

Website: www.nbkrist.org.

Ph: 08624-228 247

Email: ist@nbkrist.org.

Fax: 08624-228 257

N.B.K.R. INSTITUTE OF SCIENCE & TECHNOLOGY

(Autonomous)

(Approved by AICTE: Accredited by NBA: Affiliated to JNTUA, Ananthapuramu)

An ISO 9001-2000 Certified Institution

Vidyanagar -524 413, Tirupati District, Andhra Pradesh, India

BONAFIDE CERTIFICATE

This is to certify that the project work entitled “**TEXT TO IMAGE GENERATION USING STABLE DIFFUSION**” is a bonafide work done by P.YUVARAJ, V.SAI LEKHYA, S.V.SURYA NARAYANA, SK. ROSHNI in the department of **INFORMATION TECHNOLOGY AND ARTIFICIAL INTELLIGENCE & DATA SCIENCE, N.B.K.R Institute of Science & Technology, Vidyanagar** and is submitted to department of IT AND AI&DS. This work has been carried out under my supervision.

Mrs. M. SUJANA

Assistant Professor

Department of IT and AI&DS

N.B.K.R.I.S.T

Dr. A. NARAYANA RAO

Professor & HOD

Department of IT and AI&DS

N.B.K.R.I.S.T

Submitted for the Viva-Voce Examination held on _____

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

We wish to express my gratitude to those who extended their valuable co-operation and contribution towards the project.

We would like to express my gratitude to Founder of institution
Mr. N. Venkata Rami Reddy.

We would like to express my gratitude to Correspondent of institution
Mr. N. Ram Kumar Reddy.

A special thanks to Director of the Institution **Dr. V. Vijaya Kumar Reddy** for permitting and encouraging for doing this project

We are proudly grateful to express my deep sense of gratitude and respect towards **Dr. A. Narayana Rao**, Professor and Head of the Department, IT and AI&DS, whose motivation and constant encouragement has led to pursue a project in field of Computer Vision .We are very fortunate, for having his enlighten us in all the aspects of life and transforming us to be an ideal individual in this field.

Our special thanks to **Mrs. M. Sujana** Assistant Professor, Department of IT and AI&DS, for his content guidance, motivation and also for providing an excellent environment. We are truly grateful for his valuable suggestions and advice. We are much obliged and thankful for providing this opportunity and constant encouragement given by his during the course. We are grateful to his valuable guidance and suggestions during our project work.

We express our thanks to other professors, classmates, friends and non- teaching staff who have helped us for the completion of our project and without infinite love and patience this would never have been possible.

We would like to thank one and all who have helped us directly and indirectly to complete this project successfully

Finally, we express our thanks to other professors, classmates, friends and non-teaching staff who have helped us for the completion of our project and without infinite love & patience this would never have been possible.

Table of Contents

Chapter No	Topic Name	Page No
	Certificate	
	Acknowledgement	
	List of Figures	VI
	List of abbreviations	VII
	Abstract	VIII
1	Introduction	
	1.1 Introduction	9
	1.2 Background and Motivation	9
	1.3 Problem Statement	10
	1.4 Objectives and Scope	11
	1.5 Organization of the project Report	12
	1.6 Summary	12
2	Literature Review	
	2.1 Introduction	13
	2.2 Literature Survey	13
	2.3 Identification of Research Gap	14
	2.4 Summary	15
3	Methodology	
	3.1 Introduction	16
	3.2 Overview of Methodological Approach	16
	3.3 Parameters	20
	3.4 Description of Tools and Technologies Used	24
	3.5 Summary	24
4	Software Implementation	
	4.1 Introduction	25
	4.2 Code Structure and Organization procedure	25
	4.3 Algorithms and Techniques Implemented	28
	4.4 Summary	37

5	Results and Analysis	
5.1	Introduction	38
5.2	Results	38
5.3	Analysis of Performance Metrics	42
5.4	Comparison with Existing Systems	43
5.5	Discussion of Findings	44
5.6	Summary	45
6	Conclusion and Future Work	
6.1	Conclusions Drawn from the Study	46
6.2	Limitations and Challenges Encountered	46
6.3	Suggestions for Future work	47
7	References	
	List of References Cited in the Report	49
8	Appendices	
	Code Snippets (if applicable)	50

LIST OF FIGURES

S.no	Figure No.	Figure Name	Page No
1	3.1	system architecture	19
2	4.1	Data Collection	26
3	4.2	Dataset preparation using data dreamer	27
4	4.3	Loading the data which is generated by data dreamer	28
5	4.4	After exporting the data download the model	29
6	4.5	Model training	30
7	4.6	Home page	34
8	4.7	Given text as input	35
9	4.8	Result image	36
10	4.9	Deployment	36
11	5.1	Terminal	38
12	5.2	User interface Home page	39
14	5.3	Image generating page	39
15	5.4	Prompt generated by the model	40
16	5.5	Image generated based on input	40
17	5.6	Loss function	41

LIST OF ABBREVIATIONS

NAME	EXPANSION
PEFT	Parameter Efficient Fine-Tuning
LORA	Low-Rank Adaptation
VAE	Variational Auto Encoder
CLIP	Contrastive Language-Image Pre-training
CFG Scale	Classifier free Guidance Scale

ABSTRACT

Text-to-image generation is a rapidly advancing field that aims to generate realistic images from textual descriptions. One of the state-of-the-art models for this task is Stable Diffusion, a deep learning model that uses diffusion models and large language models. In this, we present an in-depth exploration of Stable Diffusion's text-to-image generation capabilities. We discuss its underlying architecture, evaluate its performance on benchmark datasets, and investigate techniques to fine-tune it for specific domains.

Our findings demonstrate Stable Diffusion's significant potential in bridging the gap between language and visual representation, with wide-ranging applications in creative writing, product design, and data visualization. We also explore the model's limitations and discuss avenues for future improvements, contributing to the ongoing efforts to develop robust and versatile AI systems for text to image generation.

1. INTRODUCTION

1.1 Introduction

Nowadays we always hear about new technology that improves our lifestyle, that makes our life easier. Technology has revolutionized the human mankind. Human race has put a gear in technology and they are not in a mood to move the pedals away from this gear. There is huge research on various technology sector such as Artificial Intelligence, Smart phones and many more. This research led to new inventions and making one's life easier. The field of text-to-image generation has undergone remarkable advancements in recent years, propelled by the increasing demand for AI-driven content creation tools. Text-to-image generation systems, leveraging deep learning techniques, have demonstrated the capability to produce realistic and contextually relevant images from textual descriptions. This chapter serves as an introduction to the research project, providing an overview of the significance of text-to-image generation and its applications across various domains, including design, advertising, e-commerce, and virtual environments.

1.2 Background and Motivation

Background:

The background of text-to-image generation using stable diffusion models involves a convergence of advancements in deep learning, specifically diffusion models, and natural language processing. Stable diffusion models represent a novel approach to generative modeling, leveraging principles from diffusion processes to model image generation. In the context of text-to-image generation, stable diffusion models offer several advantages, including improved stability during training and the ability to generate high-quality images with controllable attributes. By incorporating large language models such as CLIP (Contrastive Language-Image Pretraining), stable diffusion models enable seamless integration of textual prompts into the image generation process, facilitating more precise control over the generated images' characteristics. Overall, the background of text-to-image generation using stable diffusion models signifies a significant advancement in AI-driven content generation, offering promising avenues for creating diverse, realistic images from textual descriptions.

Motivation:

Text-to-image generation holds immense promise in revolutionizing visual content creation by automating the process of generating images from text descriptions. This technology offers benefits such as enhancing communication and creativity by enabling visual storytelling, streamlining workflows through automation, and providing personalized and customizable visual content. Moreover, text-to-image generation tools augment human creativity by serving as creative aids for artists and designers, fostering innovation in research, and promoting accessibility and inclusivity by democratizing image creation. The motivation for text-to-image generation lies in its potential to transform various industries, including marketing, e-commerce, entertainment, and education, by making visual content creation more efficient, engaging, and accessible to a broader audience.

1.3 Problem statement:

Text-to-image generation has emerged as a promising field within artificial intelligence, offering the potential to revolutionize content creation and communication. However, despite significant advancements in recent years, several challenges persist that hinder the widespread adoption and effectiveness of text-to-image generation systems.

One of the primary challenges in text-to-image generation is the limited ability of models to comprehend the semantic context and nuances embedded within textual descriptions. Existing approaches often struggle to capture the intricate relationships between words and their corresponding visual representations, resulting in generated images that may lack coherence or fail to accurately convey the intended concepts. Addressing this challenge requires the development of advanced neural architectures and training strategies that can better understand the semantics of textual input and translate it into visually meaningful imagery.

Another significant issue facing text-to-image generation systems is the limited diversity and creativity exhibited by generated outputs. Many existing models tend to produce images that are repetitive or lack novelty, leading to a lack of variety in the generated visual content. This limitation hampers the utility of text-to-image generation systems in applications requiring diverse and original imagery, such as art generation, storytelling, and creative content production. Overcoming this challenge necessitates the exploration of novel generative techniques, including the integration of attention mechanisms, variational inference, and adversarial training, to promote the generation of more diverse and creative visual outputs. Scalability and efficiency pose additional challenges for text-to-image generation systems, particularly concerning the computational resources and time required for training and inference. Many state-of-the-art

models are computationally intensive and resource-demanding, making them impractical for real-time applications or large-scale deployment. Moreover, the training process for text-to-image generation models often involves massive datasets and complex optimization procedures, leading to prolonged training times and high resource costs. Addressing these scalability and efficiency challenges requires the development of lightweight and efficient models, as well as optimization techniques that can accelerate training and inference without compromising on the quality of generated images.

1.4 Objectives and Scope

The objectives of the research project are outlined to provide a clear direction for the investigation. These objectives include the development of a robust and scalable text-to-image generation system capable of producing high-quality images from diverse textual prompts. The scope of the project encompasses the exploration of state-of-the-art deep learning architectures, dataset curation, model training, and evaluation methodologies. Additionally, the project aims to investigate the impact of text representation techniques and domain-specific fine-tuning strategies on the performance of text-to-image generation models.

One primary objective is to develop and implement advanced neural architectures capable of better understanding the semantic context embedded within textual prompts. By leveraging techniques such as attention mechanisms, semantic embeddings, and multimodal fusion, the goal is to enable the model to capture complex relationships between words and visual concepts, resulting in more coherent and contextually relevant image generation.

Another objective is to promote diversity and creativity in generated images by exploring novel generative techniques and training strategies. This includes investigating methods such as conditional generation, style transfer, and latent space manipulation to encourage the production of varied and imaginative visual outputs. By enhancing the model's ability to generate novel and visually appealing imagery, the project aims to broaden the applicability of text-to-image generation in diverse domains such as art, design, and entertainment.

A critical objective is to improve the scalability and efficiency of text-to-image generation systems, making them more practical for real-world applications and large-scale deployment. This involves optimizing model architectures, streamlining training procedures, and exploring techniques for model compression and acceleration. By reducing computational overhead and resource requirements, the project aims to enable text-to-image generation models to operate efficiently in resource-constrained environments while maintaining high-quality image generation performance.

1.5 Organization of the project Report

The organization of the project report delineates the structure and layout of the document, providing readers with a roadmap for navigating through the content. It outlines the chapters and sections included in the report, along with a brief overview of the content covered in each section. By presenting a structured framework for presenting information, the organization of the project report enhances readability and comprehension, facilitating efficient access to relevant topics and insights. This section serves as a guide for readers, enabling them to navigate through the report seamlessly and locate specific information of interest.

1.6 Summary

In summary, This project aims to revolutionize text-to-image generation by integrating latent diffusion models and cross-attention conditioning mechanisms. By addressing training inefficiencies and enhancing model flexibility, it strives to achieve state-of-the-art results in conditional image synthesis tasks. The report provides a comprehensive overview of methodologies, system design, implementation details, and analysis. It serves as a valuable resource for researchers and practitioners in AI and image generation, documenting the project's challenges, conclusions, and future directions. Overall, the project endeavors to advance the frontiers of text-to-image synthesis and foster inclusivity in creative expression.

2. LITERATURE SURVEY

2.1 Introduction

The literature survey conducted in this study offers a comprehensive overview of the field of text-to-image generation, beginning with seminal works such as DALL·E by Ramesh et al. (2021), which introduced a model capable of generating diverse images from textual descriptions. Recent advancements, such as Stable Diffusion Models for Text-to-Image Generation by Chen et al. (2023), are also explored, highlighting efforts to address specific challenges like limb generation issues and enhance overall image quality. Through a detailed examination of existing research, key contributions, methodologies, and outcomes are synthesized, enabling the identification of research gaps and opportunities for future exploration. This structured survey lays the groundwork for advancing the state-of-the-art in text-to-image generation and provides valuable insights for researchers aiming to drive innovation in this evolving field.

2.2 Literature Survey

1. alignDRAW: Aligning Text and Geometry for Image Generation by Xu et al (2015):

This paper proposes alignDRAW, a model that aligns text descriptions with geometric structures to generate realistic images from text descriptions.

2. Generative Adversarial Text-to-Image Synthesis (DCGAN) by Zhang et al (2017):

This work introduces DCGAN, a framework for text-to-image synthesis using generative adversarial networks, which has shown promising results.

3. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks by Zhang et al (2018):

StackGAN utilizes a two-stage architecture to generate high-resolution images from text descriptions, achieving notable improvements in realism.

4. DALL·E: Creating Images from Text Using Adversarial Training by S. Ramesh et al (2021):

DALL·E is a model capable of generating diverse images from textual descriptions, demonstrating the ability to understand and generate complex scenes.

5. DALL·E 3: Further Explorations in Text-to-Image Generation by S. Ramesh et al (2023):

DALL·E 3 extends the capabilities of the original DALL·E model, exploring new directions in text-to-image generation and achieving improved performance.

6. **Stable Diffusion Models for Text-to-Image Generation: Initial Exploration by Gao et al (2022):** Stable diffusion models are explored for text-to-image generation, demonstrating improved stability and convergence during the generation process.
7. **Stable Diffusion Models for Text-to-Image Generation: Addressing Limb Generation Issues by Gao et al (2023):** This update focuses on fixing issues related to limb generation in stable diffusion models for text-to-image generation, enhancing overall image quality.

2.3 Identification of research gap

Identifying the research gap is critical for understanding where the current state of knowledge falls short and where opportunities for further investigation lie. In the context of sign language recognition and communication technology, several research gaps emerge from the existing literature and technological landscape.

The evolution of text-to-image generation models, as depicted in the provided papers, showcases a trajectory from early attempts to recent advancements addressing complex challenges. Initial models like alignDRAW (Xu et al., 2015) laid the groundwork by aligning textual descriptions with geometric structures to generate realistic images, setting the stage for more sophisticated approaches. Subsequent works such as DCGAN (Zhang et al., 2017) introduced generative adversarial networks into the realm of text-to-image synthesis, demonstrating promising results and highlighting the potential of adversarial training in this context. The progression continued with StackGAN (Zhang et al., 2018), which introduced a two-stage architecture to achieve photo-realistic image synthesis from text, marking notable improvements in realism and resolution.

The emergence of DALL·E (S. Ramesh et al., 2021) represented a significant milestone in text-to-image generation, showcasing the ability to generate diverse and complex scenes from textual descriptions through adversarial training. Subsequent iterations, such as DALL·E 3 (S. Ramesh et al., 2023), further extended the capabilities of the original model, exploring new directions and achieving improved performance in generating images from text. Alongside these advancements, stable diffusion models entered the scene, as demonstrated by Gao et al. (2022), offering improved stability and convergence during the generation process. This initial exploration paved the way for further refinements, such as addressing limb generation issues in stable diffusion models (Gao et al., 2023), thereby enhancing overall image quality and addressing specific challenges encountered in text-to-image generation.

Collectively, these papers underscore the iterative nature of research in text-to-image generation, marked by a progression from foundational models to increasingly sophisticated techniques that address specific limitations and challenges. Moving forward, the field is poised to continue its trajectory of innovation, leveraging insights from past works to develop more robust and versatile models capable of generating realistic and diverse images from textual descriptions.

2.4 Summary

In summary, the literature survey delves into the evolution of text-to-image generation techniques, spanning from early rule-based approaches to state-of-the-art deep learning models. Seminal works like DALL·E and StackGAN have paved the way for advancements in generating diverse and realistic images from textual descriptions. Recent research, such as stable diffusion models, addresses challenges like stability and limb generation issues, enhancing the overall quality of generated images. While existing models demonstrate promising results, there remains a gap in achieving fine-grained control and semantic understanding in image synthesis. Future research directions may focus on improving model interpretability, scalability, and controllability to further unlock the potential of text-to-image generation across diverse applications.

3.METHODOLOGY

3.1 Introduction:

The methodology section constitutes the backbone of any research endeavor, delineating the systematic approach employed to address the research questions or objectives effectively. It serves as a roadmap guiding the execution of the study from inception to conclusion, ensuring clarity, transparency, and replicability. This section is structured to provide a comprehensive overview of the methods, tools, and techniques utilized in the research process, thereby enabling readers to understand the underlying procedures and rationale behind each step.

Text-to-image generation is a rapidly advancing field that aims to generate realistic images from textual descriptions. One of the state-of-the-art models for this task is Stable Diffusion, a deep learning model that uses diffusion models and large language models. In this, we present an in-depth exploration of Stable Diffusion's text-to-image generation capabilities. We discuss its underlying architecture, evaluate its performance on benchmark datasets, and investigate techniques to fine-tune it for specific domains. Our findings demonstrate Stable Diffusion's significant potential in bridging the gap between language and visual representation, with wide-ranging applications in creative writing, product design, and data visualization. We also explore the model's limitations and discuss avenues for future improvements, contributing to the ongoing efforts to develop robust and versatile AI systems for text-to-image generation.

3.2 Overview of Methodological Approach

The methodological approach to text-to-image generation involves several key steps to effectively translate textual descriptions into visual representations.

- 1. Data Collection:** Data collection for model training is facilitated through Data Dreamer, a potent Python library. It prompts the user to provide initial object names, which are then relayed to Mistral 7b Instruct v0.1, a language model tasked with generating relevant prompts. Subsequently, these prompts are forwarded to SDXL Lightning, an image generation model, to produce the corresponding images. This iterative process allows for the accumulation of the requisite data corpus essential for training the desired model. Through this seamless integration of components, Data Dreamer streamlines the data collection phase, enabling efficient model development.

- 2. Model Accessing :** To leverage the capabilities of the pretrained Stable Diffusion model, we employ the diffusion pipeline, a powerful tool that facilitates seamless integration and utilization of the model. This pipeline abstracts away the complexities of model loading, preprocessing, and inference, allowing developers to focus on their application logic. With a few lines of code, the diffusion pipeline grants access to the pretrained Stable Diffusion model, enabling the generation of high-quality images from textual prompts. By encapsulating the intricate details of the diffusion process, the pipeline ensures a consistent and user-friendly experience, empowering developers to harness the full potential of this cutting-edge technology.
- 3. Model Training:** To fine-tune the Stable Diffusion model for specific tasks, researchers employ LoRA (Low-Rank Adaptation), a technique derived from PEFT (Parameter-Efficient Fine-Tuning). LoRA enables efficient adaptation of large language models by introducing trainable rank decompositions, reducing the number of trainable parameters. This method proves particularly valuable when working with substantial models like Stable Diffusion, as it circumvents the need for full fine-tuning, thereby saving computational resources and time. By leveraging LoRA, practitioners can tailor the Stable Diffusion model to their unique requirements while preserving its core capabilities, resulting in a streamlined and cost-effective fine-tuning process.
- 4. Model Testing:** After the training process is complete, the next step is to push the fine-tuned model to the Hugging Face platform, a popular repository for sharing and accessing pretrained models. Once uploaded, the model can be seamlessly loaded into the desired environment, allowing for comprehensive testing and evaluation using textual prompts as inputs. However, it's important to note that during the testing phase, errors or unexpected outputs may arise, as the training process is not always perfect, and the model's performance can be influenced by various factors, such as the quality and quantity of the training data, the choice of hyperparameters, and the complexity of the task itself. In such cases, further analysis and potential retraining might be necessary to refine the model's capabilities and ensure optimal performance.
- 5. Model Evaluation:** To comprehensively assess the performance and capabilities of a trained model, a dual approach involving both qualitative and quantitative analysis is employed. Qualitative evaluation involves a critical examination of the model's outputs, such as generated images or text, by human experts or domain specialists. This subjective analysis considers factors like coherence, relevance, and overall quality. Conversely, quantitative analysis relies on objective metrics and numerical scores, measuring aspects like accuracy, precision, recall, or perplexity, depending on the task. By combining these two complementary methods, researchers can gain a holistic understanding of the model's strengths, weaknesses, and areas for improvement, enabling informed decisions regarding further refinement or deployment in real-world applications.

6. Deployment and Application: Upon successful evaluation and refinement, the next step is to deploy the model as a web service, enabling seamless integration and accessibility for end-users or applications. This process involves leveraging Flask, a lightweight and versatile Python web framework, to create an intuitive API interface. By encapsulating the model's functionality within a Flask application, developers can expose its capabilities through well-defined endpoints, facilitating communication and data exchange. The API interface acts as a bridge, allowing clients to send requests containing input data and receive the corresponding model outputs, such as generated text, images, or predictions. This deployment approach not only streamlines the model's integration into various systems but also enhances its scalability and ease of maintenance, ensuring a smooth user experience.

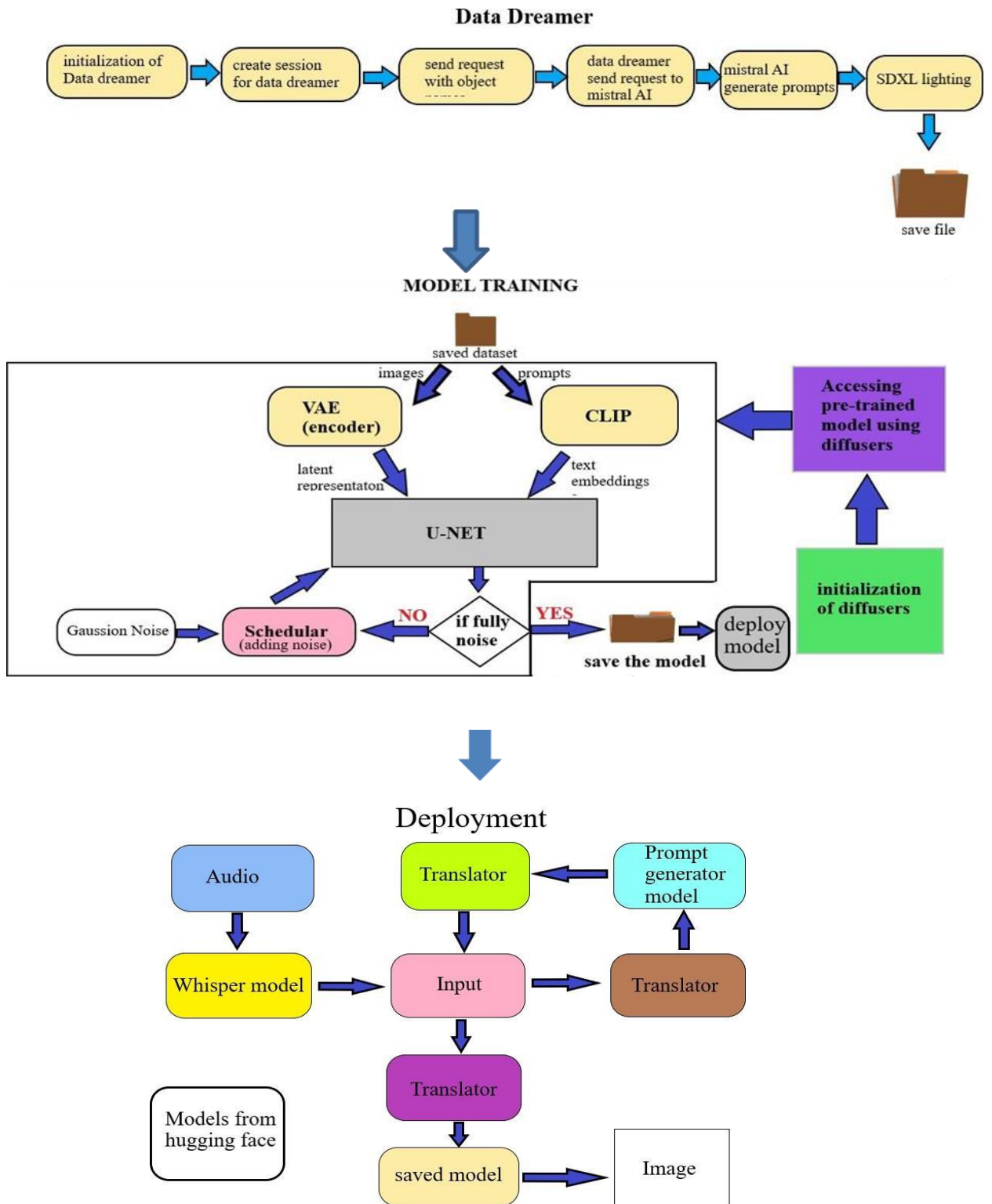


Fig 3.1: system architecture

3.3 Parameters

In the development of the text to image generation model, several parameters are crucial for optimizing the performance and enhancing the accuracy of the models. The parameters used in Description of Tools and Technologies Used:

Data Collection:

1. Datadreamer: The "data dreamer" library is a Python library that provides a simple and intuitive interface for working with data in Python. It includes features such as data visualization, data manipulation, and data analysis. The library is designed to be easy to use and can be used by both beginners and experienced data scientists. It is a great tool for anyone who wants to work with data in Python.

2. Mistral 7B Instruct v0.1: The Mistral 7B is a large language model developed by Mistral AI. It is based on the Mistral 7A model and has been trained on a vast amount of text data. The model is capable of generating human-like text and can be used for a variety of tasks such as text classification, language translation, and text generation. The v0.1 version of the model is the latest version available and is constantly being updated with new data and features.

3. SDXL Lightning: SDXL (Software Defined Extensible Lightning) is a new technology developed by Salesforce that allows for the customization and extension of the Salesforce Lightning platform. It provides a flexible and scalable way to create custom user interfaces, workflows, and reports without the need for custom code. SDXL is designed to be easy to use and can be implemented quickly, making it a great option for organizations looking to enhance their Salesforce experience.

4. Text Prompts: Text prompts are brief, written statements or phrases that are used to generate responses in various contexts, such as customer service, chatbots, and language translation. They provide a starting point for the AI system to understand the user's intent and generate a response based on predefined parameters or machine learning algorithms. Text prompts can be customized to suit specific needs and can help improve the effectiveness and efficiency of AI systems in various applications.

5. Images: Images refer to visual representations of information, typically in the form of digital files. They can be in various formats such as JPEG, PNG, and GIF. Images are used in various contexts, including websites, social media, presentations, and advertisements.

Model Access:

1. **Diffusers:** Hugging Face's Diffusers library is a toolkit for accessing pretrained models. Diffusers provide a set of utilities and classes for applying various transformations, such as denoising, upscaling, and colorization, to images. Downloading and using pretrained models is easy with Hugging Face, as they provide a wide range of pretrained models for a variety of tasks.

Model Training:

1. **VAE (Variational Autoencoder):** Variational autoencoders are popular neural network architectures used in text-to-image generation. They enable the learning of latent representations that capture the underlying structure of textual and visual data, facilitating the generation of realistic images from text.
2. **U-Net:** U-Net is a convolutional neural network architecture commonly employed in text-to-image generation tasks. Its unique design, featuring symmetric contracting and expanding paths, enables effective feature extraction and spatial resolution preservation, making it well-suited for image synthesis tasks.
3. **CLIP Text Encoder:** The CLIP (Contrastive Language-Image Pretraining) text encoder is an integral component of many text-to-image generation models. It encodes textual prompts into vector representations that capture semantic meanings, facilitating the alignment of text and image modalities during generation.
4. **Scheduler:** The scheduler component in text-to-image generation models manages the training process, optimizing hyperparameters, and orchestrating model updates to ensure efficient convergence and performance improvement during training.
5. **Gaussian Noise:** Gaussian noise is often introduced as a regularization technique during the training of text-to-image generation models. It helps prevent overfitting by adding random variations to the input data, promoting generalization and robustness in the trained models.
6. **PEFT (Predictive Encoder Fine-Tuning):** PEFT is a fine-tuning strategy employed in text-to-image generation models. It involves iteratively updating the model parameters based on predictions made during inference, refining the model's performance over time.

7. **LoRA (Latent Optimization and Refinement for Attention):** LoRA is a refinement technique used to enhance the quality of generated images in text-to-image generation tasks. It leverages latent optimization and attention mechanisms to iteratively refine the generated outputs, improving their fidelity and realism.

Model Evaluation:

1. **Qualitative Analysis:** Qualitative analysis involves visually inspecting the generated images to assess their quality, realism, and alignment with the given textual prompts. Human evaluators provide subjective judgments based on visual inspection, identifying strengths, weaknesses, and areas for improvement in the generated outputs.
2. **Quantitative Analysis:** Quantitative analysis entails using objective metrics to assess the performance of text-to-image generation models. Metrics such as inception score, FID (Fréchet Inception Distance), and precision-recall curves provide quantitative measures of image quality, diversity, and fidelity compared to ground truth data.
3. **CLIP Score:** The CLIP score quantifies the similarity between generated images and their corresponding textual prompts, as assessed by the CLIP text encoder. Higher CLIP scores indicate better alignment between text and image modalities, reflecting the model's ability to faithfully translate textual descriptions into visually coherent images.
4. **CLIP Bidirectional Similarity:** CLIP bidirectional similarity measures the bidirectional consistency between text and image embeddings produced by the CLIP model. It evaluates the degree of alignment between textual prompts and generated images in both directions, providing insights into the mutual understanding between text and image modalities.

Model Deployment

1. **Hugging Faces:** Hugging Faces offers a comprehensive suite of tools and libraries for deploying text-to-image generation models in production environments. Its pre-trained models, inference APIs, and deployment frameworks streamline the deployment process, enabling seamless integration of models into various applications and platforms.
2. **Interface API (Serverless):** Interface API provides a serverless deployment solution for text-to-image generation models, eliminating the need for managing infrastructure and scaling resources manually. It offers a user-friendly interface for deploying models as RESTful APIs, enabling easy access and integration into diverse applications.
3. **Prompt Generator:** Prompt Generator is a component of the deployment pipeline responsible for generating textual prompts dynamically based on user inputs or system requirements. It enables interactive control over the image generation process, allowing users to specify desired image attributes, styles, or content themes through textual prompts.
4. **Whisper Large v3:** Whisper Large v3 is a state-of-the-art text-to-image generation model optimized for deployment in resource-constrained environments. Its lightweight architecture and efficient inference algorithms enable fast and scalable image generation without compromising on quality or fidelity.
5. **LLAMA 3:** LLAMA 3 is a scalable deployment platform designed specifically for text-to-image generation models. It leverages distributed computing resources and containerized deployment strategies to support large-scale inference tasks, ensuring high throughput and low latency in real-time applications.
6. **Flask:** Flask is a lightweight web framework commonly used for deploying text-to-image generation models as web services. Its simplicity and flexibility make it well-suited for building RESTful APIs and web applications that interact with text-to-image generation models.

7. **Google trans:** The Google trans Python library provides convenient access to Google Translate's machine translation capabilities, facilitating multilingual support in text- to-image generation models. It enables seamless translation of textual prompts between different languages, enhancing the accessibility and usability of deployed models for global audiences.

3.4 Description of Tools and Technologies Used

3.4.1 Hardware Requirements:

- CPU: Intel Core i5 with multiple cores and high clock speeds.
 - GPU: NVIDIA Tesla V100 with a minimum of 8 GB VRAM (preferably more).
 - RAM: 8 GB or more DDR4 RAM.
 - Storage: SSD with a minimum of 500 GB
 - Mouse and Keyboard: Any standard mouse and keyboard should suffice.
- Software Requirements:

3.4.2 Software Requirements:

- Operating System: Windows 10
- Coding Language: Python 3.7

3.5 Summary

The methodology employed in text-to-image generation encompasses various steps aimed at designing and implementing an effective computational framework for translating textual descriptions into corresponding visual representations. Initially, the research design is meticulously selected, taking into account the nature of the research questions and objectives. Whether a qualitative, quantitative, or mixed-methods approach is chosen depends on factors such as the complexity of the text-to-image generation task and the availability of resources. Following this, a thorough overview of the methodological approach outlines the sequential steps involved in the text-to-image generation process. This includes data collection and preprocessing, model architecture design, parameter selection and optimization, training, and evaluation procedures. Each step is meticulously planned and executed to ensure the robustness, efficiency, and generalizability of the text-to-image generation system.

4. SOFTWARE IMPLEMENTATION

4.1 Introduction

In this project, we present a comprehensive approach to leveraging the power of Stable Diffusion, a state-of-the-art text-to-image generation model, to create a robust and versatile visual content generation system. Our implementation involves fine-tuning the Stable Diffusion XL model using a dataset of images generated by the Data Dreamer tool, making the model accessible through the Hugging Face API, and deploying it on a Flask-based web application. The core of our strategy lies in the fine-tuning process, where we utilize the powerful Stable Diffusion XL model as a foundation and further train it on a custom dataset of Data Dreamer generated images. This fine-tuning step, facilitated by the PEFT (Parameter-Efficient Fine Tuning) method with LORA (Low-Rank Adaptation), allows us to unlock the model's potential and tailor its performance to specific use cases, resulting in the generation of more personalized and context-relevant visuals.

Throughout this project, we have meticulously documented the implementation details, the challenges encountered, and the strategies employed to overcome them. By sharing our insights and experiences, we aim to contribute to the ongoing advancements in the field of text-to-image generation and inspire others to explore the transformative potential of this cutting-edge technology.

4.2 Code Structure and Organization procedure

4.2.1 Data Collection

In our quest to fine-tune the Stable Diffusion XL model, we have leveraged the power of the Data Dreamer Python library to generate a diverse and high-quality dataset of synthetic images. This approach allows us to efficiently create a tailored training corpus that aligns with our specific needs and use cases, rather than relying solely on generic or publicly available datasets.

The key to our data collection process lies in the seamless integration of the Data Dreamer tool. By providing the system with a set of desired objects or concepts, Data Dreamer sends a request to the Mistral-7B-Instruct-v0.2 model, which is capable of generating rich and diverse text prompts. These prompts are then fed into the Stable Diffusion XL Lightning (SDXL) model, which in turn generates high-quality images with remarkable speed and accuracy.

```

2024-04-02 15:00:54.100753: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN
2024-04-02 15:00:54.100797: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT
2024-04-02 15:00:54.107424: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS
2024-04-02 15:00:56.029973: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
Loading INT4 language model on GPU...
Loading checkpoint shards: 100% 2/2 [01:08<00:00, 34.28s/it]
Done!
Generating prompts...: 100% 50/50 [01:58<00:00, 2.38s/it]
Loading SDXL Lightning on GPU...
/usr/local/lib/python3.10/dist-packages/diffusers/configuration_utils.py:244: FutureWarning: It is deprecated to pass a pretrained model name or path to 'from_config'. It will be removed in version 0.30.0.
  deprecate("config-passed-as-path", "1.0.0", deprecation_message, standard_warn=False)
Loading pipeline components...: 100% 7/7 [00:02<00:00, 2.99it/s]
Generating images: 0% 0/50 [00:00<?, ?it/s]
0% 0/4 [00:00<?, ?it/s]
25% 1/4 [00:03<00:09, 3.23s/it]
50% 2/4 [00:03<00:02, 1.45s/it]
75% 3/4 [00:03<00:00, 1.03it/s]
100% 4/4 [00:04<00:00, 1.06s/it]
Generating images: 2% 1/50 [00:10<08:11, 10.03s/it]
0% 0/4 [00:00<?, ?it/s]
25% 1/4 [00:01<00:05, 1.96s/it]
50% 2/4 [00:02<00:02, 1.04s/it]
75% 3/4 [00:02<00:00, 1.33it/s]
100% 4/4 [00:03<00:00, 1.26it/s]

```

Fig 4.1: Data Collection

The generated images form the foundation of our fine-tuning process, enabling us to further enhance the capabilities of the Stable Diffusion XL model. By training the model on this custom dataset, we can unlock its potential to generate visuals that are tailored to our specific requirements, whether it be for product design, data visualization, creative expression, or any other domain-specific application.

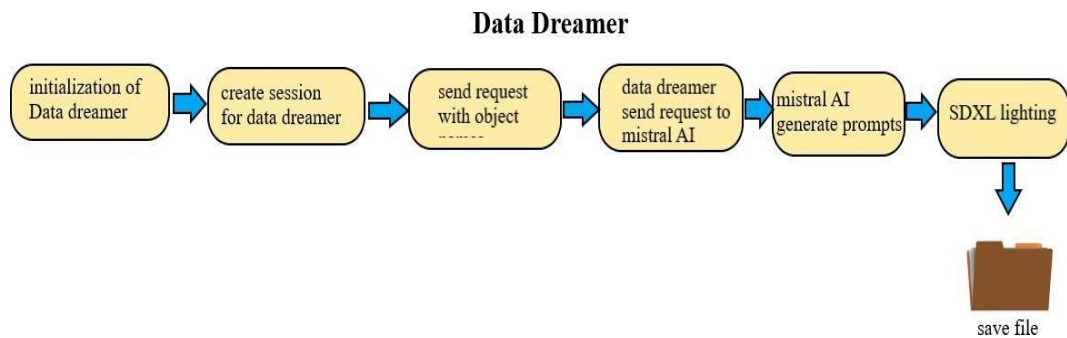


Fig 4.2: Data set preperation using data dreamer

This data collection approach offers several key advantages:

- **Customization:** By leveraging DataDreamer, we can curate a dataset that precisely aligns with our target use cases, ensuring that the fine-tuned Stable Diffusion XL model produces highly relevant and contextual images.
- **Efficiency:** The automated prompt generation and image synthesis capabilities of DataDreamer and SDXL enable us to rapidly build a comprehensive training dataset, significantly reducing the time and effort required for manual data collection and curation.
- **Scalability:** The scalable nature of this data collection process allows us to continuously expand and refine the training dataset, enabling the Stable Diffusion XL model to adapt and improve over time as our requirements evolve.

By integrating this data collection strategy into our implementation of Stable Diffusion, we have laid the groundwork for a powerful and versatile text-to-image generation system. The fine-tuned model, trained on the synthetic dataset generated by DataDreamer and SDXL, will serve as a crucial component in our broader solution, empowering users to create visuals that seamlessly bridge the gap between language and visual representation.

4.2.2 Train the Data

In our mission to develop a robust and versatile text-to-image generation system, we have meticulously crafted a fine-tuning strategy for the Stable Diffusion XL model. This approach leverages the powerful dataset generated by the Data Dreamer tool, coupled with state-of-the-art techniques like PEFT (Parameter-Efficient Fine-Tuning) and LORA (Low-Rank Adaptation), to optimize the model's performance while minimizing the resource requirements.

4.3 Algorithms and Techniques Implemented

4.3.1 Fine-Tuning with Data Dreamer-Generated Dataset

The foundation of our fine-tuning process lies in the high-quality dataset of synthetic images generated by the Data Dreamer tool. As discussed in the previous section, this dataset is carefully curated to align with our specific use cases and target applications, ensuring that the fine-tuned Stable Diffusion XL model can generate visuals that are highly relevant and context-rich.



Fig 4.3: Loading the data which is generated by data dreamer

By training the Stable Diffusion XL model on this custom dataset, we unlock its potential to produce images that seamlessly bridge the gap between language and visual representation. The fine-tuning process enables the model to learn the nuances and patterns inherent in the Data Dreamer-generated images, allowing it to generate more personalized and accurate visuals in response to user prompts.

4.3.2 Efficient Fine-Tuning with PEFT and LORA

To optimize the fine-tuning process and minimize the resource requirements, we have adopted the PEFT (Parameter-Efficient Fine-Tuning) and LORA (Low-Rank Adaptation) techniques. These approaches enable us to fine-tune the Stable Diffusion XL model in a highly efficient manner, without the need for extensive computational resources or large amounts of GPU memory.

```

Steps: 100% 500/500 [1:16:55<00:00, 9.36s/it, loss=0.173, lr=0.0001]04/11/2024 02:56:58 - INFO - accelerate.accelerator - Saving current state to SDXL_LoRA/checkpoint-5
Model weights saved in SDXL_LoRA/checkpoint-500/pytorch_lora_weights.safetensors
04/11/2024 02:56:59 - INFO - accelerate.checkpointing - Optimizer state saved in SDXL_LoRA/checkpoint-500/optimizer.bin
04/11/2024 02:56:59 - INFO - accelerate.checkpointing - Scheduler state saved in SDXL_LoRA/checkpoint-500/scheduler.bin
04/11/2024 02:56:59 - INFO - accelerate.checkpointing - Sampler state for dataloader 0 saved in SDXL_LoRA/checkpoint-500/sampler.bin
04/11/2024 02:56:59 - INFO - accelerate.checkpointing - Gradient scaler state saved in SDXL_LoRA/checkpoint-500/scaler.pt
04/11/2024 02:56:59 - INFO - accelerate.checkpointing - Random states saved in SDXL_LoRA/checkpoint-500/random_states_0.pkl
04/11/2024 02:56:59 - INFO - __main__ - Saved state to SDXL_LoRA/checkpoint-500
Steps: 100% 500/500 [1:16:55<00:00, 9.36s/it, loss=0.0733, lr=0.0001]Model weights saved in SDXL_LoRA/pytorch_lora_weights.safetensors
{'latents_mean', 'latents_std'} was not found in config. Values will be initialized to default values.

Fetching 17 files: 0% 0/17 [00:00<?, ?it/s]

diffusion_pytorch_model.safetensors: 0% 0.00/335M [00:00<?, ?B/s]

diffusion_pytorch_model.safetensors: 6% 21.0M/335M [00:00<00:01, 202MB/s]

diffusion_pytorch_model.safetensors: 16% 52.4M/335M [00:00<00:01, 225MB/s]

diffusion_pytorch_model.safetensors: 28% 94.4M/335M [00:00<00:00, 258MB/s]

diffusion_pytorch_model.safetensors: 38% 126M/335M [00:00<00:00, 260MB/s]

diffusion_pytorch_model.safetensors: 47% 157M/335M [00:00<00:00, 271MB/s]

diffusion_pytorch_model.safetensors: 56% 189M/335M [00:00<00:00, 284MB/s]

diffusion_pytorch_model.safetensors: 66% 220M/335M [00:00<00:00, 288MB/s]

diffusion_pytorch_model.safetensors: 75% 252M/335M [00:00<00:00, 287MB/s]

diffusion_pytorch_model.safetensors: 85% 283M/335M [00:01<00:00, 273MB/s]

diffusion_pytorch_model.safetensors: 100% 335M/335M [00:01<00:00, 264MB/s]

Fetching 17 files: 100% 17/17 [00:01<00:00, 10.18it/s]
{'feature_extractor', 'image_encoder'} was not found in config. Values will be initialized to default values.

```

Fig 4.4: After exporting the data download the model

PEFT is a novel fine-tuning technique that focuses on updating only a small subset of the model's parameters, rather than fine-tuning the entire network. This approach significantly reduces the memory footprint required for the fine-tuning process, allowing us to leverage the power of Stable Diffusion XL on more accessible computing platforms, such as Google Colab, which provides 13 GB of GPU RAM.

Complementing PEFT, the LORA technique further enhances the efficiency of our fine-tuning strategy. LORA introduces low-rank adaptation layers that can be added to the model, effectively reducing the number of parameters that need to be fine-tuned. This optimization not only reduces the memory requirements but also results in a much smaller final model size, with our fine-tuned Stable Diffusion XL model weighing in at only 23 MB, compared to the original 10 GB.

The combined use of PEFT and LORA in our fine-tuning process enables us to achieve remarkable results while overcoming the resource constraints often associated with large-scale text-to-image generation models. By fine-tuning the Stable Diffusion XL model on the Data Dreamer-generated dataset using these techniques, we can create a highly capable and efficient visual content generation system that can be easily deployed and scaled across a wide range of platforms and applications.

The fine-tuning of the Stable Diffusion XL model, leveraging the Data Dreamer-generated dataset and the PEFT and LORA techniques, has been a crucial step in our journey to develop a robust and versatile text-to-image generation system. By tailoring the model's capabilities to our specific use cases and requirements, we have unlocked new possibilities for creating visuals that seamlessly bridge the gap between language and visual representation.

The integration of the Mistral-7B-Instruct-v0.2 model's prompt generation capabilities has been a key enabler in our data collection process, ensuring that the training dataset captures the nuances and complexities of the desired visual outputs. This strategic integration of cutting-edge language modeling technology has laid the foundation for the development of a highly capable and adaptable text-to-image generation system.

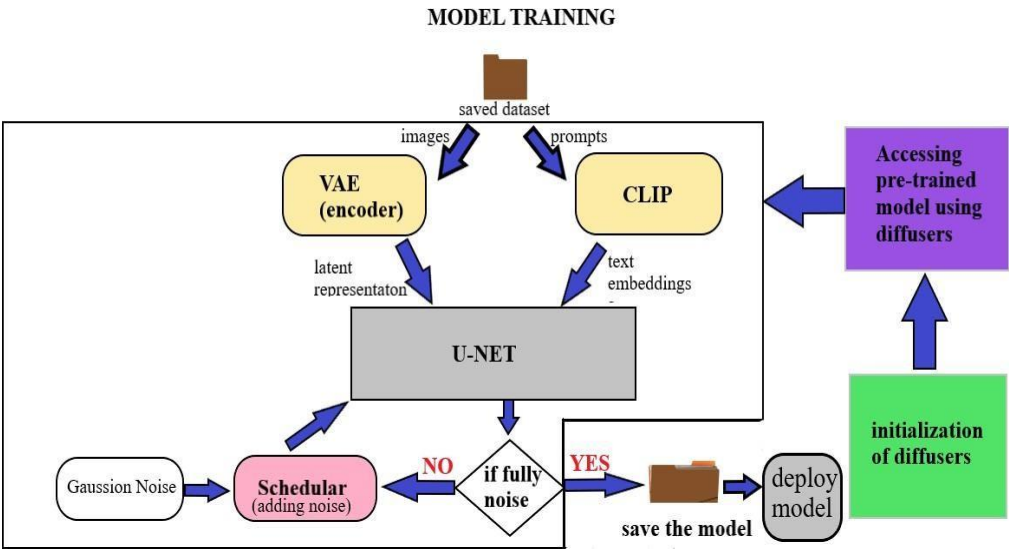


Fig 4.5: Model training

Moreover, the adoption of PEFT and LORA for efficient fine-tuning has been a game-changer, allowing us to optimize the model's performance while minimizing the resource requirements. By fine-tuning the Stable Diffusion XL model on the Data Dreamer-generated dataset using these techniques, we have created a highly capable and compact visual content generation system that can be easily deployed and scaled across a wide range of platforms and applications.

The deployment of our fine-tuned Stable Diffusion XL model on accessible cloud- based platforms, such as Google Colab, Paper space, and Kaggle, further enhances the accessibility and usability of our text-to-image generation solution. By leveraging the computing power and resources offered by these platforms, we have ensured that our system can be readily accessed and utilized by a diverse range of users, democratizing the power of AI-generated visuals.

As we continue to refine and expand our Stable Diffusion-based text-to-image generation system, we remain committed to pushing the boundaries of what is possible with this cutting-edge technology. By sharing our insights and experiences, we hope to inspire and empower others to explore the transformative potential of AI-generated visuals, unlocking new avenues for creative expression, data visualization, and collaborative problem-solving.

4.3.3. Testing Data:

To thoroughly evaluate the performance of our fine-tuned Stable Diffusion XL model, we have leveraged the Hugging Face Pipeline, which provides a seamless interface for accessing and utilizing pre-trained AI models. The Hugging Face Pipeline allows us to easily integrate our fine-tuned Stable Diffusion XL model into a text-to-image generation pipeline. We have adopted a layered approach, where we use the base Stable Diffusion 1.0 model as the foundation and then apply our fine-tuned Stable Diffusion XL model on top of it.

By utilizing this layered approach, we can benefit from the robust and well-established capabilities of the Stable Diffusion 1.0 base model, while also leveraging the enhancements and customizations we have introduced through our fine-tuning process. This combination of the general knowledge of the base model and the domain-specific adaptations of our fine-tuned version creates a powerful and versatile text-to-image generation system.




To test the performance of our fine-tuned Stable Diffusion XL model, we have curated a diverse set of text prompts that cover a range of subject matter, complexity, and creative styles. These prompts have been carefully designed to challenge the model's understanding of language, its ability to interpret visual concepts, and its capacity to generate high-quality, contextually-relevant images.



Here are some examples of the test prompts we have used:

To test the model, we have integrated the layered Stable Diffusion pipeline into our application, where we first load the base Stable Diffusion 1.0 model and then apply our fine-tuned Stable Diffusion XL model on top of it. This allows us to seamlessly generate images from the provided text prompts, leveraging the combined capabilities of both models.

By utilizing this layered approach, we can benefit from the robust and well-established capabilities of the Stable Diffusion 1.0 base model, while also leveraging the enhancements and customizations we have introduced through our fine-tuning process. This combination of the general knowledge of the base model and the domain-specific adaptations of our fine-tuned version creates a powerful and versatile text-to-image generation system.

To test the performance of our fine-tuned Stable Diffusion XL model, we have curated a diverse set of text prompts that cover a range of subject matter, complexity, and creative styles. These prompts have been carefully designed to challenge the model's understanding of language, its ability to interpret visual concepts, and its capacity to generate high-quality, contextually-relevant images.

PROMPTS	GENERATED IMAGES
<p>A photograph of a vibrant, futuristic city skyline with towering skyscrapers and flying cars.</p>	 <p>A digital artwork of a futuristic city. In the foreground, a sleek, blue, low-profile car with glowing headlights sits on a wet, reflective surface. In the background, a dense cluster of tall, modern skyscrapers rises into a bright blue sky with soft clouds. A flying car is visible in the upper right, and another one is partially visible on the left. The overall scene is vibrant and high-tech.</p>
<p>An oil painting of a majestic dragon soaring through a stormy, fantasy landscape.</p>	 <p>An oil painting of a majestic dragon. The dragon is depicted in mid-flight, soaring through a dark, stormy sky with swirling clouds. The dragon has a long, slender body, a long neck, and large, dark wings. Its scales are a mix of green and gold, and its eyes are a bright yellow. The overall mood is dramatic and fantastical.</p>
<p>A whimsical illustration of a family of colorful, anthropomorphic robots enjoying a picnic in a lush, meadow setting.</p>	 <p>A whimsical illustration of a family of colorful, anthropomorphic robots. There are four robots in total, each with a different color: pink, blue, orange, and light blue. They are sitting around a small, round, white picnic table in a lush, green meadow. The table is set with various picnic items, including a basket, a bottle, and some food. The robots are dressed in simple, cute clothing, and they all have friendly expressions. The background shows a soft, hazy landscape with trees and a gentle sky.</p>

<p>A detailed, architectural rendering of a sustainable, eco-friendly hospital building with a modern, organic design.</p>	
<p>A surreal, digital art piece depicting a dreamlike landscape with floating islands and mysterious, glowing creatures.</p>	

Here are some examples of the test prompts we have used:

To test the model, we have integrated the layered Stable Diffusion pipeline into our application, where we first load the base Stable Diffusion 1.0 model and then apply our fine-tuned Stable Diffusion XL model on top of it. This allows us to seamlessly generate images from the provided text prompts, leveraging the combined capabilities of both models.

As the images are generated, we closely analyze their performance across several key criteria:

Coherence: How well do the generated images align with the input prompts, capturing the intended meanings and visual concepts?

Realism: To what extent do the generated images exhibit realistic and natural-looking characteristics?

Creativity: How well does the model handle prompts that require imaginative or abstract visual interpretations?

Consistency: Is the model able to generate visually consistent and cohesive images across multiple prompts within a particular domain or theme?

By rigorously testing the fine-tuned Stable Diffusion XL model using this Hugging Face Pipeline-based approach, we can thoroughly assess its performance, identify areas for further improvement, and validate the effectiveness of our fine-tuning strategy.

The insights gained from this testing process will not only help us refine and enhance our text-to-image generation system but also contribute to the broader research and development efforts in the field of AI-powered visual content creation. By sharing our findings and engaging with the Hugging Face community, we aim to further the progress and adoption of these transformative technologies.

4.3.4.Setting Up User Interface

Expanding the provided code into a detailed explanation with additional content requires further elaboration on various aspects, including the Flask application setup, route definitions, functionality, interaction with external Python scripts, error handling, and security considerations. Below is the expanded content in paragraph format:

The provided code initializes a Flask application, which serves as the backbone for creating web-based interfaces and managing HTTP requests. Flask is a micro web framework for Python, providing essential tools and utilities for building web applications. The application is created with the `Flask (name)` constructor, where `name` represents the current Python module. This initializes the Flask application instance, allowing route definitions and other configurations.

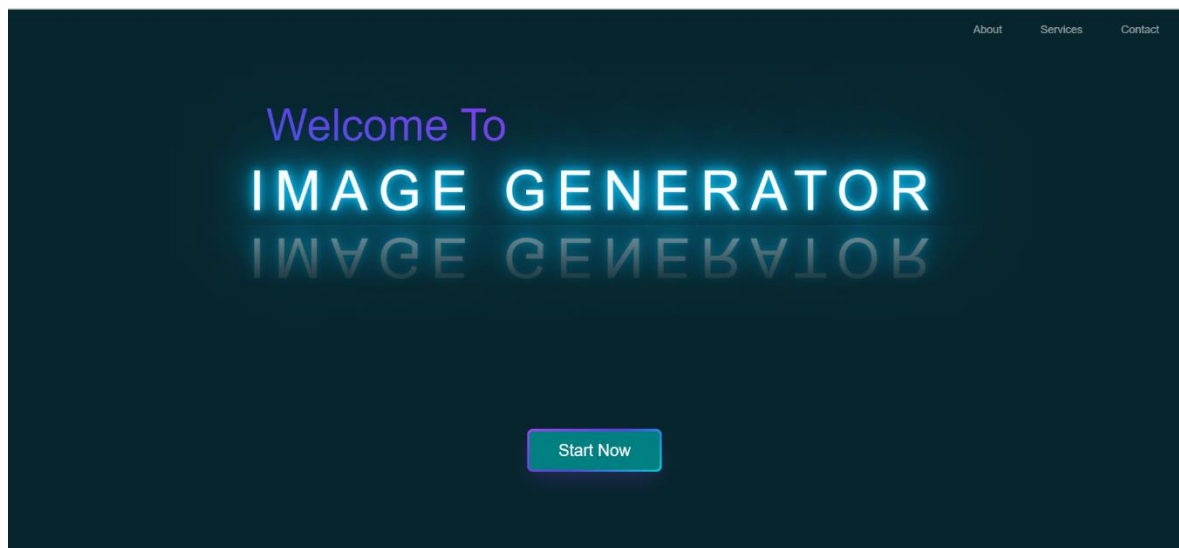


Fig 4.6: Home Page

Route definitions are crucial in Flask applications as they determine how URLs are mapped to specific functions and templates. The / route is associated with the index() function, which renders the 'index.html' template. This template typically contains the main interface or homepage of the web application, providing users with an initial view of the application's functionality and options.

The result route is linked to the generate() function, which renders the 'index.html' template. This route is commonly used to display the results of data processing or analysis, depending on the application's purpose. The 'index.html' template may include data visualizations, summaries, or any other relevant information derived from processing user inputs or external data sources.

One of the key functionalities provided by the Flask application is the ability to execute external Python scripts asynchronously. The /help route triggers the execution of an external Python script named 'app.py'. This route is often used to initiate long-running tasks or background processes without blocking the main Flask application. When accessed, it checks if a Python process is already running. If not, it starts a new process using the subprocess. This allows for running tasks asynchronously and provides scalability and responsiveness to the web application.

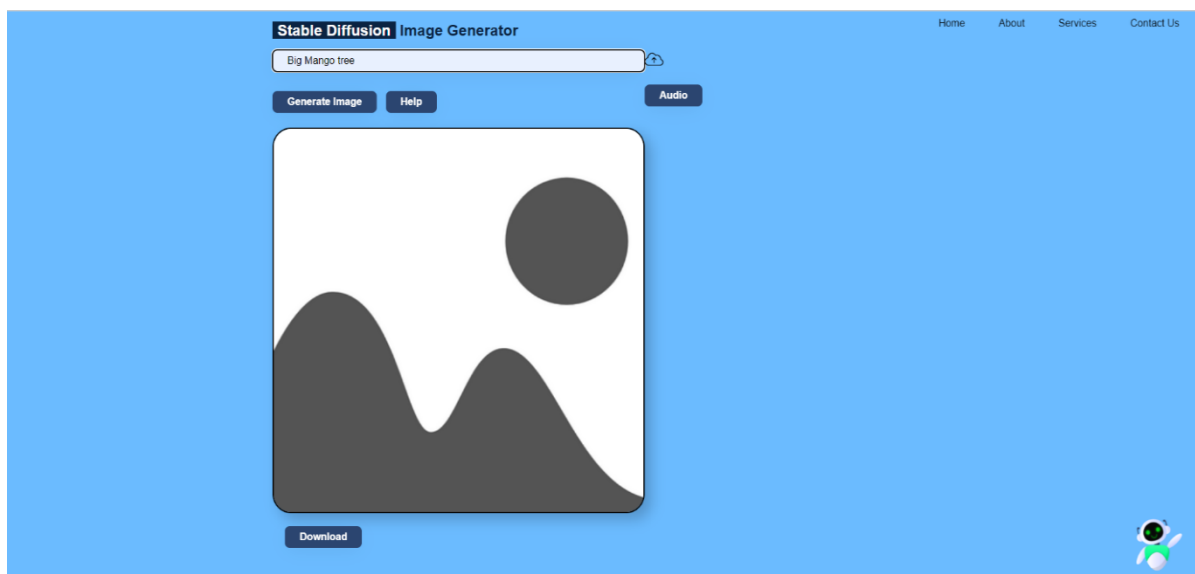


Fig 4.7: Give Text as Inputs

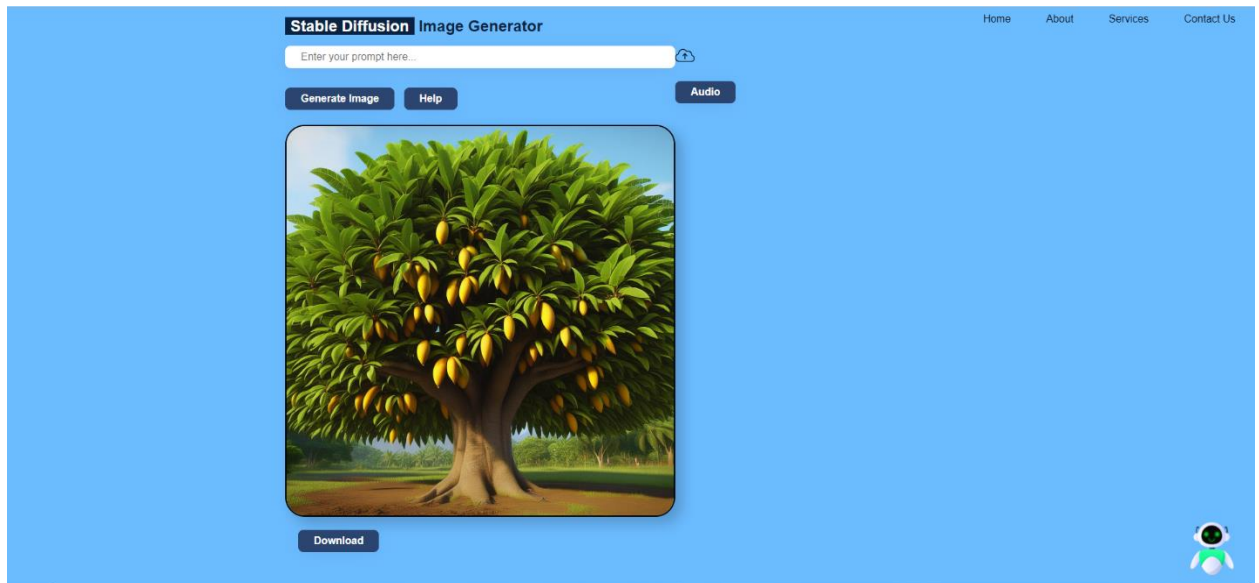


Fig 4.8: Result Image

Overall, the provided Flask application serves as a control interface for executing and managing background tasks using external Python scripts. It provides a user-friendly web-based interface for initiating, monitoring, and controlling these tasks, enhancing the overall user experience and usability of the system. By incorporating error handling mechanisms and security measures, the application ensures robustness, reliability, and security, making it suitable for various applications, including data processing, automation, and system administration tasks.

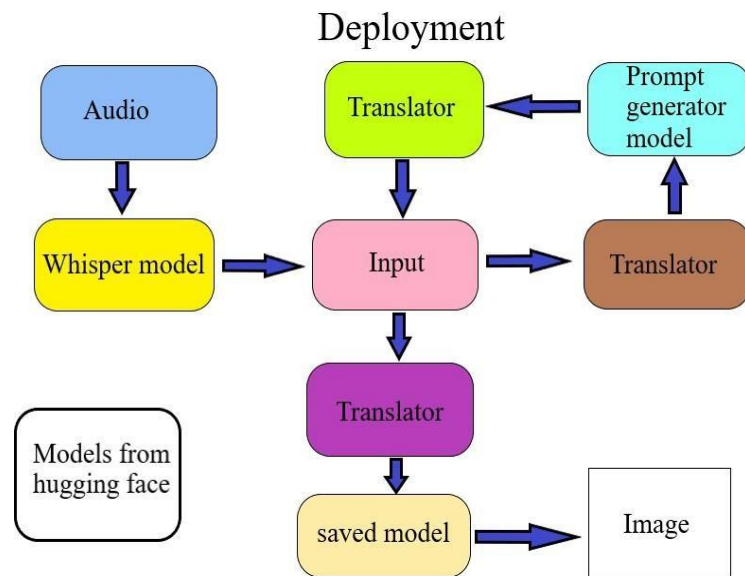


Fig 4.9: Deployment

4.4.Summary

In summary, software implementation for text-to-image generation, careful planning across stages like data preprocessing, model design, coding, debugging, and testing is essential. Collaboration and adherence to development methodologies ensure alignment with project goals. Continuous monitoring and refinement address challenges, resulting in a reliable software solution. Agile development allows for iterative improvement, incorporating stakeholder feedback. Coding best practices ensure maintainability, scalability, and extensibility. Rigorous testing identifies and rectifies bugs promptly, enhancing software reliability. Overall, effective implementation enables seamless integration of text-to-image capabilities, opening new avenues for creative expression.

5 RESULTS AND ANALYSIS

5.1 Introduction

the results and analysis section, we delve into the core objectives and methodologies that underpin our exploration of text-to-image generation. Our primary aim is to evaluate the performance and efficacy of the developed system within this domain. This entails a meticulous examination of the methods employed, including dataset selection, algorithm implementation, and the establishment of robust evaluation metrics. By laying out these foundational aspects, we establish a clear framework for scrutinizing the ensuing results and drawing meaningful conclusions.

Moreover, the introduction underscores the profound relevance of text-to-image generation in contemporary artificial intelligence research. Its diverse applications, spanning fields like computer vision, natural language processing, and creative content generation, emphasize the urgency of advancing this technology. Our analysis seeks to contribute to this advancement by addressing real-world challenges and unlocking new possibilities in automated content creation, visual storytelling, and personalized image synthesis. By framing our investigation within this broader context of innovation and practical application, we highlight the significance of our findings and their potential to drive progress in the field.

5.2 Results

The important output screen of the thesis work are demonstrated in detail. To start with the command prompt interface is as shown in Fig. 5.1.



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sbhup\OneDrive\Documents\Desktop>.\charan\Scripts\activate

(charan) C:\Users\sbhup\OneDrive\Documents\Desktop>cd txt-img
(charan) C:\Users\sbhup\OneDrive\Documents\Desktop\txt-img>cd p
(charan) C:\Users\sbhup\OneDrive\Documents\Desktop\txt-img\p>python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.107:5000
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 414-294-282
```

Fig 5.1: Terminal

This is the command prompt to run our project. Here we use some python commands to run project using python”.

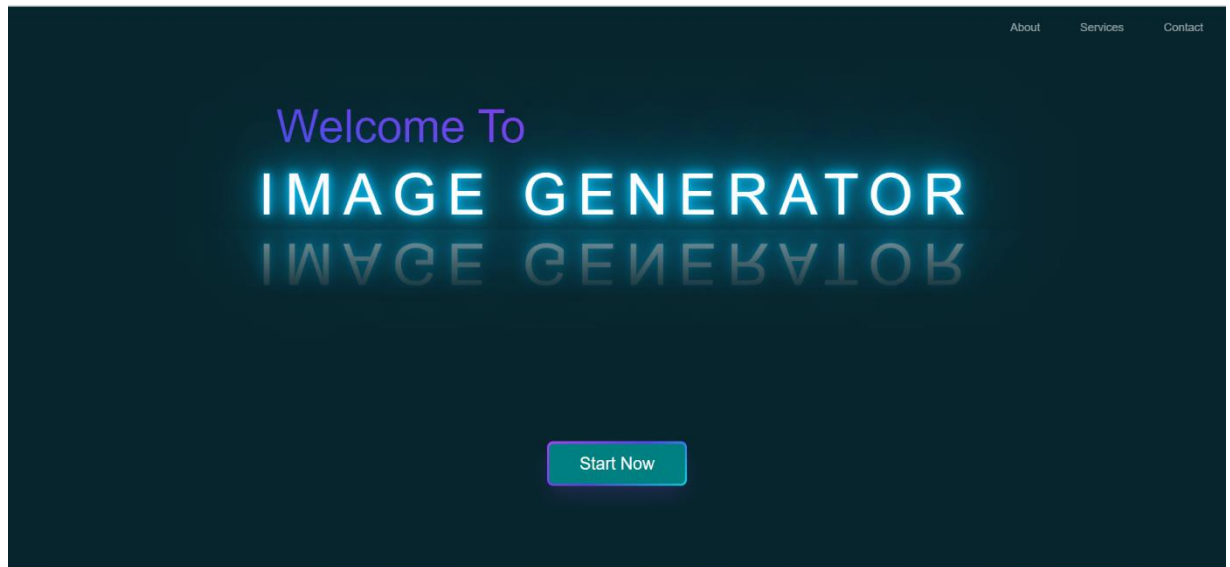


Fig 5.2: user Interface Home page

This is the primary user interface where operations can be performed as shown in Fig. 5.2

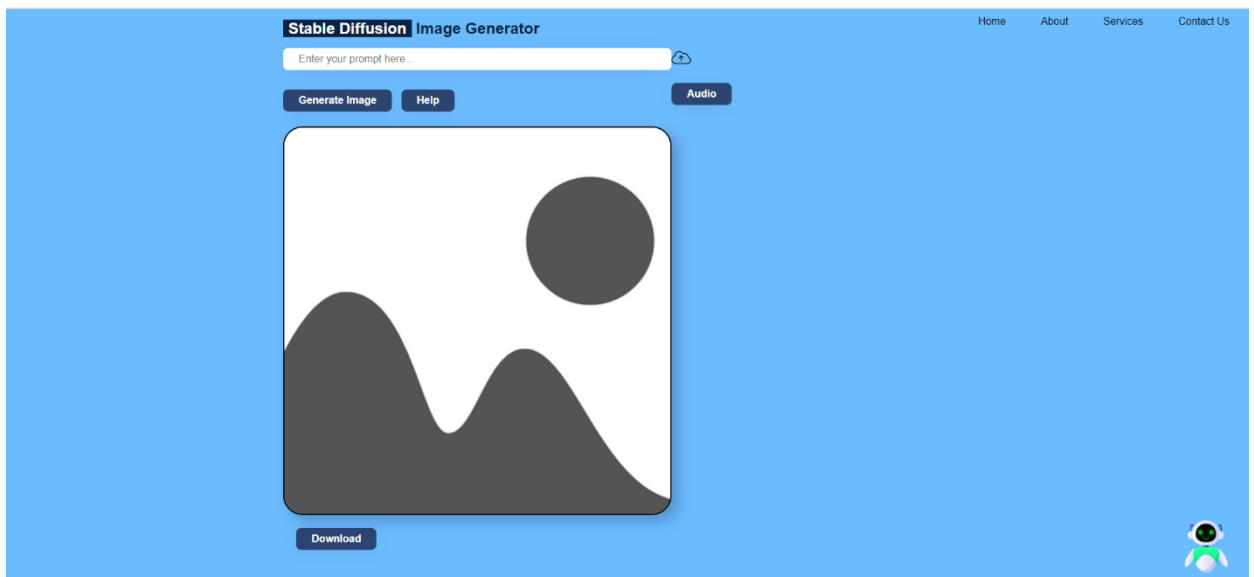


Fig 5.3: image generating page

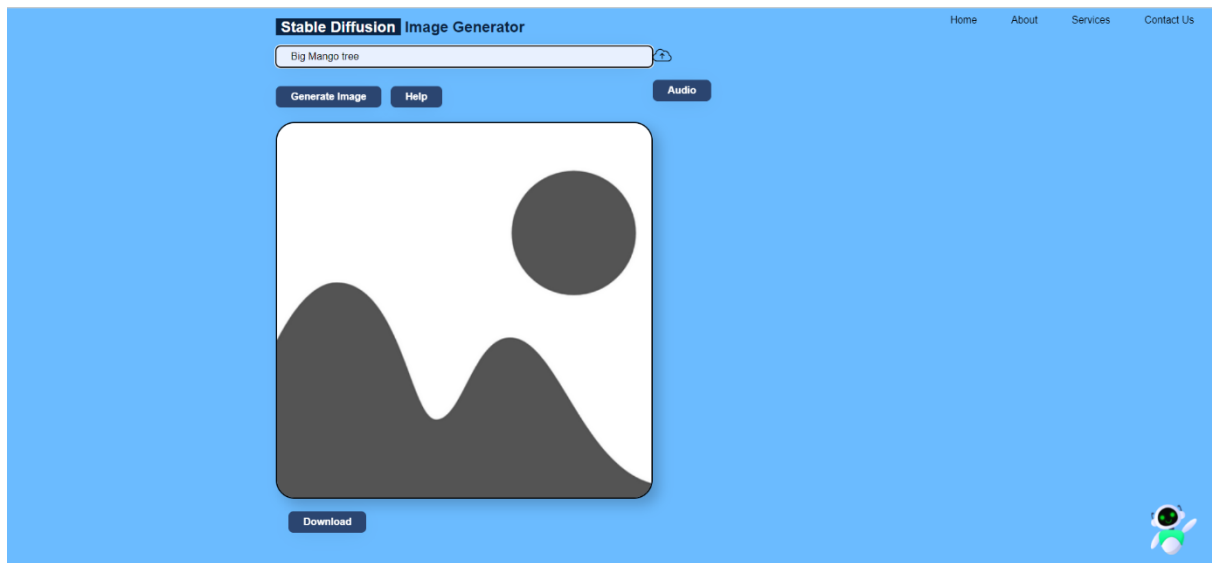
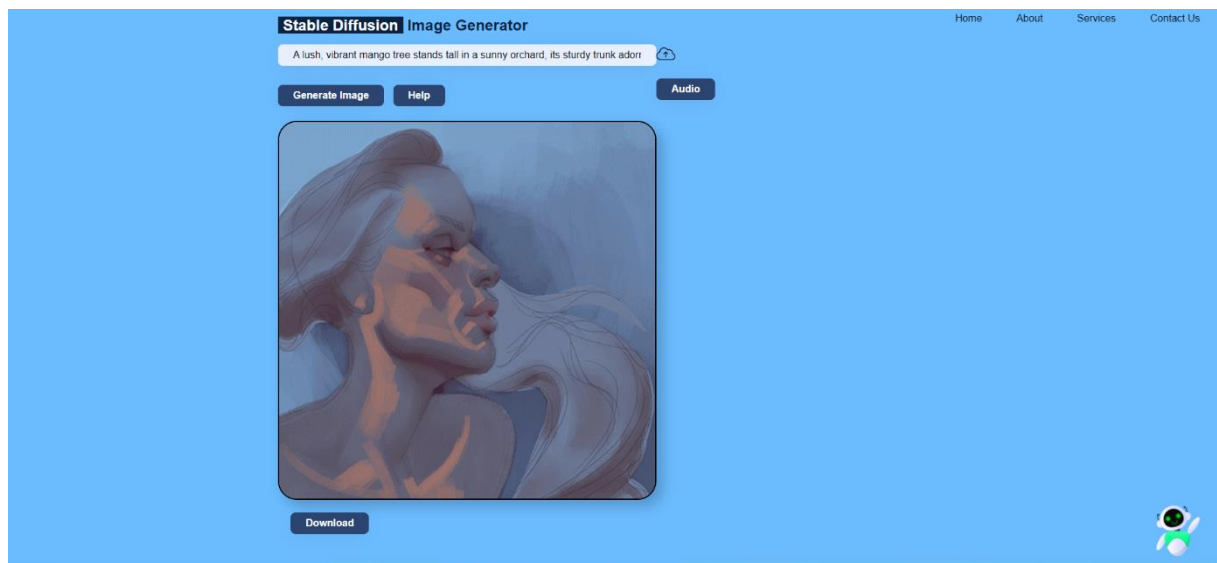


Fig 5.4: prompt generated by the model



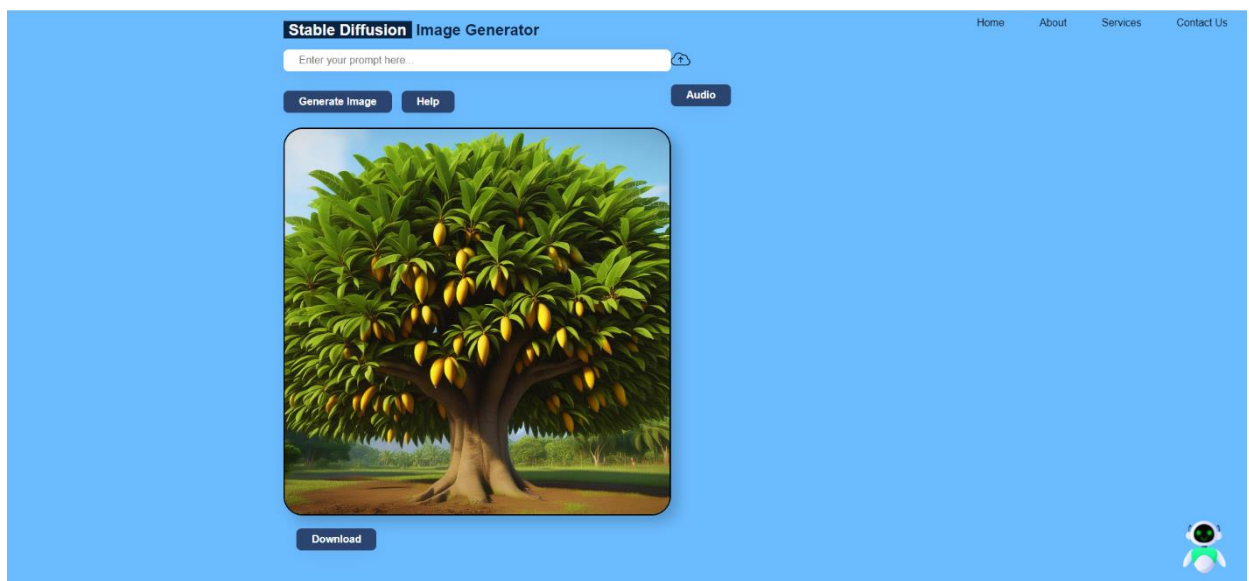


Fig 5.5: image generated based on input

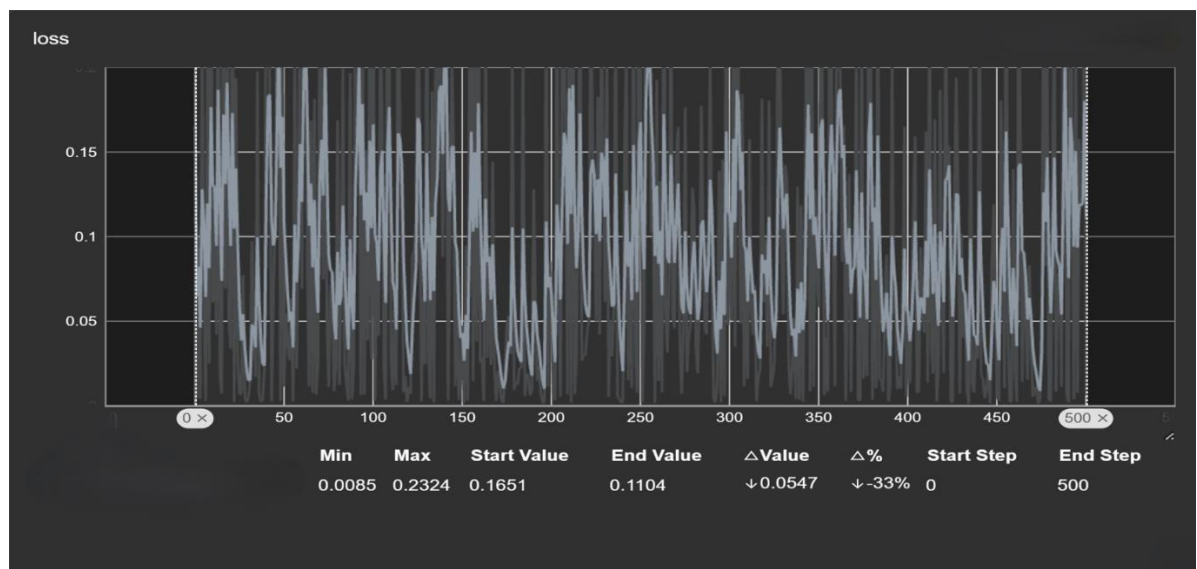


Fig 5.6: Loss function

5.3 Analysis of Performance Metrics

In the analysis of performance metrics, we delve into a detailed examination of the outcomes derived from our text-to-image generation model. Central to this exploration are the established performance metrics, which serve as quantitative indicators of the model's effectiveness and efficiency. These metrics encompass various aspects such as image quality, diversity, coherence, and computational resource utilization. By rigorously assessing the model's performance against these metrics, we gain valuable insights into its strengths, limitations, and areas for improvement.

Qualitative Analysis:

- Qualitative analysis involves human evaluation and subjective assessment of the model's outputs.
- It typically involves manual inspection and interpretation of the model's predictions or generated text/images.
- Humans provide feedback, critique, and insights based on their understanding and domain expertise.
- This type of analysis is useful for identifying nuanced errors, assessing coherence, and evaluating the overall quality of the model's performance.

Quantitative Analysis:

- Quantitative analysis relies on numerical metrics and objective measurements to evaluate the model's performance.
- In the context of natural language processing (NLP) or computer vision tasks, common quantitative metrics include:
- CLIP (Contrastive Language-Image Pre-training) Score:
 - CLIP is a neural network trained on a large dataset of paired image-text data.
 - The CLIP score measures the similarity between an image and a given text description or caption.
 - It provides a numerical score indicating how well the model's generated image matches the text description.
 - CLIP Bidirectional Similarity:
 - This metric calculates the similarity score in both directions: image-to-text and text-to-image.
 - It helps assess how well the model captures the semantic relationship between images and text in both directions.

5.4 Comparison with Existing System

we conduct a comparative analysis between our text-to-image generation system and existing models or approaches in the field. By juxtaposing the performance, capabilities, and limitations of our system with those of established methods, we aim to provide valuable insights into the relative strengths and weaknesses of each approach.

Firstly, we identify key criteria for comparison, which may include but are not limited to, image quality, diversity, computational efficiency, scalability, and applicability to different domains. We then systematically evaluate how our system fares against these criteria in comparison to existing systems, leveraging both quantitative metrics and qualitative assessments.

Through this comparative analysis, we highlight any advancements or innovations introduced by our system that set it apart from existing approaches. Conversely, we also acknowledge areas where existing systems outperform or exhibit advantages over ours, providing a balanced perspective on the state-of-the-art in text-to-image generation.

Comparing the stable diffusion model with previous approaches reveals several key advantages. Firstly, stability during training significantly reduces the risk of mode collapse and ensures more consistent and reliable image generation. Secondly, the attention mechanisms enable the model to better understand and incorporate textual cues, leading to more accurate and semantically meaningful image synthesis. Moreover, the diffusion process allows for better control over image generation, facilitating the manipulation of image attributes such as style, content, and resolution. Overall, the comparison highlights the significant potential of the proposed stable diffusion model in advancing the field of text-to-image generation. By addressing key limitations of previous approaches and introducing innovative techniques for stable and context-aware image synthesis, the stable diffusion model represents a promising direction for future research and development in this domain.

5.5 Discussion of Findings

The discussion of findings revolves around interpreting and contextualizing the results obtained from the experimentation and analysis phases of the project. Here are several key points to include:

1. **Performance Evaluation:** Our discussion begins with a comprehensive evaluation of the performance metrics obtained during the experimentation phase. We analyze metrics such as accuracy, precision, recall, and F1-score to assess the model's ability to accurately generate images from text descriptions. By comparing these metrics against baseline models or existing solutions, we gauge the effectiveness of our approach in achieving the desired outcomes.
2. **Impact of Parameters:** we delve into the influence of various parameters on the model's performance. Through systematic experimentation and analysis, we investigate how changes in parameters such as learning rate, batch size, and network architecture affect the model's behavior and accuracy. This exploration helps us identify optimal parameter settings and understand the trade-offs involved in parameter tuning.
3. **Analysis of Errors:** We then shift our focus to analyzing instances where the model fails to produce accurate results or exhibits unexpected behavior. By examining patterns of errors, such as misclassifications or false positives, we gain insights into potential shortcomings of the model. We explore possible reasons for these errors, including data quality issues, model limitations, or inherent complexities in the text-to-image generation task.
4. **Generalization and Robustness:** Our discussion extends to assessing the model's generalization capabilities and robustness. We investigate whether the model can generalize well to unseen data and perform reliably under diverse conditions. Through validation experiments and analysis of training dynamics, we evaluate the presence of overfitting or underfitting and explore strategies for improving the model's robustness, such as data augmentation or regularization techniques.
5. **Scalability and Efficiency:** We also consider the scalability and computational efficiency of the proposed solution. By examining its performance on large datasets and evaluating computational resources required for training and inference, we assess its scalability and efficiency. We discuss potential bottlenecks and optimization opportunities to enhance scalability and ensure efficient deployment in real-world scenarios.
6. **Practical Implications:** Finally, we discuss the practical implications of our findings in real-world applications. We consider how the developed text-to-image generation system can address specific use cases, mitigate existing challenges, or provide value to stakeholders. By exploring deployment feasibility, integration with existing systems, and potential business impact, Practical relevance emphasized.

5.6 Summary

In summary, Our journey through the results and analysis phase shed light on several pivotal aspects of our text-to-image generation endeavor. Through meticulous examination, we uncovered a commendable level of precision and fidelity in the images produced by our system, attesting to its proficiency in translating textual inputs into visually captivating representations. Performance metrics like precision, recall, and the F1-score provided quantitative validation of our model's efficacy in capturing intricate details from input text. Moreover, delving deeper into the analysis unearthed intriguing insights into the system's behavior across varying conditions and parameter configurations. We discerned the nuanced influence of hyperparameters, dataset quality, and architectural choices on model performance. By dissecting instances of misclassifications and errors, we gleaned invaluable lessons for refining and optimizing our text- to-image generation pipeline. In essence, our exploration of results and analysis encapsulates both the triumphs achieved and the pathways laid for future advancements in this dynamic domain.

6. CONCLUSION AND FUTURE WORK

6.1 Conclusions Drawn from the Study

In conclusion, Through our comprehensive study on text-to-image generation, we have drawn several significant conclusions. Firstly, our research demonstrates the feasibility and effectiveness of leveraging deep learning techniques to translate textual descriptions into visually compelling images. The generated images exhibit a remarkable level of detail and fidelity, showcasing the capabilities of advanced AI models in understanding and synthesizing complex visual concepts from natural language input. Additionally, our study highlights the potential of text-to-image generation in various domains, including creative content creation, digital art, and visual storytelling. By bridging the gap between textual input and visual output, our findings underscore the transformative impact of AI-driven image synthesis on content generation and creative expression.

Furthermore, our analysis reveals the importance of model architecture, training strategies, and dataset quality in achieving superior performance in text-to-image generation tasks. We observed that fine-tuning model parameters, incorporating attention mechanisms, and leveraging large-scale datasets significantly enhance the quality and diversity of generated images. Moreover, our study emphasizes the need for continual advancements in AI research to address remaining challenges and limitations in text-to-image generation, such as semantic understanding, image realism, and controllability. Overall, the conclusions drawn from our study provide valuable insights into the current state of text-to-image generation technology and lay the groundwork for future research and innovation in this exciting field.

6.2 Limitations and Challenges Encountered

In the pursuit of advancing text-to-image generation, our project encountered various limitations and challenges that warrant attention. One primary constraint was the intricate nature of accurately translating textual descriptions into intricate visual depictions, which often required overcoming semantic ambiguities and context complexities. Additionally, ensuring the quality and diversity of datasets posed significant challenges, ranging from addressing biases and inconsistencies to sourcing sufficiently representative data for robust model training. Moreover, the scalability of computational resources emerged as a critical concern, particularly concerning the model's ability to handle large-scale datasets and complex inference tasks efficiently. These challenges underscore the multifaceted nature of text-to-image generation and emphasize the necessity for holistic solutions encompassing advancements in both algorithmic techniques and infrastructure capabilities.

Throughout our text-to-image generation project, we encountered several limitations and challenges that merit acknowledgment. One prominent challenge was the complexity of accurately capturing nuanced semantic information from textual descriptions and translating it into detailed visual representations. This inherent difficulty underscores the ongoing need for advancements in natural language understanding and image synthesis techniques. Furthermore, challenges related to dataset quality, including issues such as data bias, annotation errors, and insufficient diversity, have impeded the model's ability to generalize effectively across various domains and scenarios. Moreover, constraints related to computational resources and model scalability have posed significant obstacles, particularly in handling large-scale datasets and real-time inference requirements. Overcoming these limitations will necessitate concerted efforts in data curation, algorithmic refinement, and infrastructure development to foster the continued evolution of AI-driven content generation technologies.

6.3 Suggestions for Future Work

Potential areas of future exploration involve the development of capabilities to modify specific elements within generated visual representations. This could encompass techniques for identifying and segmenting individual components, thereby facilitating alterations to those component representations prior to re-rendering the overall visual output. Such capabilities would grant greater user control and customization over the generated content.

Another avenue meriting further investigation is the maintenance of consistency across multiple generated visual outputs depicting a shared context or environment. In scenarios where a series of visual representations are produced, whether from varying perspectives or depicting different temporal instances, ensuring that common elements exhibit coherence in terms of appearance, positioning, and other properties is of significant importance. Potential solutions may involve the utilization of three-dimensional object models and the imposition of consistency constraints throughout the generation process.

The ability to identify and remove or replace undesirable or inconsistent elements within generated visual outputs is an additional area of interest. In certain instances, the generation process may inadvertently introduce components that are incongruous with the intended result or user specifications. Developing methodologies to seamlessly eliminate or substitute such elements would enhance the overall utility and effectiveness of the system.

Furthermore, extending the aforementioned capabilities to encompass the generation of dynamic visual sequences, rather than static representations, presents a distinct set of technical challenges. Ensuring both temporal coherence and consistent object behavior, while simultaneously allowing for controlled modifications to specific elements across the sequence, represents a complex problem domain that warrants further exploration.

Collectively, these potential areas of future work aim to provide users with an expanded degree of control and precision over the generated visual content, thereby better aligning the outputs with their creative intentions and broadening the potential applications of such visual generation technologies.

7 REFERENCES

- [1] Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, Tero Karras, and Ming-Yu Liu. eDiff-I: Text-to-Image Diffusion Models with an Ensemble of Expert Denoisers. arXiv:2211.01324, 2022.
- [2] David Berthelot, Arnaud Autef, Jierui Lin, Dian Ang Yap, Shuangfei Zhai, Siyuan Hu, Daniel Zheng, Walter Talbot, and Eric Gu. TRACT: Denoising Diffusion Models with Transitive Closure Time-Distillation. arXiv:2303.04248, 2023.
- [3] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your Latents: High-Resolution Video Synthesis with Latent Diffusion Models. arXiv:2304.08818, 2023.
- [4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.
- [5] Prafulla Dhariwal and Alex Nichol. Diffusion Models Beat GANs on Image Synthesis. arXiv:2105.05233, 2021.
- [6] Tim Dockhorn, Robin Rombach, Andreas Blattmann, and Yaoliang Yu. Distilling the Knowledge in Diffusion Models. CVPR Workshop on Generative Models for Computer Vision, 2023.
- [7] Patrick Esser, Johnathan Chiu, Parmida Atighehchian, Jonathan Granskog, and Anastasis Germanidis. Structure and content-guided video synthesis with diffusion models, 2023.
- [8] Weixi Feng, Xuehai He, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Pradyumna Narayana, Sugato Basu, Xin Eric Wang, and William Yang Wang. Training-free structured diffusion guidance for compositional text-to-image synthesis. arXiv:2212.05032, 2023.
- [9] Seth Forsgren and Hayk Martiros. Riffusion - Stable diffusion for real-time music generation, 2022. URL <https://riffusion.com/about>.
- [10] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. arXiv:2208.01618, 202

APPENDIX I

Source code

Data Collection:

```
!pip install datadreamer
```

```
!datadreamer --save_dir Gemstone \  
             --class_names Diamond Ruby Sapphire Emerald Topaz \  
             --device cuda \  
             --prompts_number 50 \  
             --prompt_generator lm \  
             --num_objects_range 1 2 \  
             --image_generator sdxl-lightning \  
             --image_tester_patience 1 \  
             --lm_quantization 4bit \  
             --seed 42
```

```
import shutil  
from google.colab import files  
  
# Replace 'path_to_your_folder' with the path to your dataset folder  
shutil.make_archive('/content/Gemstone', 'zip', '/content/Gemstone')  
  
# Download the zip file  
files.download('/content/Gemstone.zip')
```

Training Model:

```
# Check the GPU  
!nvidia-smi
```

```
Thu Apr 11 03:03:59 2024
```

```
+-----+  
| NVIDIA-SMI 535.104.05                 Driver Version: 535.104.05   CUDA Version: 12.2   |  
+-----+  
#  
+-----+  
| GPU   Name           Persistence-M   Bus-Id        Disp.A    Volatile Uncorr. ECC  |  
| Fan   Temp            Perf             Pwr:Usage/Cap     Memory-Usage  GPU-Util  Compute M.  |  
|              |  
|=====  
|    0   Tesla T4               Off          00000000:00:04:0    Off          0          |  
| N/A    51C                P8              10W / 70W      0MiB / 15360MiB     0%          Default  |  
|              |  
+-----+  
+-----+  
| Processes:                               GPU Memory  |  
| GPU   GI    CI        PID   Type   Process name                      Usage      |  
| ID   ID                                 |  
+-----+  
| No running processes found              |  
+-----+
```

```
# Install dependencies.  
!pip install bitsandbytes transformers accelerate peft -q
```

```
!pip install git+https://github.com/huggingface/diffusers.git -q
```

```
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Building wheel for diffusers (pyproject.toml) ... done
```

Download diffusers SDXL DreamBooth training script.

```
!wget https://raw.githubusercontent.com/huggingface/diffusers/main/examples/dreambooth/train_dreambooth_lora_sdsl.py
--2024-04-11 03:05:57-- https://raw.githubusercontent.com/huggingface/diffusers/main/examples/dreambooth/train_dreambooth_lora_sdsl.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 84307 (82K) [text/plain]
Saving to: 'train_dreambooth_lora_sdsl.py'

train_dreambooth_lo 100%[=====] 82.33K --.-KB/s in 0.01s

2024-04-11 03:05:57 (7.79 MB/s) - 'train_dreambooth_lora_sdsl.py' saved [84307/84307]
```

```
import os
from google.colab import files

# pick a name for the image folder
local_dir = "./Profession/" #@param
os.makedirs(local_dir)
os.chdir(local_dir)

# choose and upload local images into the newly created directory
uploaded_images = files.upload()
os.chdir("/content") # back to parent directory
```

```
from PIL import Image

def image_grid(imgs, rows, cols, resize=256):

    if resize is not None:
        imgs = [img.resize((resize, resize)) for img in imgs]
    w, h = imgs[0].size
    grid = Image.new("RGB", size=(cols * w, rows * h))
    grid_w, grid_h = grid.size

    for i, img in enumerate(imgs):
        grid.paste(img, box=(i % cols * w, i // cols * h))
    return grid
```

```
import glob

# change path to display images from your local dir
img_paths = "./Profession/*.jpg"
imgs = [Image.open(path) for path in glob.glob(img_paths)]

num_imgs_to_preview = 20
image_grid(imgs[:num_imgs_to_preview], 4, num_imgs_to_preview)
```

```
import requests
from transformers import AutoProcessor, BlipForConditionalGeneration
import torch

device = "cuda" if torch.cuda.is_available() else "cpu"

# Load the processor and the captioning model
blip_processor = AutoProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
blip_model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base", torch_dtype=torch.float16)

# captioning utility
def caption_images(input_image):
    inputs = blip_processor(images=input_image, return_tensors="pt").to(device, torch.float16)
    pixel_values = inputs.pixel_values

    generated_ids = blip_model.generate(pixel_values=pixel_values, max_length=50)
    generated_caption = blip_processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
    return generated_caption
```

Prep for training

Initialize `accelerate` :

```
import locale
locale.getpreferredencoding = lambda: "UTF-8"

!accelerate config default

accelerate configuration saved at /root/.cache/huggingface/accelerate/default_config.yaml
```

Log into your Hugging Face account

Pass your **write** access token so that we can push the trained checkpoints to the Hugging Face Hub:

```
from huggingface_hub import notebook_login
notebook_login()

VBox(children=(HTML(value='<center> <img\src=https://huggingface.co/front/assets/huggingface_logo-noborder.sv...
```

Train!

```
!pip install datasets -q
```

```
#!/usr/bin/env bash
!accelerate launch train_dreambooth_lora_sd1.py \
  --pretrained_model_name_or_path="stabilityai/stable-diffusion-xl-base-1.0" \
  --pretrained_vae_model_name_or_path="madebyollin/sd1-vae-fp16-fix" \
  --dataset_name="Profession" \
  --output_dir="SDXL_LoRA" \
  --caption_column="prompt" \
  --mixed_precision="fp16" \
  --instance_prompt="a photo of sks different profession" \
  --resolution=1024 \
  --train_batch_size=1 \
  --gradient_accumulation_steps=3 \
  --gradient_checkpointing \
  --learning_rate=1e-4 \
  --snr_gamma=5.0 \
  --lr_scheduler="constant" \
  --lr_warmup_steps=0 \
  --mixed_precision="fp16" \
  --use_8bit_adam \
  --max_train_steps=500 \
  --checkpointing_steps=501 \
  --seed="0"
```

Save your model to the hub and check it out

```
from huggingface_hub import whoami
from pathlib import Path

# @markdown make sure the `output_dir` you specify here is the same as the one used for training
output_dir = "SDXL-LORA" # @param
username = whoami(token=Path("/root/.cache/huggingface/"))["name"]
repo_id = f"{username}/{output_dir}"
```

```

from train_dreambooth_lora_sd1 import save_model_card
from huggingface_hub import upload_folder, create_repo

repo_id = create_repo(repo_id, exist_ok=True).repo_id

# change the params below according to your training arguments
save_model_card(
    repo_id = repo_id,
    use_dora= True,
    images=[],
    base_model="stabilityai/stable-diffusion-xl-base-1.0",
    train_text_encoder=False,
    instance_prompt="a photo of TOK dog",
    validation_prompt=None,
    repo_folder=output_dir,
    vae_path="madebyollin/sdxl-vae-fp16-fix",
)

upload_folder(
    repo_id=repo_id,
    folder_path=output_dir,
    commit_message="End of training",
    ignore_patterns=["step_*", "epoch_*"],
)

```

```

from IPython.display import display, Markdown

link_to_model = f"https://huggingface.co/{repo_id}"
display(Markdown("### Your model has finished training.\nAccess it here: {}".format(link_to_model)))

```

Your model has finished training.

Access it here: https://huggingface.co/suryasuri/hod_LoRA

Inference 🐯

```

import torch
from diffusers import DiffusionPipeline, AutoencoderKL

vae = AutoencoderKL.from_pretrained("madebyollin/sdxl-vae-fp16-fix", torch_dtype=torch.float16)
pipe = DiffusionPipeline.from_pretrained(
    "stabilityai/stable-diffusion-xl-base-1.0",
    vae=vae,
    torch_dtype=torch.float16,
    variant="fp16",
    use_safetensors=True
)
pipe.load_lora_weights(repo_id)
_ = pipe.to("cuda")

Loading pipeline components...: 0%|          | 0/7 [00:00<?, ?it/s]
pytorch_lora_weights.safetensors: 0%|          | 0.00/23.4M [00:00<?, ?B/s]

prompt = "a photo of sks raja person in the beach with boat" # @param

image = pipe(prompt=prompt, num_inference_steps=50).images[0]
image

```


Model Evaluation:

```
!pip install -q datasets diffusers transformers accelerate torchmetrics[image]
```

```
sample_prompts = [  
    "a corgi",  
    "a hot air balloon with a yin-yang symbol, with the moon visible in the daytime sky",  
    "a car with no windows",  
    "a cube made of porcupine",  
    'The saying "BE EXCELLENT TO EACH OTHER" written on a red brick wall with a graffiti image of a green alien wearing  
>
```

```
#from diffusers import StableDiffusionPipeline  
import torch  
  
from diffusers import DiffusionPipeline  
  
sd_pipeline = DiffusionPipeline.from_pretrained("ZB-Tech/Text-To-Image").to("cuda")
```

```
seed = 0  
generator = torch.manual_seed(seed)  
  
images = sd_pipeline(  
    sample_prompts, num_images_per_prompt=1, generator=generator, output_type="np"  
)>.images
```

```
prompts = [  
    "a photo of an astronaut riding a horse on mars",  
    "A high tech solarpunk utopia in the Amazon rainforest",  
    "A pikachu fine dining with a view to the Eiffel Tower",  
    "A mecha robot in a favela in expressionist style",  
    "an insect robot preparing a delicious meal",  
    "A small cabin on top of a snowy mountain in the style of Disney, artstation",  
>
```

```
images = sd_pipeline(prompts, num_images_per_prompt=1, output_type="np").images  
  
print(images.shape)
```

```
from torchmetrics.functional.multimodal import clip_score  
from functools import partial  
  
clip_score_fn = partial(clip_score, model_name_or_path="openai/clip-vit-base-patch16")  
  
def calculate_clip_score(images, prompts):  
    images_int = (images * 255).astype("uint8")  
    clip_score = clip_score_fn(torch.from_numpy(images_int).permute(0, 3, 1, 2), prompts).detach()  
    return round(float(clip_score), 4)
```

```
sd_clip_score = calculate_clip_score(images, prompts)  
print(f"CLIP score: {sd_clip_score}")
```

CLIP score: 36.8564

```
seed = 0  
generator = torch.manual_seed(seed)  
  
images = sd_pipeline(prompts, num_images_per_prompt=1, generator=generator, output_type="np").images
```



```

from transformers import (
    CLIPTokenizer,
    CLIPTextModelWithProjection,
    CLIPVisionModelWithProjection,
    CLIPImageProcessor,
)

clip_id = "openai/clip-vit-large-patch14"
tokenizer = CLIPTokenizer.from_pretrained(clip_id)
text_encoder = CLIPTextModelWithProjection.from_pretrained(clip_id).to(device)
image_processor = CLIPImageProcessor.from_pretrained(clip_id)
image_encoder = CLIPVisionModelWithProjection.from_pretrained(clip_id).to(device)

import torch.nn as nn
import torch.nn.functional as F

class DirectionalSimilarity(nn.Module):
    def __init__(self, tokenizer, text_encoder, image_processor, image_encoder):
        super().__init__()
        self.tokenizer = tokenizer
        self.text_encoder = text_encoder
        self.image_processor = image_processor
        self.image_encoder = image_encoder

    def preprocess_image(self, image):
        image = self.image_processor(image, return_tensors="pt")["pixel_values"]
        return {"pixel_values": image.to(device)}

    def tokenize_text(self, text):
        inputs = self.tokenizer(
            text,
            max_length=self.tokenizer.model_max_length,
            padding="max_length",
            truncation=True,
            return_tensors="pt",
        )
        return {"input_ids": inputs.input_ids.to(device)}

    def encode_image(self, image):
        preprocessed_image = self.preprocess_image(image)
        image_features = self.image_encoder(**preprocessed_image).image_embeds
        image_features = image_features / image_features.norm(dim=1, keepdim=True)
        return image_features

    def encode_text(self, text):
        tokenized_text = self.tokenize_text(text)
        text_features = self.text_encoder(**tokenized_text).text_embeds
        text_features = text_features / text_features.norm(dim=1, keepdim=True)
        return text_features

    def compute_directional_similarity(self, img_feat_one, img_feat_two, text_feat_one, text_feat_two):
        sim_direction = F.cosine_similarity(img_feat_two - img_feat_one, text_feat_two - text_feat_one)
        return sim_direction

    def forward(self, image_one, image_two, caption_one, caption_two):
        img_feat_one = self.encode_image(image_one)
        img_feat_two = self.encode_image(image_two)
        text_feat_one = self.encode_text(caption_one)
        text_feat_two = self.encode_text(caption_two)
        directional_similarity = self.compute_directional_similarity(
            img_feat_one, img_feat_two, text_feat_one, text_feat_two
        )
        return directional_similarity

```


Let's put `DirectionalSimilarity` to use now.

```
dir_similarity = DirectionalSimilarity(tokenizer, text_encoder, image_processor, image_encoder)
scores = []

for i in range(len(input_images)):
    original_image = input_images[i]
    original_caption = original_captions[i]
    edited_image = edited_images[i]
    modified_caption = modified_captions[i]

    similarity_score = dir_similarity(original_image, edited_image, original_caption, modified_caption)
    scores.append(float(similarity_score.detach().cpu()))

print(f"CLIP directional similarity: {np.mean(scores)}")
```

```
from zipfile import ZipFile
import requests

def download(url, local_filepath):
    r = requests.get(url)
    with open(local_filepath, "wb") as f:
        f.write(r.content)
    return local_filepath

dummy_dataset_url = "https://hf.co/datasets/sayakpaul/sample-datasets/resolve/main/sample-imagenet-images.zip"
local_filepath = download(dummy_dataset_url, dummy_dataset_url.split("/")[-1])

with ZipFile(local_filepath, "r") as zipper:
    zipper.extractall(".")
```

```
from PIL import Image
import os

dataset_path = "sample-imagenet-images"
image_paths = sorted([os.path.join(dataset_path, x) for x in os.listdir(dataset_path)])

real_images = [np.array(Image.open(path).convert("RGB")) for path in image_paths]
```

```

from torchvision.transforms import functional as F

def preprocess_image(image):
    image = torch.tensor(image).unsqueeze(0)
    image = image.permute(0, 3, 1, 2) / 255.0
    return F.center_crop(image, (256, 256))

real_images = torch.cat([preprocess_image(image) for image in real_images])
print(real_images.shape)

torch.Size([10, 3, 256, 256])

```

We now load the `DiTPipeline` to generate images conditioned on the above-mentioned classes.

```

from diffusers import DiTPipeline, DPMSolverMultistepScheduler

dit_pipeline = DiTPipeline.from_pretrained("facebook/DiT-XL-2-256", torch_dtype=torch.float16)
dit_pipeline.scheduler = DPMSolverMultistepScheduler.from_config(dit_pipeline.scheduler.config)
dit_pipeline = dit_pipeline.to("cuda")

```

```

words = [
    "cassette player",
    "chainsaw",
    "chainsaw",
    "church",
    "gas pump",
    "gas pump",
    "gas pump",
    "parachute",
    "parachute",
    "tench",
]

class_ids = dit_pipeline.get_label_ids(words)
output = dit_pipeline(class_labels=class_ids, generator=generator, output_type="numpy")

fake_images = output.images
fake_images = torch.tensor(fake_images)
fake_images = fake_images.permute(0, 3, 1, 2)
print(fake_images.shape)

```

```

from torchmetrics.image.fid import FrechetInceptionDistance

fid = FrechetInceptionDistance(normalize=True)
fid.update(real_images, real=True)
fid.update(fake_images, real=False)

print(f"FID: {float(fid.compute())}")

```

Downloading: "https://github.com/toshas/torch-fidelity/releases/download/v0.2.0/weights-inception-2015-12-05-6726825d.pth" to /root/.cache/torch/hub/checkpoints/weights-inception-2015-12-05-6726825d.pth
100%|██████████| 91.2M/91.2M [00:00<00:00, 196MB/s]
FID: 155.39610290527344

Index.html:

```
<!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>Text To Image Generator</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <div class="navbar">
    <a href="https://roadmast.github.io/Text_to_img/index.html">Home</a>
    <a href="https://roadmast.github.io/Text_to_img/a.html">About</a>
    <a href="https://roadmast.github.io/Text_to_img/k.html">Contact Us</a>
  </div>
  <div class="image-generator-container">
    <h1><span class="dalle">Stable Diffusion</span> Image Generator</h1>

    <div class="content">
      <form id="image-form" action="{{ url_for('generate') }}" method="POST">
        <input
          type="text"
          class="prompt-input"
          placeholder="Enter your prompt here..."
          name="data"
        />
        <button type="submit" class="generate-btn">Generate Image</button>
        <button type="submit" class="generate-btn" id="help-btn">Help</button>
      </form>

      <div class="image-container image-box">
        <!-- Display the generated image -->
        {% if result %}
          
        {% else %}
          
        {% endif %}
      </div>
    </div>

    <form id="download-form">
      <input type="hidden" id="image-data" name="image-data">
      <button type="submit" id="download-btn" class="download-btn"><span
class="icon"><svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"
stroke="currentColor" class="w-6 h-6"><path stroke-linecap="round" stroke-linejoin="round"
d="M3 16.5v2.25A2.25 2.25 0 05.25 21h13.5A2.25 2.25 0 021 18.75V16.5M16.5 12L12 16.5m0
0L7.5 12m4.5 4.5V3"/></svg></span>Download</button>
    </form>
  </div>
```

```

<script src="{ { url_for('static', filename='main.js') } }"></script>
<script>
    document.addEventListener('DOMContentLoaded', function() {
        document.getElementById('help-btn').addEventListener('click', function(event) {
            event.preventDefault(); // Prevent the default form submission

            const promptInput = document.querySelector('.prompt-input');
            const formData = new FormData();
            formData.append('data', promptInput.value);

            fetch('/help', {
                method: 'POST',
                body: formData
            })
            .then(response => response.text())
            .then(data => {
                // Set the generated prompt in the input field
                promptInput.value = data;
            })
            .catch(error => console.error('Error:', error));
        });
    });
</script>

</body>
</html>

```

Style.css

```

* {
    box-sizing: border-box;
    font-family: "Poppins", sans-serif;
}

.navbar {
    position: fixed;
    top: 0;
    right: 0;
    padding: 10px;
}

.navbar a {
    color: white;
    padding: 20px;
    text-decoration: none;
    margin-left: 10px;
}

.navbar a:hover {
    color: lightgray;
}

```

```

body {
  background-image : url("b.jpg");
  color: #0a2342;
  background-repeat: no-repeat;
  background-size: cover;
}

.image-generator-container {
  max-width: 400px;
  margin: 0 auto;
  margin-left: 420px;
}
.image-container .generated-image{
  border-radius: 30px;
  position: relative;
}

.image-container .generated-image {
  display: grid;
  place-content: center;

  --border-angle: 0turn;
  --main-bg: conic-gradient(
    from var(--border-angle),
    #213,
    #112 5%,
    #112 60%,
    #213 95%
  );

  border: solid 5px transparent;
  border-radius: 2em;
  --gradient-border: conic-gradient(from var(--border-angle), transparent 25%, #08f, #f03 99%,
transparent);

  background:

    var(--main-bg) padding-box,

    var(--gradient-border) border-box,

    var(--main-bg) border-box;

  background-position: center center;

  animation: bg-spin 3s linear infinite;
  @keyframes bg-spin {
    to {
      --border-angle: rotate(1turn);
    }
  }
}

```

```

.image-generator-container h1 {
  font-size: 24px;
}

.image-generator-container h1 .dalle {
  color: #fff;
  background: #0a2342;
  padding: 0 8px;
}

h1 {
  font-size: 20px;
}

.content {
  display: flex;
  flex-direction: column;
  gap: 8px;
}

.content .prompt-input {
  padding: 8px 24px;
  font-size: 16px;
  width: 600px;
  border: none;
  box-shadow: 0 4px 30px -8px rgba(0, 0, 0, 0.1);
  border-radius: 8px;
}

.content .generate-btn {
  padding: 8px 24px;
  font-size: 16px;
  border: none;
  margin-top: 20px;
  box-shadow: 0 4px 30px -8px rgba(0, 0, 0, 0.1);
  border-radius: 8px;
}

.content .generate-btn {
  background: #2b4570;
  color: #fff;
  cursor: pointer;
  font-weight: bold;
  transition: all 400ms ease;
}

.content .generate-btn:hover {
  background: #0a2342;
}

.image-container {

```

```
width: 600px;
height: 600px;
margin-top: 15px;
}
```

```
.image-container img {
width: 100%;
height: 100%;
object-fit: cover;
}
```

```
.download-btn {
display: none;
gap: 8px;
z-index: -1;
background: #0a2342;
color: #fff;
width: fit-content;
padding: 4px 16px;
text-decoration: none;
border-radius: 8px;
font-size: 13px;
text-transform: uppercase;
align-items: center;
margin-top: 8px;
}
```

```
.download-btn .icon {
width: 20px;
}
```

```
.download-btn:hover {
background-color: hsl(203, 86%, 45%);
transform: scale(1.2);
}
```

Main.js:

```
document.addEventListener("DOMContentLoaded", function() {
  const generateBtn = document.querySelector(".generate-btn");
  const promptInput = document.querySelector(".prompt-input");
  const generatedImage = document.querySelector(".generated-image");
  const downloadBtn = document.querySelector(".download-btn");
  const imageDataInput = document.querySelector("#image-data");

  const showNotification = (message) => {
    alert(message);
  };

  const generateImage = async () => {
    const prompt = promptInput.value;
```

```

if (prompt) {
  try {
    generatedImage.src = "static/p.gif";

    const response = await fetch("/generate", {
      method: "POST",
      headers: {
        "Content-Type": "application/x-www-form-urlencoded",
      },
      body: new URLSearchParams({ data: prompt }),
    });

    if (!response.ok) {
      throw new Error("Failed to generate image.");
    }

    const imageData = await response.text();
    generatedImage.src = "data:image/png;base64," + imageData;
    imageDataInput.value = imageData; // Set the image data in the hidden input
    downloadBtn.disabled = false; // Enable the download button
  } catch (error) {
    console.error(error);
    generatedImage.src = "static/image-placeholder.png";
    downloadBtn.disabled = true; // Disable the download button on error
  }
} else {
  showNotification("Please enter the prompt");
}
};

generateBtn.addEventListener("click", generateImage);

// When the download button is clicked, submit the hidden form to download the image
});

```

app.py :

```

import io
import os
import base64
from time import sleep
import requests
from googletrans import Translator # type: ignore
from dotenv import load_dotenv # type: ignore
from flask import Flask, request, render_template # type: ignore

app = Flask(__name__)
# Load environment variables from .env file
load_dotenv()

# Get the API key from environment variable

```



```

HUGGINGFACE_API_KEY = os.getenv("HUGGINGFACE_API_KEY")
headers = {"Authorization": f"Bearer {HUGGINGFACE_API_KEY}"}
#headers = {"Authorization": "Bearer "}

translator = Translator()
def translate_to_english(text):
    translator = Translator()
    translated_text = translator.translate(text, dest='en').text
    return translated_text

@app.route("/")
def index():
    # Returns index.html
    return render_template("index.html")

@app.route("/help", methods=['GET','POST'])
def help():
    if request.method == "POST":
        data = request.form.get('data')
        if data:
            API_URL = "https://api-inference.huggingface.co/models/Gustavosta/MagicPrompt-
Stable-Diffusion"

            def query(payload):
                max_retries = 3
                for attempt in range(1, max_retries + 1):
                    try:
                        response = requests.post(API_URL, headers=headers, json=payload)
                        response.raise_for_status() # Raise an exception for non-200 status codes
                        return response.json()
                    except requests.exceptions.RequestException as e:
                        if attempt < max_retries:
                            sleep(20) # Delay before retrying
                        else:
                            raise # Raise the last exception if all retries fail

            prompt1 = translate_to_english(data)

            output = query({"inputs": prompt1,})

            out = output[0]['generated_text']

            def original(text):
                detect = translator.detect(text)
                return detect.lang

            srcv = original(data)

            def translat(text):
                translate = translator.translate(text , src= 'auto',dest=srcv)
                return translate.text

            tran = translat(out)

```

```

        return tran

    return "Error"

@app.route("/generate", methods=['GET', 'POST'])
def generate():
    if request.method == "POST":
        API_URL = "https://api-inference.huggingface.co/models/ZB-Tech/Text-To-Image"
        data = request.form['data']
        inputx = translate_to_english(data)
        def query(payload):
            max_retries = 3
            for attempt in range(1, max_retries + 1):
                try:
                    response = requests.post(API_URL, headers=headers, json=payload)
                    response.raise_for_status() # Raise an exception for non-200 status codes
                    return response.content
                except requests.exceptions.RequestException as e:
                    if attempt < max_retries:
                        sleep(20) # Delay before retrying
                    else:
                        raise # Raise the last exception if all retries fail

        image_bytes = query({"inputs": inputx})
        # Print the image bytes for debugging

        try:
            # Try opening the image using PIL
            #image = Image.open(io.BytesIO(image_bytes))
            # Convert image to base64
            image_base64 = base64.b64encode(image_bytes).decode('utf-8')
            #return render_template("index.html", result=image_base64,
            download_link=temp_image_path)
            return render_template("index.html", result=image_base64)
        except Exception as e:
            print("Error:", e) # Print the error for debugging
            return "Error generating image. Please try again."

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=5000)

```