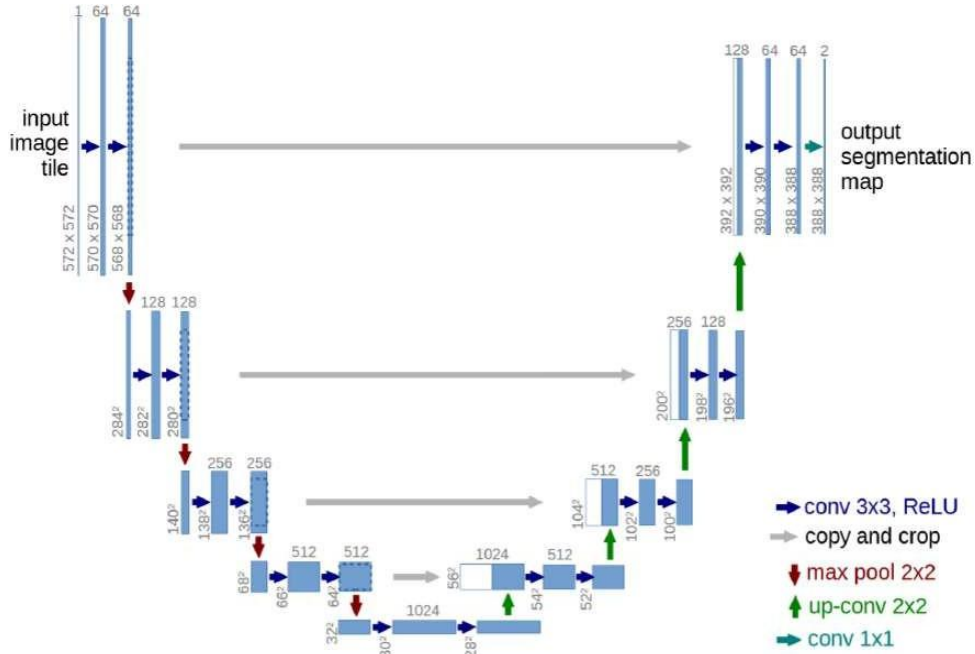


Fig. 3.3: U-Net Architecture

The training and sampling process for DDPMs involves conditioning on initial noise, predicting noise at each timestep, and optimizing a simplified loss term to generate high-quality images.

The process is similar to Variational Autoencoders (VAEs) in terms of optimization through negative log-likelihood. It introduces the Evidence Lower Bound (ELBO) comprised of reconstruction, Gaussian proximity, and denoising discrepancy terms, emphasizing the tractability of denoising steps by conditioning on initial noise. By sampling each timestep conditioned on the initial noise, DDPMs incrementally generate images. A parameterization trick involving a neural network facilitates the

prediction of noise at each timestep, simplifying the loss term to focus on noise prediction.

ALGORITHM 1 TRAINING

```

1: repeat
2:  $x_0 \sim q(x_0)$ 
3:  $t \sim \text{Uniform}(\{1, \dots, T\})$ 
4:  $\epsilon \sim \mathcal{N}(0, I)$ 
5: Take gradient descent step on
   
$$\|\epsilon - \epsilon_\theta(\sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t} \epsilon, t)\|^2$$

6: until converged

```

ALGORITHM 2 SAMPLING

```

1:  $x_T \sim \mathcal{N}(0, I)$ 
2: for  $t = T, \dots, 1$  do
3:  $z \sim \mathcal{N}(0, I)$  if  $t > 1$ , else  $z = 0$ 
4:  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} (x_t - \frac{1 - \alpha_t}{\sqrt{1 - \alpha_t}} \epsilon_\theta(x_t, t)) + \sigma_t z$ 
5: end for
6: return  $x_0$ 

```

Diffusion Models consist of two processes: forward diffusion and reverse diffusion.

FORWARD PROCESS:

The forward diffusion process consists of iteratively adding Gaussian noise. By using the closed-form formula in just one step, we remove the need of iterating.

Given a data-point x_0 sampled from the real data distribution $q(x)$ ($x_0 \sim q(x)$), one can define a forward diffusion process by adding noise. Specifically, at each step of the Markov chain we add Gaussian noise with variance β_t to x_{t-1} , producing a new latent variable x_t with distribution $q(x_t|x_{t-1})$. This diffusion process can be formulated as follows:

$$\begin{aligned}
 q(x_{1:T}|x_0) &\rightarrow \prod_{t=1}^T q(x_t|x_{t-1}) \quad , \\
 q(x_t|x_{t-1}) &\rightarrow \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I)
 \end{aligned}$$

REVERSE PROCESS:

The reverse diffusion process involves utilizing a neural network to approximate the denoising process. The role of this neural network is to predict the whole noise present in the image at a certain timestep t . Then, by comparing this prediction with the real noise added to the image, the network can be trained. This process is iterative, step by step, until recovering the original image.

$$\begin{aligned} p_{\theta}(x_{0:T}) &\rightarrow p(x_T) \prod_{t=1}^T p_{\theta}(x_{t-1}|x_t) , \\ p_{\theta}(x_{t-1}|x_t) &\rightarrow \mathcal{N}(x_{t-1}; \mu_{\theta}(x_t, t), \Sigma_{\theta}(x_t, t)) \end{aligned}$$

3.7 CONCLUSION

This approach merges the power of convolutional neural networks for feature extraction and the integration of U-Net architecture into the stable diffusion process revolutionizes image processing by efficiently capturing features through its encoder-decoder design.

4. DESIGN

4.1 INTRODUCTION

Software Design is a process of planning a new or modified system. The design step produces the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Analysis specifies what a new or modified system does. The design specifies how to accomplish the same. Design is essentially a bridge between requirement specification and the final solution satisfying the requirement. It is a blueprint or a solution for the system. The design step produces a data design, an architectural design and a procedural design. The design process for a software system has two levels. At the first level the focus is on depending on which modules are needed for the system, the specification of these modules and how the modules should be interconnected. This is what is called system designing or top-level design. In the second level, the internal design of the modules, or how the specification should be interconnected.

Top Level Design

It is the first level of the design which produces the system design, which defines the components needed for the system, and how the components interact with each other. It focuses on depending on which modules are needed for the system; the specifications of this module should be interconnected.

Logic Design

The logical design of this application shows the major features and how they are related to one another. The detailed specifications are drawn based on user requirements. The outputs, inputs and relationship between the variables are designed in this phase.

4.2 UML DIAGRAMS

UML stands for Unified Modelling Language. UML is a standard language for specifying, visualizing, constructing, and documenting the artefacts of software

systems. UML is different from the other common programming languages like C++, Java, COBOL etc. It is designed to enable users to develop an expressive and ready-to-use visual modelling language. In addition, it supports high-level development concepts such as frameworks, patterns and collaborations. UML can be described as a general-purpose visual modelling language to visualise, specify, construct and document software systems. But it is not limited within this boundary. To be more precise, UML is a pictorial language used to make software blueprints. UML has a direct relation with object-oriented analysis and design. After some standardisation, UML became an OMG (ObjectManagement Group) standard.

The UML is a Language for

- Visualising
- Specifying
- Constructing
- Documenting

Basic Building Blocks of UML

The basic building blocks in UML are things and relationships. These are combined in different ways following different rules to create different types of diagrams. In UML there are Eight types of diagrams, below is a list and brief description of them.

Components of the UML

The UML consists of several graphical elements that combine to form diagrams. Because it's a language, the UML has rules for combining these elements. The purpose of the diagrams is to present multiple views of the system, and this set of multiple views is called a Model. A UML Model of a system is something like a scale model of a building. UML model describes what a system is supposed to do. It doesn't tell how to implement the system. These are the artefacts of a software-intensive system. The abbreviation for UML is Unified Modelling Language and is being brought off designed to make sure that the existing ER Diagrams that do not serve the purpose will be replaced by these UML Diagrams where these languages as their own set of

Diagrams. Some of the Diagrams that help with the Diagrammatic Approach for Object-Oriented Software Engineering are:

- Class Diagrams
- Use Case Diagrams
- Sequence Diagrams
- State chart Diagrams
- Activity Diagrams

Using the above-mentioned diagram we can show the entire system regarding the working of the system or the control flow and sequence of flow the state of the system

4.2.1 CLASS DIAGRAM

A Class is a category or group of things that have similar attributes and common behaviour. A Rectangle is the icon that represents the class it is divided into three areas. The uppermost area contains the name, the middle area contains the attributes and the lowest areas show the operations. Class diagrams provide the representation that developers work from. The classes in a Class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are represented with boxes which contain three parts:

1. The top part contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
2. The middle part contains the attributes of the class. They are left-aligned and the first letter is lowercase.
3. The bottom part contains the methods the class can execute. They are also left-aligned and the first letter is lowercase. In the design of a system, several classes are identified and grouped in a class diagram which helps to determine the static relations between those described objects. With detailed modelling, the classes of the conceptual design are often split into several subclasses.

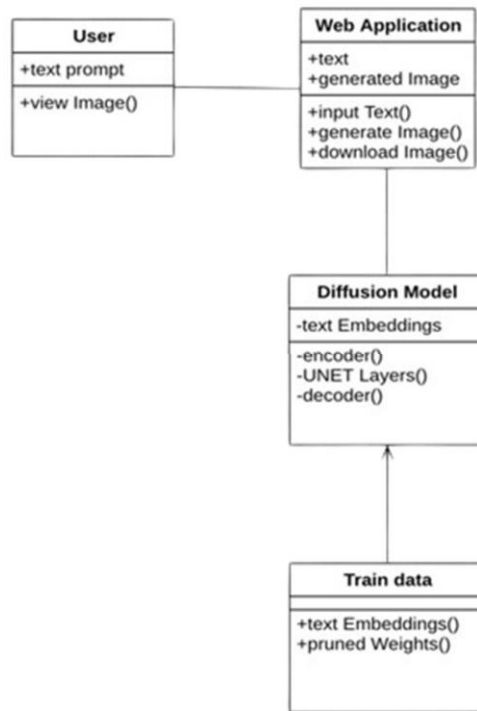


Fig. 4.1: Class Diagram

4.2.2 USE CASE DIAGRAM

A use case is a description of system behaviour from an understanding point. For system developers, this is a valuable tool: it's a tried-and-true technique for gathering system requirements from a user's point of view. That is important if the goal is to build a system that real people can use. A little stick figure is used to identify an actor the ellipse represents a use case. Here, we have two actors namely developer and user. The developer is involved in all the steps whereas the user is only interested in visualising the result, testing the model and displaying the result.

Use cases: A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.

Actors: An actor is a person, Organisation, or external system that plays a role in one or more interactions with your system. Actors are drawn as stick figures.

Associations: Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever an actor is involved with an interaction described by a use case. Associations are modelled as lines connecting use cases and actors to one another, with an optional arrowhead on one end of the line.

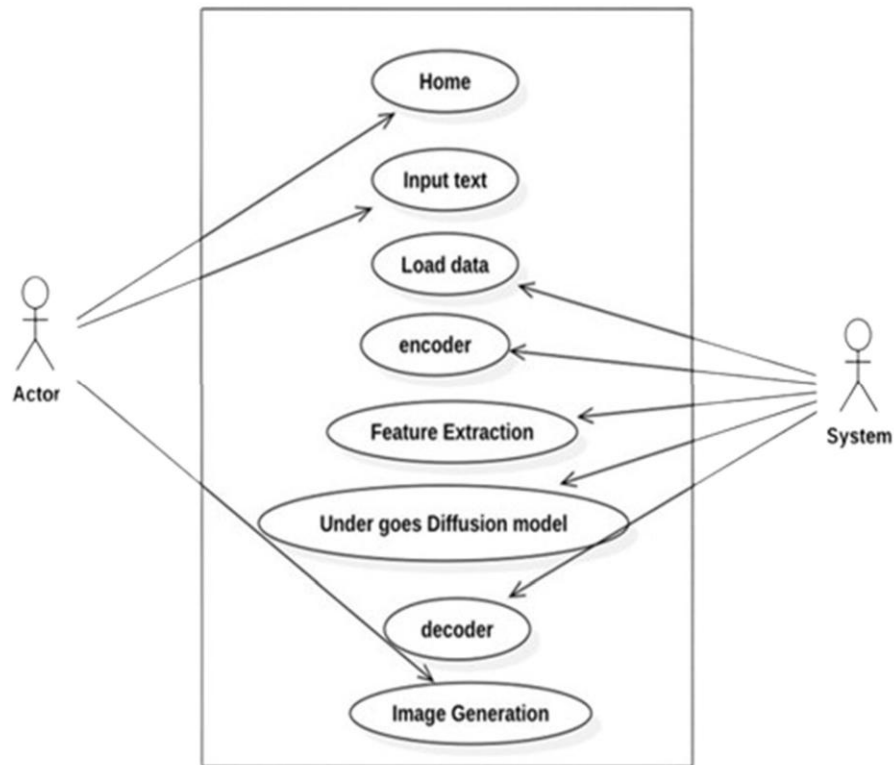


Fig. 4.2: Use Case Diagram

4.2.3 SEQUENCE DIAGRAMS

In a functioning system, objects interact with one another and these interactions occur over time. The UML Sequence Diagrams show the time-based dynamics of the interaction. The sequence diagrams consist of objects represented in the use always named rectangles (If the name is underlined), messages represented as solid line arrows and time represented as a vertical progression. A narrow rectangle on an object's lifetime represents activation- an exchange of one of the operations of that object.

Actor: An actor is a character who interacts with the subject and plays a part. It isn't covered by the system's capabilities. It depicts a role in which human users interact with external devices or subjects.

Message: Message denotes the interactions between the messages. They are sequential ordering the timeline.

Call message: By defining a specific communication between the interaction's lifelines, it shows that the target lifeline has invoked an operation.

Return Message: It specifies a specific communication between the interaction lifelines, which reflects the flow of data from the receiver of the associated caller message.

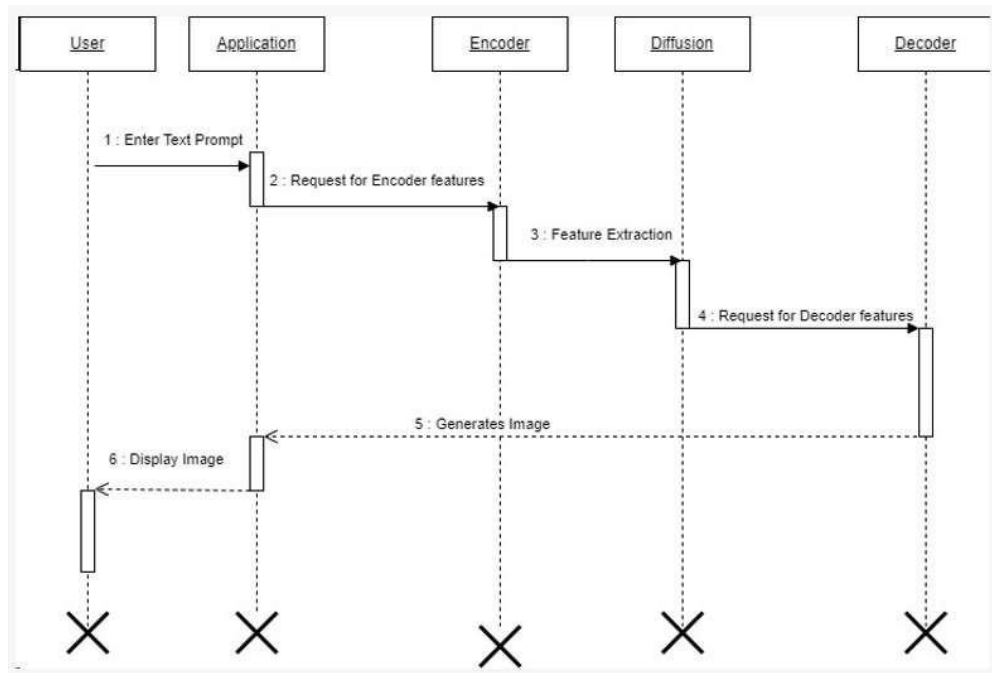


Fig. 4.3: Sequence Diagram

4.2.4 ACTIVITY DIAGRAM

The activity diagram is another important diagram in UML to describe the dynamic aspects of the system. An activity diagram is a new chart to represent the flow from one activity to another activity. The activity Can be described as an operation of the system. So the control now is drawn from one operation to another, This flow can

be sequential, branched or concurrent. Activity diagrams deal with all types Of flow control by using different elements like fork, join etc.

PURPOSE: The basic purposes of activity diagrams are similar to the other four diagrams. Other four diagrams are used to show the message flow from one object to another but the activity diagram is used to show the message flow from one activity to another. Activity is a particular operation of the system. It captures the dynamic behaviour of the system.

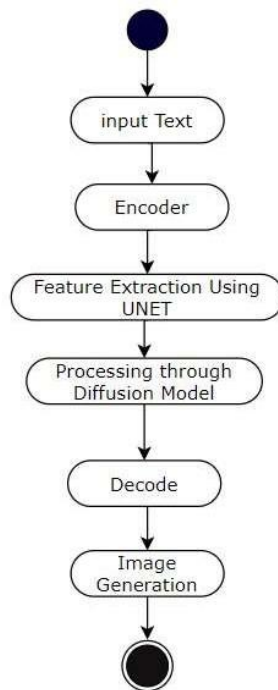


Fig. 4.4: Activity Diagram

4.3 CONCLUSION

UML diagrams are not only made for developers but also for business users, common people, and anybody interested in understanding the system. The system can be a software or non-software system. Thus it must be clear that UML is not a development method rather it accompanies processes to make it a successful system. In conclusion, the goal of UML can be defined as a simple modelling mechanism to model all possible practical systems in today's complex environment.

5. IMPLEMENTATION AND RESULTS

5.1 INTRODUCTION

The project's implementation stage is when the theoretical design is translated into a workable system. As a result, it can be seen as the most crucial stage in ensuring the success of a new system and giving the user confidence that the system will work and be effective. The implementation step entails meticulous planning, research of the existing system and its implementation limitations, designing of changeover methods, and evaluation of changeover methods.

5.2 EXPLANATION OF KEY FUNCTIONS

Sentiment Analysis:

Natural language processing, statistics, and text analysis are used in sentiment analysis to study client sentiment.

Fine-grained Textual Description Analysis:

This involves analyzing the input text to understand the polarity of sentiment expressed, such as positive, negative, or neutral. Precise results can be achieved regarding the sentiment conveyed in the text, which helps in guiding the image generation process.

Software Design Front-End Design

HTML

Hypertext markup language (HTML) language for texts intended to be viewed in a web browser. Technologies such as Cascading Style Sheets (CSS) and programming languages like JavaScript can help.

CSS

CSS, or Cascading Style Sheets, is a simple design language intended to make the process of making web pages presentable easier. Styles can be applied to web pages

using CSS. More crucially, CSS allows you to do so without relying on the HTML.

Flask

Flask is a lightweight and flexible web application framework for Python. It is designed to be simple, easy to use, and extensible, making it an ideal choice for building web applications and APIs.

app.py

In Flask applications, the `app.py` file typically serves as the entry point for the application, where the Flask application object is created and configured. This file contains the necessary setup to initialize the Flask application, define routes, and configure various aspects of the application.

5.3 MODEL BUILDING

5.3.1 STABLE DIFFUSION MODEL

- 1) **Data Collection and Preprocessing:** Gather a large dataset of text-image pairs. Ensure that the text descriptions are clear and concise and correspond accurately to the images. Preprocess the data, including tokenization of text and resizing and normalization of images.
- 2) **Model Architecture Selection:** Choose an appropriate architecture for both the text and image processing components. For text, you might use recurrent neural networks (RNNs), transformers, or a combination of both. For images, convolutional neural networks (CNNs) are commonly used.
- 3) **Multimodal Fusion Mechanism:** Design a fusion mechanism to combine information from the text and image modalities effectively. This could involve concatenation, attention mechanisms, or other fusion strategies.
- 4) **Training Objective:** Define a suitable training objective for the model. This could involve maximizing the likelihood of generating the correct image given the input text, or it could involve learning a joint embedding space where text and image representations are semantically aligned.

- 5) **Training Procedure:** Train the model using the collected dataset. Use techniques such as curriculum learning, where the model is initially trained on easier examples before being exposed to more complex ones, to stabilize training. Employ regularization techniques like dropout or weight decay to prevent overfitting.
- 6) **Evaluation Metrics:** Define appropriate evaluation metrics to assess the performance of the model. This could include perceptual metrics like Inception Score or Frechet Inception Distance for image quality, as well as semantic metrics to ensure the generated images accurately reflect the input text.
- 7) **Fine-tuning and Iterative Improvement:** After the initial training, fine-tune the model on specific tasks or domains if necessary. Continuously monitor performance metrics and iterate on the model architecture, training procedure, and data collection to improve overall performance.
- 8) **Testing and Deployment:** Once satisfied with the model's performance, evaluate it on a separate test set to ensure generalization. Deploy the model in a production environment, considering factors such as computational efficiency and scalability.

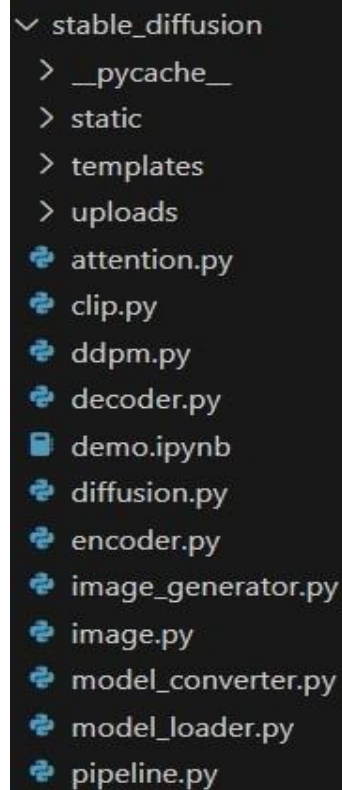
5.3.2 HUGGING FACE MODEL

In this project we have also used hugging face models to generate the Images. Hugging Face is a machine learning and data science platform and community that helps users build, deploy and train machine learning models. This community has several features. The users can access Models, Datasets to build and develop their applications. It has two main libraries that provide access to pre-trained models. Transformers and Diffusers. Diffusers can handle image-based tasks, such as image synthesis, image editing, and image captioning.

5.3.3 INTEGRATION OF FRONTEND WITH BACKEND CODE

Sample Code

The project is designed with the help of Flask framework. In this Project file is named as `stable_diffusion` and the app name is `image`



```

▼ stable_diffusion
  > __pycache__
  > static
  > templates
  > uploads
  attention.py
  clip.py
  ddpm.py
  decoder.py
  demo.ipynb
  diffusion.py
  encoder.py
  image_generator.py
  image.py
  model_converter.py
  model_loader.py
  pipeline.py

```

Screen 1: Project source code directory

Backend code

image.py- flask application that take a web request and return a response. The actual logic of the website will be written in this python file. This file includes all the backend process.

```

from flask import Flask, render_template, request, send_file, send_from_directory
from image_generator import generate_image
import os
image = Flask(__name__)
UPLOAD_FOLDER = 'uploads'

```

```

image.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
os.makedirs(image.config['UPLOAD_FOLDER'], exist_ok=True)
@image.route('/')
def home():
    return render_template('index.html')
@image.route('/generate_image', methods=['POST'])
def generate_and_serve_image():
    text_prompt = request.form['text_prompt']
    generated_image = generate_image(text_prompt)
    image_path=os.path.join(image.config['UPLOAD_FOLDER'],
'generated_image.png')
    generated_image.save(image_path)
    return send_file(image_path, mimetype='image/png')
@image.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory(image.config['UPLOAD_FOLDER'], filename)
if __name__ == '__main__':
    image.run(debug=True)
image_generator.py
import model_loader
import pipeline
from PIL import Image
from transformers import CLIPTokenizer
import torch
def generate_image(prompt):
    DEVICE = "cpu"
    ALLOW_CUDA = True
    ALLOW_MPS = False
    if torch.cuda.is_available() and ALLOW_CUDA:
        DEVICE = "cuda"
    elif (torch.has_mps or torch.backends.mps.is_available()) and ALLOW_MPS:
        DEVICE = "mps"
    print(f"Using device: {DEVICE}")
    tokenizer = CLIPTokenizer("../data/vocab.json", merges_file="../data/merges.txt")

```

```

model_file = "../data/v1-5-pruned-emaonly.ckpt"
models=model_loader.preload_models_from_standard_weights(model_file,
DEVICE)
uncond_prompt = ""
do_cfg = True
cfg_scale = 8
input_image = None
image_path = "../images/dog.jpg"
strength = 0.9
sampler = "ddpm"
num_inference_steps = 50
seed = 42
# Generate image
output_image = pipeline.generate(
    prompt=prompt,
    uncond_prompt=uncond_prompt,
    input_image=input_image,
    strength=strength,
    do_cfg=do_cfg,
    cfg_scale=cfg_scale,
    sampler_name=sampler,
    n_inference_steps=num_inference_steps,
    seed=seed,
    models=models,
    device=DEVICE,
    idle_device="cpu",
    tokenizer=tokenizer, )
output_image_pil = Image.fromarray(output_image)
return output_image_pil

```

encoder.py

```

import torch
from torch import nn
from torch.nn import functional as F
from decoder import VAE_AttentionBlock, VAE_ResidualBlock

```

```

class VAE_Encoder(nn.Sequential):
    def __init__(self):
        super().__init__(
            nn.Conv2d(3, 128, kernel_size=3, padding=1),
            VAE_ResidualBlock(128, 128),
            VAE_ResidualBlock(128, 128),
            nn.Conv2d(128, 128, kernel_size=3, stride=2, padding=0),
            VAE_ResidualBlock(128, 256),
            VAE_ResidualBlock(256, 256),
            nn.Conv2d(256, 256, kernel_size=3, stride=2, padding=0),
            VAE_ResidualBlock(256, 512),
            VAE_ResidualBlock(512, 512),
            nn.Conv2d(512, 512, kernel_size=3, stride=2, padding=0),
            VAE_ResidualBlock(512, 512),
            VAE_ResidualBlock(512, 512),
            VAE_ResidualBlock(512, 512),
            VAE_AttentionBlock(512),
            VAE_ResidualBlock(512, 512),
            nn.GroupNorm(32, 512),
            nn.SiLU(),
            nn.Conv2d(512, 8, kernel_size=3, padding=1),
            nn.Conv2d(8, 8, kernel_size=1, padding=0),
        )
    def forward(self, x, noise):
        for module in self:
            if getattr(module, 'stride', None) == (2, 2):
                x = F.pad(x, (0, 1, 0, 1))
            x = module(x)
        mean, log_variance = torch.chunk(x, 2, dim=1)
        log_variance = torch.clamp(log_variance, -30, 20)
        variance = log_variance.exp()
        stdev = variance.sqrt()
        x = mean + stdev * noise
        x *= 0.18215

```



```
return x
```

diffusion.py

```
import torch
from torch import nn
from torch.nn import functional as F
from attention import SelfAttention, CrossAttention
class TimeEmbedding(nn.Module):
    def __init__(self, n_embd):
        super().__init__()
        self.linear_1 = nn.Linear(n_embd, 4 * n_embd)
        self.linear_2 = nn.Linear(4 * n_embd, 4 * n_embd)
class UNET_ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, n_time=1280):
        super().__init__()
        self.groupnorm_feature = nn.GroupNorm(32, in_channels)
        self.conv_feature=nn.Conv2d(in_channels,out_channels,kernel_size=3,
padding=1)
        self.linear_time = nn.Linear(n_time, out_channels)
        self.groupnorm_merged = nn.GroupNorm(32, out_channels)
        self.conv_merged=nn.Conv2d(out_channels,out_channels,kernel_size=3,
padding=1)
        if in_channels == out_channels:
            self.residual_layer = nn.Identity()
        else:
            self.residual_layer=nn.Conv2d(in_channels,out_channels,kernel_size=1,
padding=0)
    def forward(self, feature, time):
        residue = feature
        feature = self.groupnorm_feature(feature)
        feature = F.silu(feature)
        feature = self.conv_feature(feature)
        time = F.silu(time)
        time = self.linear_time(time)
        merged = feature + time.unsqueeze(-1).unsqueeze(-1)
```

```

merged = self.groupnorm_merged(merged)
merged = F.silu(merged)
merged = self.conv_merged(merged)
return merged + self.residual_layer(residue)

```

```

class UNET_AttentionBlock(nn.Module):

```

```

    def __init__(self, n_head: int, n_embd: int, d_context=768):
        super().__init__()
        channels = n_head * n_embd
        self.groupnorm = nn.GroupNorm(32, channels, eps=1e-6)
        self.conv_input = nn.Conv2d(channels, channels, kernel_size=1, padding=0)
        self.layernorm_1 = nn.LayerNorm(channels)
        self.attention_1 = SelfAttention(n_head, channels, in_proj_bias=False)
        self.layernorm_2 = nn.LayerNorm(channels)
        self.attention_2 = CrossAttention(n_head, channels, d_context, in_proj_bias=False)
        self.layernorm_3 = nn.LayerNorm(channels)
        self.linear_geglu_1 = nn.Linear(channels, 4 * channels * 2)
        self.linear_geglu_2 = nn.Linear(4 * channels, channels)
        self.conv_output = nn.Conv2d(channels, channels, kernel_size=1, padding=0)

    def forward(self, x, context):
        residue_long = x
        x = self.groupnorm(x)
        x = self.conv_input(x)
        n, c, h, w = x.shape
        x = x.view((n, c, h * w))
        x = x.transpose(-1, -2)
        residue_short = x
        x = self.layernorm_1(x)
        x = self.attention_1(x)
        x += residue_short
        residue_short = x
        x = self.layernorm_2(x)
        x = self.attention_2(x, context)
        x += residue_short

```

```

residue_short = x
x = self.layer_norm_3(x)
x, gate = self.linear_geglu_1(x).chunk(2, dim=-1)
x = x * F.gelu(gate)
x = self.linear_geglu_2(x)
x += residue_short
x = x.transpose(-1, -2)
x = x.view((n, c, h, w))
return self.conv_output(x) + residue_long

```

```

class Upsample(nn.Module):

```

```

    def __init__(self, channels):
        super().__init__()
        self.conv = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
    def forward(self, x):
        x = F.interpolate(x, scale_factor=2, mode='nearest')
        return self.conv(x)

```

```

class SwitchSequential(nn.Sequential):

```

```

    def forward(self, x, context, time):
        for layer in self:
            if isinstance(layer, UNET_AttentionBlock):
                x = layer(x, context)
            elif isinstance(layer, UNET_ResidualBlock):
                x = layer(x, time)
            else:
                x = layer(x)
        return x

```

```

class UNET(nn.Module):

```

```

    def __init__(self):
        super().__init__()
        self.encoders = nn.ModuleList([
            SwitchSequential(nn.Conv2d(4, 320, kernel_size=3, padding=1)),
            SwitchSequential(UNET_ResidualBlock(320, 320), UNET_AttentionBlock(8, 40)),
            SwitchSequential(UNET_ResidualBlock(320, 320), UNET_AttentionBlock(8, 40)),

```

```

SwitchSequential(nn.Conv2d(320, 320, kernel_size=3, stride=2, padding=1)),
SwitchSequential(UNET_ResidualBlock(320, 640), UNET_AttentionBlock(8, 80)),
SwitchSequential(UNET_ResidualBlock(640, 640), UNET_AttentionBlock(8, 80)),
    self.bottleneck = SwitchSequential(
        UNET_ResidualBlock(1280, 1280),
        UNET_AttentionBlock(8, 160),
        UNET_ResidualBlock(1280, 1280),
    )
self.decoders = nn.ModuleList([
    SwitchSequential(UNET_ResidualBlock(2560, 1280)),
    SwitchSequential(UNET_ResidualBlock(2560, 1280)),
    SwitchSequential(UNET_ResidualBlock(2560, 1280), Upsample(1280)),
    SwitchSequential(UNET_ResidualBlock(2560, 1280),
UNET_AttentionBlock(8, 160)),
SwitchSequential(UNET_ResidualBlock(960, 640), UNET_AttentionBlock(8, 80),
Upsample(640)),
SwitchSequential(UNET_ResidualBlock(960, 320), UNET_AttentionBlock(8, 40)),
SwitchSequential(UNET_ResidualBlock(640, 320), UNET_AttentionBlock(8, 40)),
    SwitchSequential(UNET_ResidualBlock(640, 320),
UNET_AttentionBlock(8, 40)),
])
def forward(self, x, context, time):
    skip_connections = []
    for layers in self.encoders:
        x = layers(x, context, time)
        skip_connections.append(x)
    x = self.bottleneck(x, context, time)
    for layers in self.decoders:
        x = torch.cat((x, skip_connections.pop()), dim=1)
        x = layers(x, context, time)
    return x
class UNET_OutputLayer(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()

```

```

self.groupnorm = nn.GroupNorm(32, in_channels)
self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)

def forward(self, x):
    x = self.groupnorm(x)
    x = F.silu(x)
    x = self.conv(x)
    return x

class Diffusion(nn.Module):
    def __init__(self):
        super().__init__()
        self.time_embedding = TimeEmbedding(320)
        self.unet = UNET()
        self.final = UNET_OutputLayer(320, 4)

    def forward(self, latent, context, time):
        time = self.time_embedding(time)
        output = self.unet(latent, context, time)
        output = self.final(output)
        return output

```

decoder.py

```

import torch
from torch import nn
from torch.nn import functional as F
from attention import SelfAttention
class VAE_AttentionBlock(nn.Module):
    def __init__(self, channels):
        super().__init__()
        self.groupnorm = nn.GroupNorm(32, channels)
        self.attention = SelfAttention(1, channels)

    def forward(self, x):
        residue = x
        x = self.groupnorm(x)

```

```

n, c, h, w = x.shape
x = x.view((n, c, h * w))
x = x.transpose(-1, -2)
x = self.attention(x)
x = x.transpose(-1, -2)
x = x.view((n, c, h, w))
x += residue
return x

class VAE_ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.groupnorm_1 = nn.GroupNorm(32, in_channels)
        self.conv_1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
        self.groupnorm_2 = nn.GroupNorm(32, out_channels)
        self.conv_2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
        if in_channels == out_channels:
            self.residual_layer = nn.Identity()
        else:
            self.residual_layer = nn.Conv2d(in_channels, out_channels, kernel_size=1,
padding=0)

class VAE_Decoder(nn.Sequential):
    def __init__(self):
        super().__init__(
            nn.Conv2d(4, 4, kernel_size=1, padding=0),
            nn.Conv2d(4, 512, kernel_size=3, padding=1),
            VAE_ResidualBlock(512, 512),
            VAE_AttentionBlock(512),
            VAE_ResidualBlock(512, 512),
            VAE_ResidualBlock(512, 512),
            VAE_ResidualBlock(512, 512),
            VAE_ResidualBlock(512, 512),
            nn.Upsample(scale_factor=2)
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            VAE_ResidualBlock(512, 256),

```

```

        VAE_ResidualBlock(256, 256),
        VAE_ResidualBlock(256, 256),
        nn.Upsample(scale_factor=2),
        nn.Conv2d(256, 256, kernel_size=3, padding=1),
        nn.GroupNorm(32, 128),
        nn.SiLU(),
        nn.Conv2d(128, 3, kernel_size=3, padding=1),
    )

    def forward(self, x):
        x /= 0.18215
        for module in self:
            x = module(x)
        return x

```

Frontend code

All the frontend templates are in templates directory:

homepage.html- This is the parent html page of our all templates ,where navbar functionality will extends Home, About, Models, Prompts pages.

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Personal Portfolio Website </title>
    <link rel="stylesheet" href="home.css">
    <script
                                src="https://kit.fontawesome.com/942cab3b07.js"
crossorigin="anonymous"></script>
  </head>
  <body>
    <div id="header">
      <div class="container">
        <nav>
          <ul>

```

```

        <li><a href="#header">Home</a></li>
        <li><a href="#about">About</a></li>
        <li><a href="#models">Models</a></li>
        <li><a href="#services">Prompts</a></li>
        <!--<li><a href="/GenAI/demo.html">Tools </a></li>-->
    </ul>
    </nav>

    <div class="header-text">
        <p></p>
        <h1><span>Text to Image<br> Generation </span><br>using GEN
AI</h1>
    </div>
</div>

<div id="about">
    <div class="container">
        <div class="row">
            <div class="about-col-1">
                <br><br><br>
                
            </div>
            <div class="about-col-2">
                <br>
                <h1 class="sub-title">Stable Diffusion</h1><br>
                <p >
                    Images play a pivotal role in conveying information, emotions, and ideas. With
                    the exponential growth of textual data across various platforms, there came different
                    models for image generation, such as Generative Adversarial Networks (GANs) and
                    Variational Autoencoders (VAEs) which had limitations to produce high-quality,
                    diverse images that accurately reflected textual descriptions. The proposed Stable
                    Diffusion model represents a significant advancement in text-to-image generation.
                    capabilities of image generation, thus pushing the boundaries of text-to-image synthesis
                    within the field of artificial intelligence.
                </p>
            </div>
        </div>
    </div>

```



```

<!----Models-->
<div id="models">
  <div class="container">
    <br><br><br><br><br>
    <h1 class="sub-title">Models used in generation of images</h1>
    <div class="models-list">
      <div class="card-container">
        <div class="card text-center">
          <div class="title">
            <h2>Stable Diffusion Model</h2>
            <br>
          </div>
          <div class="option">
            <ul>
              <li> <i class="fa fa-picture-o" aria-hidden="true"></i> Stable Diffusion
model is based on diffusion techniques for generating images from textual
descriptions.</li>
              <li> <i class="fa fa-crop" aria-hidden="true"></i> It uses deep neural
networks, often based on architectures like Transformers or autoregressive models, to
generate images.</li>
              <li> <i class="fa fa-retweet" aria-hidden="true"></i> Trained on image
</li>
              <li> <i class="fa fa-sign-out" aria-hidden="true"></i> Output of Stable
Diffusion models is typically images generated based on the input noise or textual
prompts. </li>
              <li> <i class="fa fa-tasks" aria-hidden="true"></i> Performs various
image generation tasks, such as creating artwork, generating images from textual
descriptions, or enhancing low-resolution images.</li>
            </ul>
            <a href="http://127.0.0.1:5000">Stable Diffusion Model</a>
          </div>
        </div>
        <div class="card text-center">
          <div class="title">
            <h2>Hugging Face Model</h2><br>
          </div>

```

```

<div class="price">
    <ul>
        <li> <i class="fa fa-picture-o" aria-hidden="true"></i> Hugging Face
provides a wide range of natural language processing (NLP) models and tools,
including tokenizers and pre-trained models.</li>
        <li> <i class="fa fa-crop" aria-hidden="true"></i> Hugging Face Token
p-rimarily deals with NLP models, such as Transformer-based architectures like GPT
(Generative Pre-trained Transformer) models.</li>
        <li> <i class="fa fa-retweet" aria-hidden="true"></i> Models are trained
on large-scale text corpora and primarily used for tasks like text
generation,summarization,translation,etc.</li>
        <li> <i class="fa fa-tasks" aria-hidden="true"></i> Used for NLP tasks,
including text generation, translation, sentiment analysis, and more. </li>
    </ul>
</div>
<a href="/GenAI/demo.html">Hugging face-token Model </a>
</div> </div>

```

```

</div> </div>

```

```

<!-----PROMPTS-->

```

```

<div id="services">

```

```

    <div class="container"><br>

```

```

        <h1 class="sub-title">Sample Prompts & Images</h1>

```

```

        <div class="services-list">

```

```

            <div>

```

```

                <br>

```

```

                <p><b>Prompt:</b>Sci-fi cosmic diarama of a quasar and jellyfish in a resin
cube, volumetric lighting, high resolution, hdr, sharpen, Photorealism</p>

```

```

            </div>

```

```

            <div>

```

```

                <br>

```

```

                <p><b>Prompt:</b> beautiful castle beside a waterfall in the woods.</p>

```

```

            </div>

```

```

            <div>

```

```

                <br>

```

```

<p><b>Prompt:</b>A city with animatic visual effect in pink and blue with
a car in pink color </p>
<br>
<p><b>Prompt:</b> city with beautiful sunset with sugar maple trees and
with people</p>
</div> </div> </div> </div>
</body>
</html>

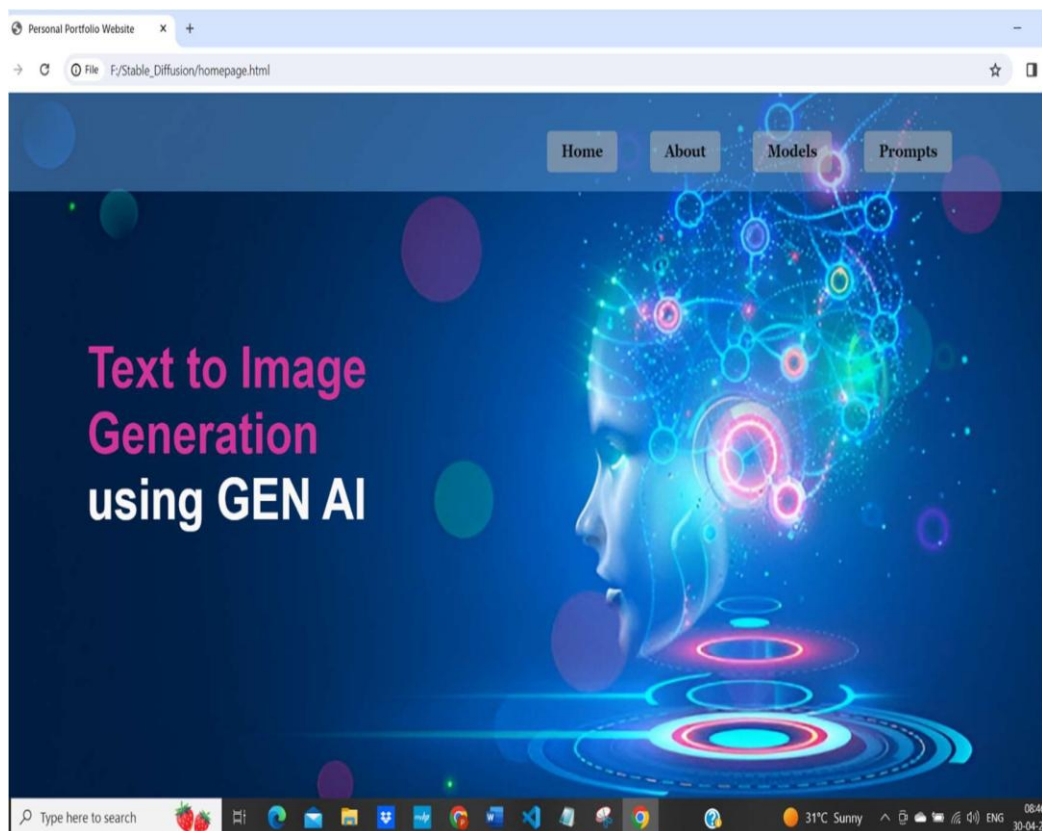
```

5.4 RESULTS

5.4.1 OUTPUT SCREENS

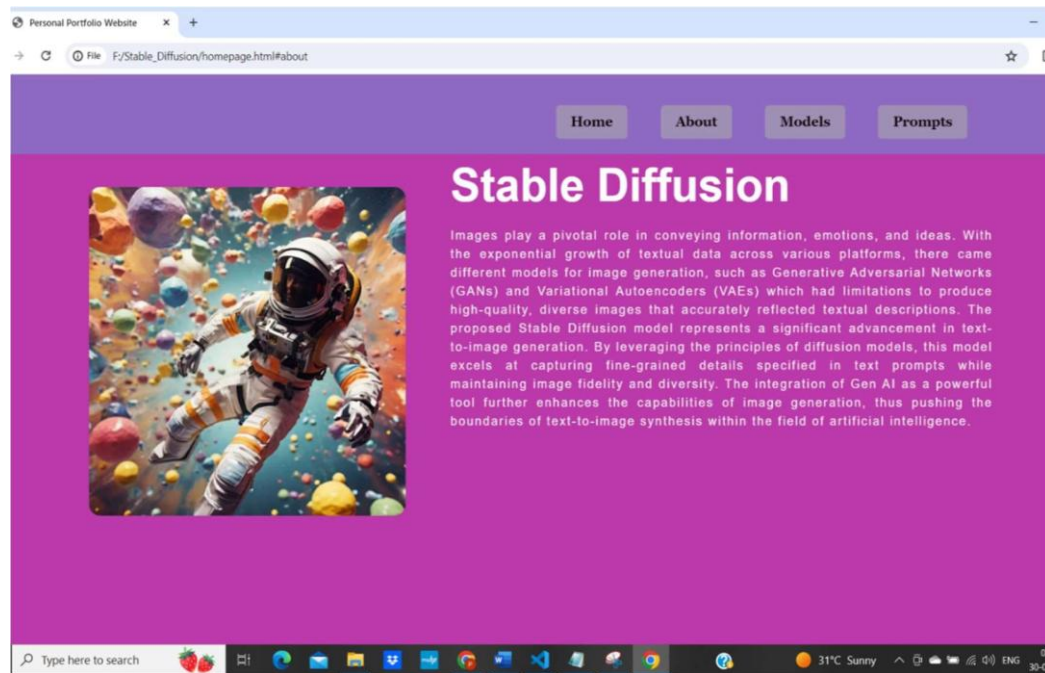
Here are the following output screens of our project.

Home page: Screen 2 shows the home page of the project which contains the navigation buttons for other pages of the project.



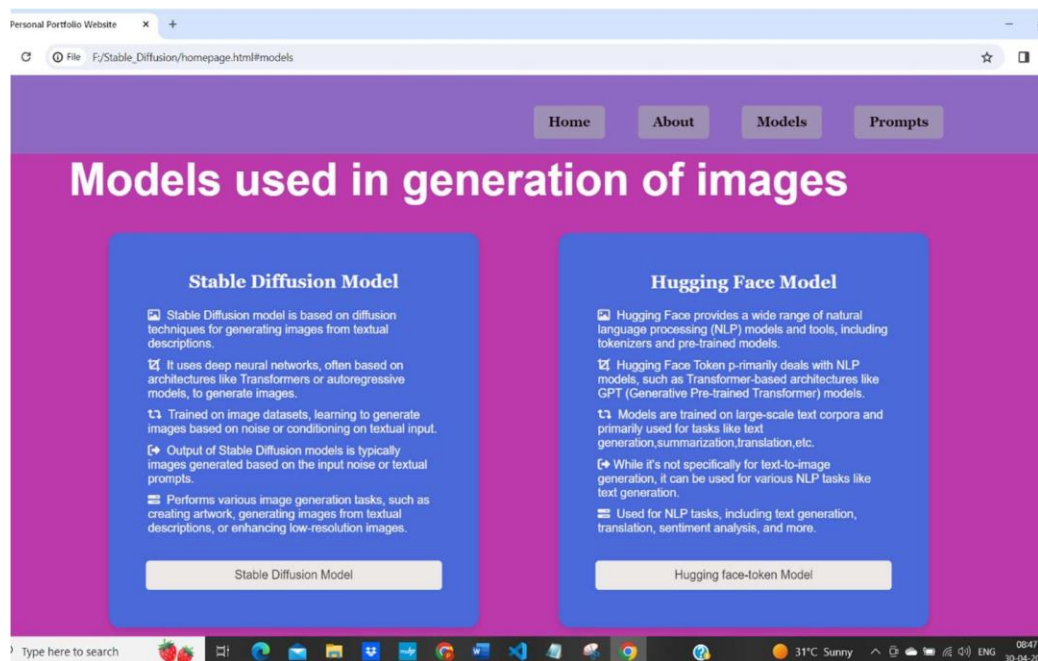
Screen 2. Home Page

About page: Screen 3 gives the brief description of the developed Stable Diffusion model.



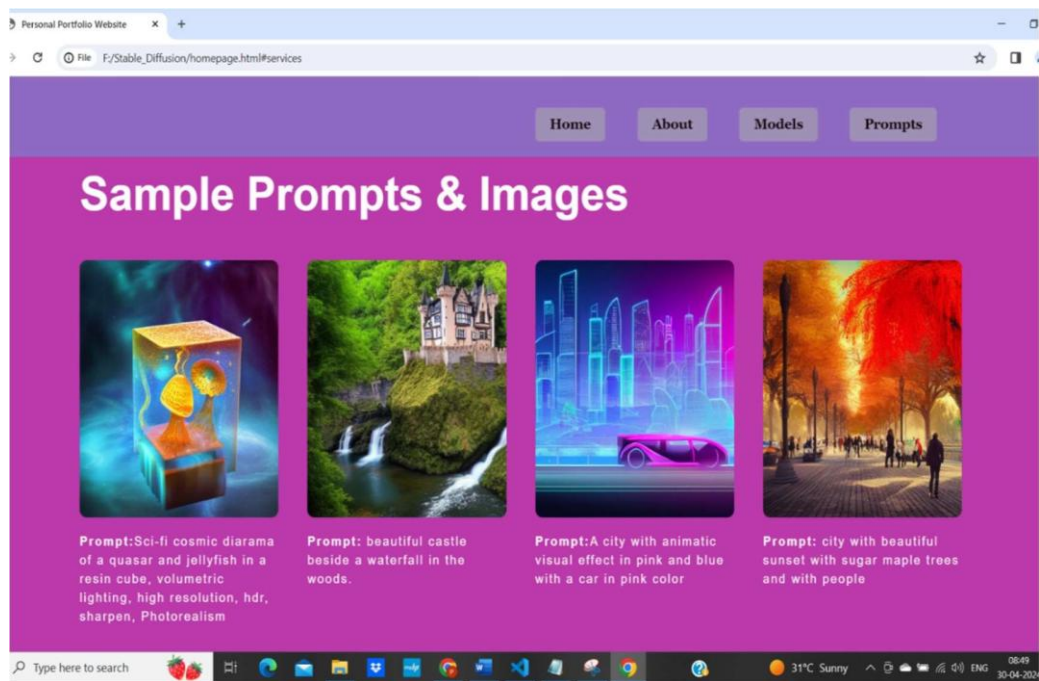
Screen 3. About Model

Models page: Screen 4 gives the information about the two models that are used in text-to-image generation



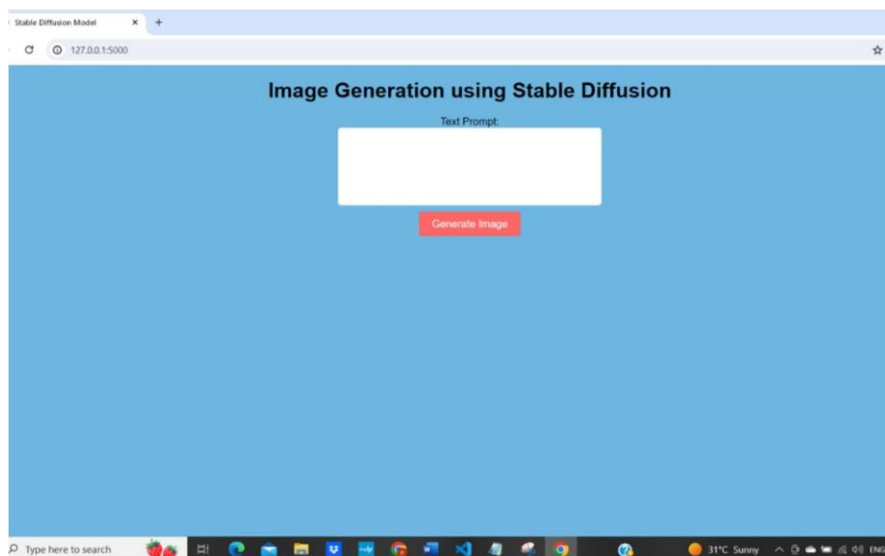
Screen 4. Comparison between Models

Sample Prompts & Images page: Screen 5 gives the information on sample prompts and images that are generated by our model.



Screen 5. Sample Prompts and the images generated by the model

Screen 6 displays the page to enter the prompt and display the image that is generated by the Stable diffusion model.

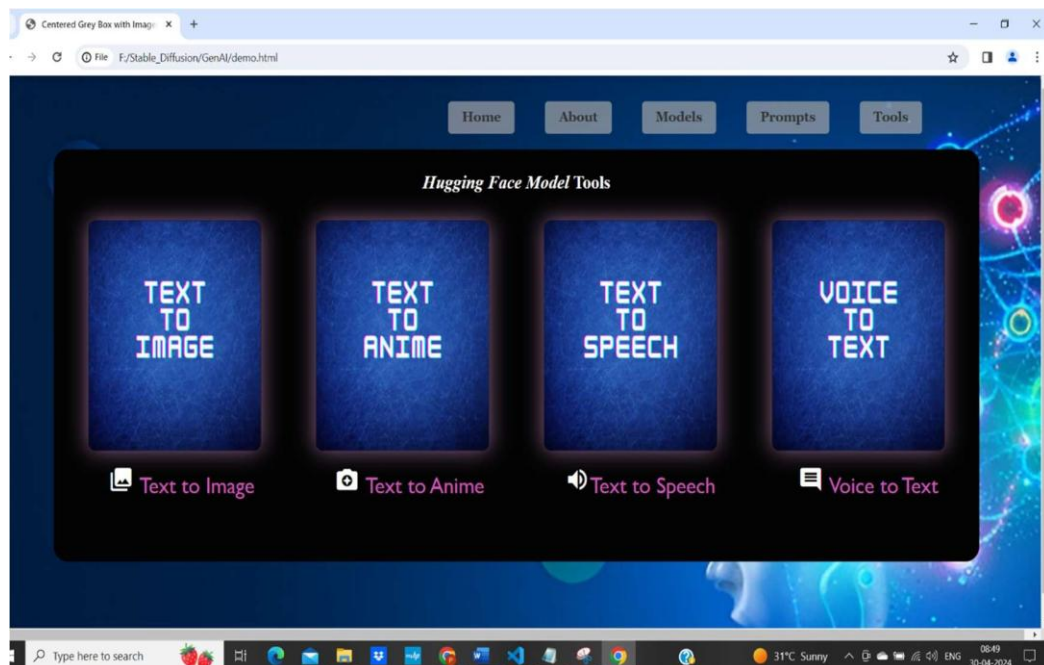


Screen 6. Page to enter the prompt to generate image using Stable Diffusion model



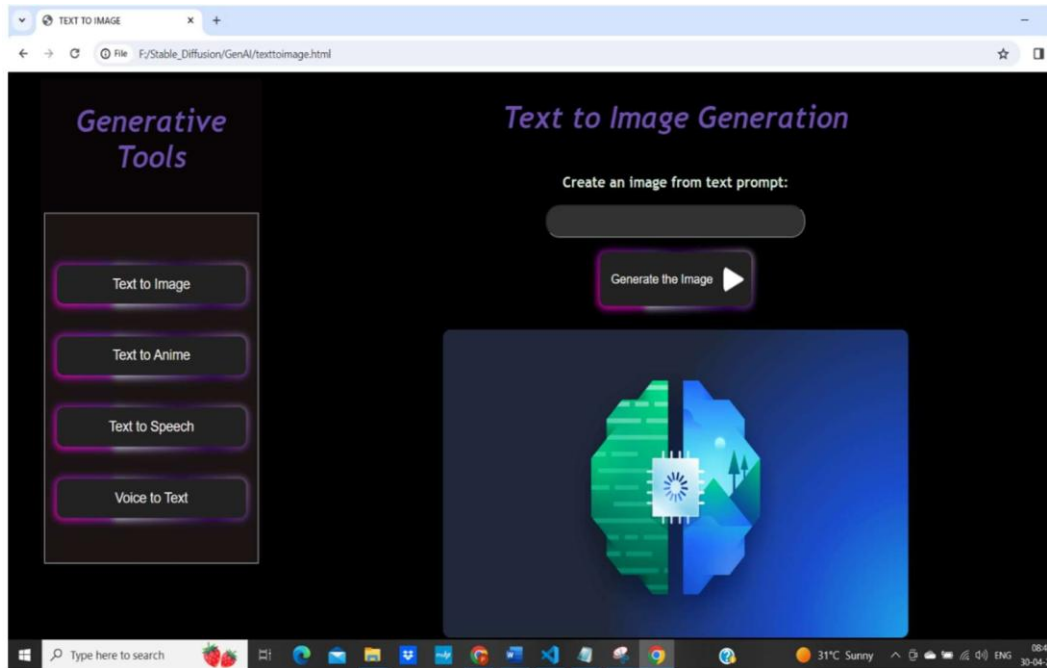
Screen 7. Image generated by the Stable Diffusion(Prompt: Cars in a full traffic)

Hugging Face Model page: Screen 8 displays the home page for the hugging face model tools to image,text and audio generation.

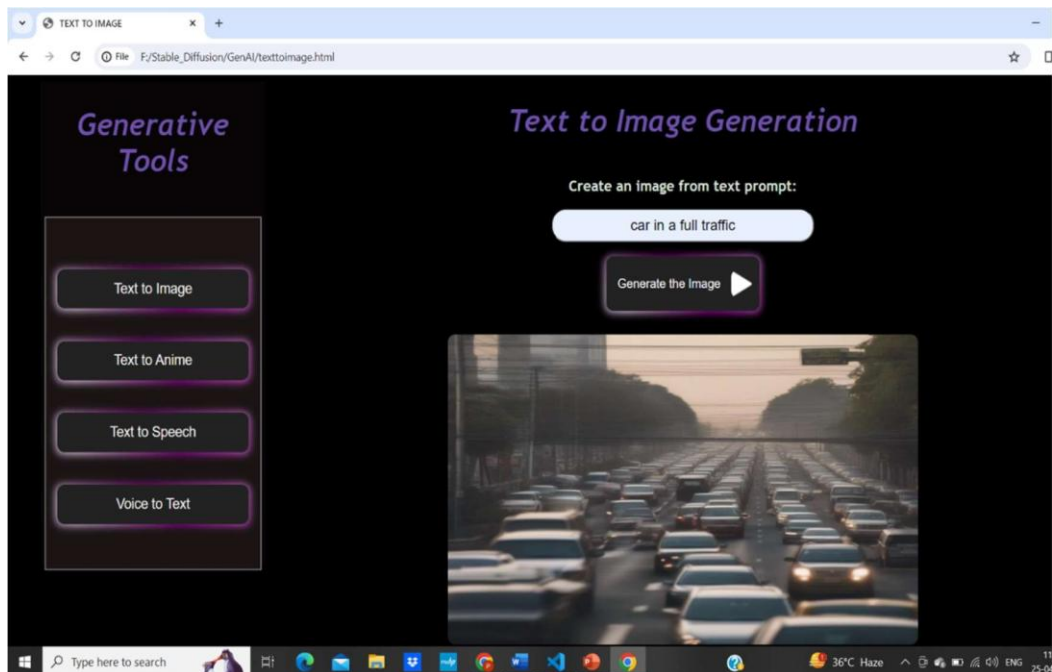


Screen 8. Hugging face model to generate images, text and audio.

Screen 9 and 10 displays the page that generates the image from the text using token generated by Hugging model

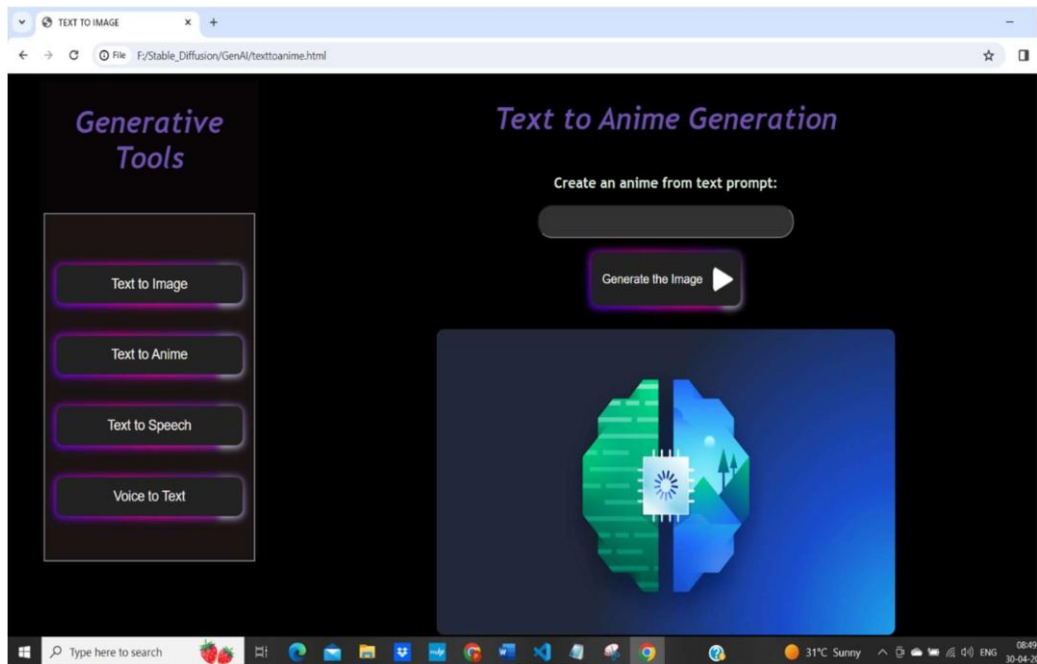


Screen 9. Text to image generation page implemented using Hugging face model

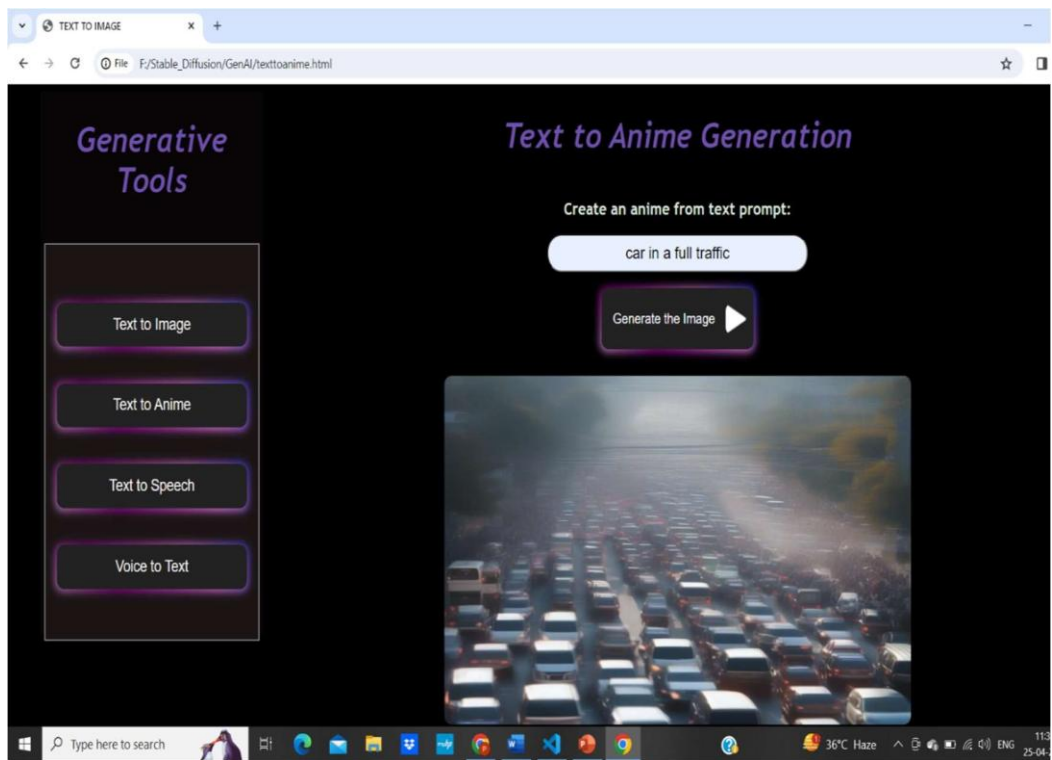


Screen 10. Image generated for the prompt car in a full traffic

Screen 11 and 12 displays the page that generates the anime-image from the text using token generated by Hugging model.

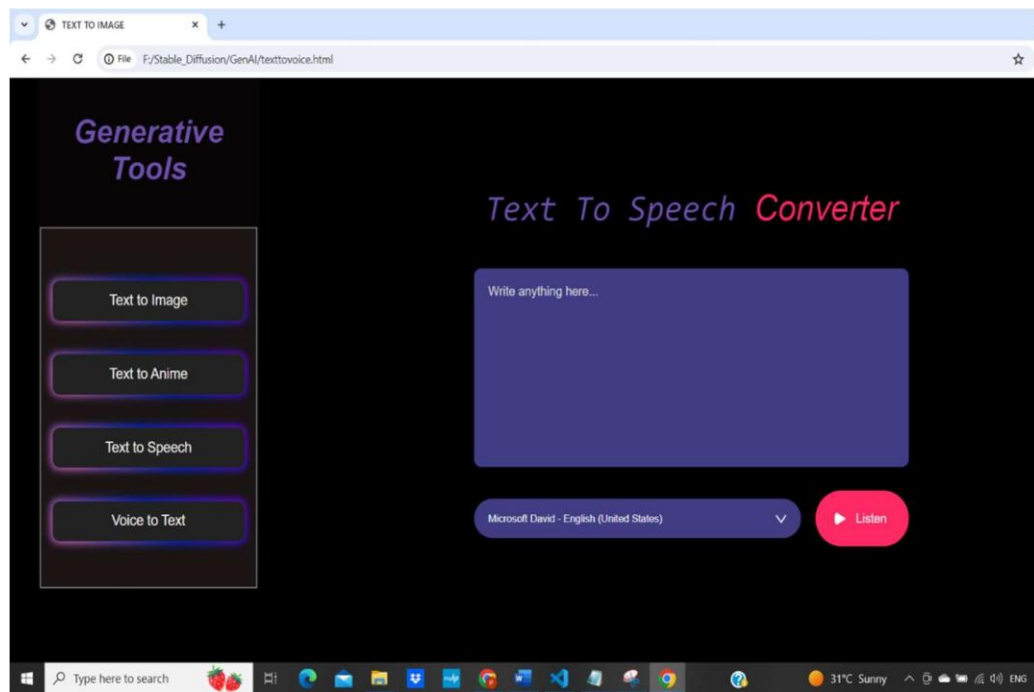


Screen 11. Text to Anime generation implemented using Hugging face model



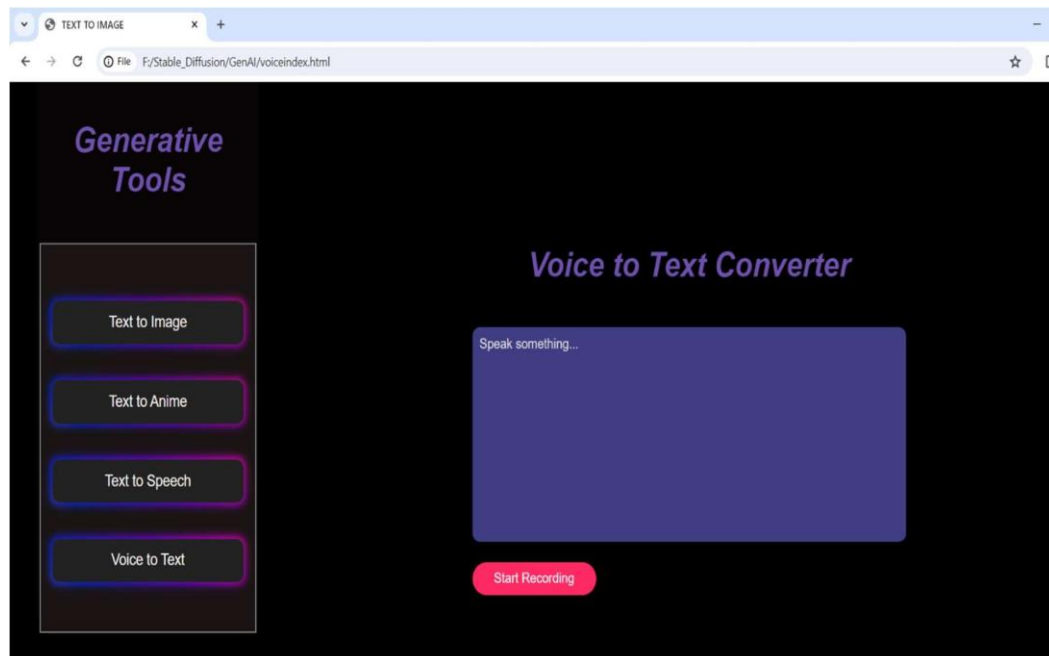
Screen 12. Anime-Image generated for the prompt car in a full traffic

Screen 13 displays the page that converts the given text prompt to audio speech.



Screen13. Text to Speech Converter Screen

Screen 14 displays the page that converts the given audio speech to text and displays the text.



Screen 14. Voice to Text Converter screen

5.5 CONCLUSION

In conclusion, the implementation of our text-to-image generation project using stable diffusion marks a significant milestone in bridging the gap between textual descriptions and visual representations. Through meticulous planning, research, and development, we have successfully translated theoretical concepts into a functional system capable of generating high-quality and diverse images from textual input. Utilizing advanced techniques such as stable diffusion models and U-Net architecture, we have overcome challenges in capturing fine-grained details and maintaining image fidelity while synthesizing images from textual descriptions. The integration of Flask as the frontend framework provides a user-friendly interface for interacting with the system, allowing users to input text descriptions and receive corresponding images as output.

6. TESTING AND VALIDATION

6.1 INTRODUCTION

Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.

Testing is important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effective and customer satisfaction.

6.1.1 Scope

A primary purpose for testing is to detect software failures so that defects may be uncovered and corrected. This is a non-trivial pursuit. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

6.1.2 Defects and Failures

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirements gaps, e.g., unrecognized requirements that result in errors of omission by the program designer. A common source of requirements gaps is non-functional requirements such as testability, scalability, maintainability, usability, performance, and security. Software faults occur through the following processes. A programmer makes an error which results a defect in the software source code. If this defect is executed, in this case the system will produce wrong results.

For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new hardware platform, alterations in source data or interacting with different software. A single defect may result in a wide range of failure symptoms.

6.1.3 Compatibility

A frequent cause of software failure is compatibility with another application, a new operating system, or, increasingly, web browser version. In the case of lack of backward compatibility, this can occur because the programmers have only considered coding their programs for, or testing the software upon, "the latest version of" this-or-that operating system. The unintended consequence of this fact is that: their latest work might not be fully compatible with earlier mixtures of software/hardware, or it might not be fully compatible with another important operating system. In any case, these differences, whatever they might be, may have resulted in software failures, as witnessed by some significant population of computer users.

6.1.4 Input Combinations and Preconditions

A very fundamental problem with software testing is that testing under all combinations of inputs and preconditions is not feasible, even with a simple product. This means that the number of defects in a software product can be very large and defects that occur infrequently are difficult to find in testing. More significantly, non functional dimensions of quality (how it is supposed to be versus what it is supposed to do) can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another. Precondition is defined as Environmental and state conditions that must be fulfilled before the component or system can be executed with a particular test or test procedure. In this project requires textual descriptions as input, ranging from simple sentences to more complex paragraphs, each describing a specific scene, object, or concept. These textual descriptions serve as the guiding input for the generation process, providing the semantic context from which the model generates corresponding images. Additionally, auxiliary information such as conditioning labels or attributes may be incorporated into the input to further guide the generation process and control specific visual attributes of the output images.

6.1.5 Static Vs Dynamic Testing

The main objective of static testing is to improve the quality of software applications by finding errors in early stages of the software development process. Whereas in dynamic testing, code is executed. It checks for functional behaviour of the software system, memory/CPU usage and overall performance of the system.

6.1.6 Software Verification and Validation

Verification: It is the process of checking that software achieves its goals without any bugs. It is the process to ensure whether the product is right or not.

Validation: It is the process of checking whether the software product is up to the mark or else product has high level requirements. It validates the right product or not.

6.2 DESIGN OF TEST CASES AND SCENARIO

The test cases are designed for our model as below in Table 9. The test cases are formatted so that the user can enter values in the proper format.

Table 6. Testcases and it's validations

Test Case	Input Prompt	Result
Testing from the user	Input the valid prompt with meaning	Generates Exact image
Testing from the user	Prompt with grammatical mistakes	Generates image with some changes
Testing from the user	Prompt without meaning but with nouns included	Generates image with objects
Testing from the user	Prompt without exact meaning but random words(nouns)	Generates image with less accurate things
Testing from the user	Prompt contains random letters without meaning	Generates a plain or blank images
Testing from the user	Blank Prompt	Generates Blank image

6.2.1 Validation Testing

It is testing where testers performed functional and non-functional testing. Functional testing includes Unit Testing, Integration Testing and System Testing and non- functional testing includes User acceptance testing.

Testing Objectives:

- 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.
- A successful test uncovers an as-yet-undiscovered error.

Test Levels: The test strategy describes the test level to be performed. There are primarily three levels of testing.

- Unit Testing
- Integration Testing
- System Testing

In most software development organizations, the developers are responsible for unit testing. Individual testers or test teams are responsible for integration and testing.

6.2.2 Unit Testing

Unit testing is done on individual modules as they are completed and become executable. It is confined only to the designer's requirements. Each module can be tested using the following two strategies.

6.2.3 Black Box Testing

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. In this testing only the output is checked for correctness. The logical flow of the data is not checked. This testing has been uses to find errors in the following categories:

- Incorrect or missing functions
- Interface errors

- Errors in data structure or external database access
- Performance errors

6.2.4 White Box Testing

White Box Testing is software testing technique in which internal structure, design and coding of software is tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing. In this test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases.

6.2.5 Integration Testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

6.2.6 System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of systemtesting is the configuration-oriented system integration test. System testing is based onprocess descriptions and flows, emphasizing pre-driven process links and integration points.

6.3 CONCLUSION

This chapter emphasizes the validation of the system using different testing approaches like black box testing and unit testing and obtaining the highest accuracy.

7. CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The fusion of stable diffusion and U-Net methods in text-to-image generation marks a groundbreaking advancement at the intersection of natural language processing and image synthesis. By leveraging stable diffusion for controlled refinement and U-Net for detailed reconstruction, we've achieved a paradigm shift in generating highly realistic and diverse images from textual descriptions. This innovative approach not only overcomes previous limitations but also sets a new standard for capturing the richness and complexity of textual descriptions in visually compelling representations. As we continue to explore and refine this transformative approach, the possibilities for creating immersive visual content from textual input are virtually limitless, ushering in a new era of creativity and expression in artificial intelligence-driven image synthesis.

7.2 FUTURE SCOPE

The future prospects of stable diffusion-based text-to-image generation are promising and diverse. Enhancements in fidelity and diversity of generated images through advanced diffusion models and training techniques are anticipated. The approach's applicability could expand beyond text-to-image synthesis to include multimodal inputs like audio or video, benefiting fields such as virtual reality and multimedia storytelling. Addressing scalability and efficiency challenges is crucial for real-world deployment, requiring optimization of computational resources and parallelized algorithms. Additionally, focusing on interpretability and controllability aspects could empower users to manipulate visual attributes and guide generation towards desired outcomes. Ultimately, the future scope involves advancing image synthesis, generating realistic and controllable visual content from text, and extending applications across various domains.

REFERENCES

- [1] Sadia Ramzan, Muhammad Munwar Iqbal and Tehmina Kalsum, “Text-to-Image Generation Using Deep Learning”, Published in MDPI, 29 July 2022, DOI:10.3390/ACCESS.2022020016
- [2] Unnati kolte, Samprada Kale, Priti Yamkar, Sakshi Deshpande, “Text-Image Generation Stable Diffusion“, Published in IJSREM (Volume:7), 5 May 2023, DOI:10.55041/IJSREM21136
- [3] Agneet Chatterjee, Gabriela Ben Melech Stan, Estelle Aflalo, Sayak Paul”, “Improving Spatial Consistency in Text-Image Models”, Published in arXiv, 1 April 2024, DOI:10.48550/ACCESS.2404.01197
- [4] Seongmin Lee, Benjamin Hoover, Hendrik Strobelt, Zijie J. Wang, ShengYun, "Diffusion Explainer: Visual Explanation for Text-to-image Stable Diffusion", Published in arXiv, 8 May 2023, DOI:10.48550/ ACCESS.2305.03509
- [5] Yogesh Balaji, Seungjun Nah, Xun Huang, “eDiff-I: Text-to Image Diffusion Models with an Ensemble of Expert Denoisers”, Published in arXiv, 14 March 2023, DOI:10.48550/ACCESS.2211.01324
- [6] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala, “Adding Conditional Control to Text-to-Image Diffusion Models”, Published in arXiv, 26 Nov 2023, DOI:10.48550/ACCESS.2302.05543
- [7] Xinyuan Chen, Yaohui Wang, Yue Lu, Yu Qiao, “Brush Your Text: Synthesize Any Scene Text on Images via Diffusion Model”, Published in arXiv, 19 Dec 2023, DOI:10.48550/ACCESS.2312.12232
- [8] Syed Sha Alam, Jeyamurun, Mohamed Faiz Ali, Veerasundari, “Stable Diffusion Text-Image Generation”, Published in IJSREM (Volume:7), 2 February 2023, DOI:10.55041/ACCESS.IJSREM17744
- [9] Mr. R. Nanda Kumar, Manoj Kumar M, Hari Hara Sudhan V, Santhosh R, “Text To Image Generation Using AI”, Published in IJCRT (Vol:11), 5 May 2023.

- [10] Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, “Text-to-Image Diffusion Model on Mobile Devices within Two Seconds”, Published in arXiv, DOI:10.48550/ACCESS.2306.00980
- [11] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, Published in arXiv, 23 May 2022, DOI:10.48550/ ACCESS.2205.11487
- [12] Chunyang Bi, Xin Luo, Sheng Shen, Mengxi Zhang, Huanjing Yue and Jingyu Yang, “Towards Real-World Image Super-Resolution via Degradation-Aware Stable Diffusion”, Published in arXiv, DOI:10.48550/ACCESS.2404.00661
- [13] Divyanshu Mataghare, Shailendra S. Aote, Ramchand Hablani, “Text To Image Generation Using Stable Diffusion“, Published in ECB, DOI:10.31838/ ACCESS.2023.12.s3.496.
- [14] Yutong He, Alexander Robey, Naoki Murata, Yiding Jiang, Joshua Salakhutdinov, and J. Zico Kolter, "Automated Black-box Prompt Engineering for Personalized Text-to-Image Generation", Published in arXiv, 28 Mar 2024, DOI:10.48550/ACCESS.2403.19103
- [15] Scott Reed, Zeynep Akata, Xincheng Yan, Lajanugen Logeswaran, “Generative Adversarial Text to Image Synthesis”, Published in arXiv, 5 June 2016, DOI:10.48550/ACCESS.1605.05396
- [16] <https://aws.amazon.com/what-is/stable-diffusion/#>
- [17] <https://blog.marvik.ai/2023/11/28/an-introduction-to-diffusion-models-and- stable-diffusion/>
- [18] <https://www.sciencedirect.com/science/article/pii/S2405844023039646>
- [19] <https://docs.openvino.ai/2024/notebooks/236-stable-diffusion-v2-text-to- image-with-output.html#>
- [20] <https://poloclub.github.io/diffusion-explainer/#>

