**Image Generation**

|  |  |
| --- | --- |
| **Name of the Employee** | **Employee Id** |
| Deepthi Chiruvella | 46274989 |
| Daneesh Bari | 46300663 |
| Sarang Vartak | 46220330 |
| Vishnu Devbhaktuni | 46225190 |
| Aniruddha Bhalerao | 46221026 |

**Table of Contents**

**1.Introduction**

**2.Project overview**

**3.Requirements**

**4.Project Architecture**

**5.Code Module**

**6.Challenges and Solutions**

**7.Features**

**8**.**Future Scope**

**9.Conclusion**

**Introduction**

In a world where image editing often requires complex software and advanced programming skills, many individuals face barriers in enhancing or fixing their pictures creatively. Recognizing this challenge, our team is dedicated to developing an innovative solution that simplifies image editing for everyone. By leveraging cutting-edge technology and intuitive design, we aim to provide users with a user-friendly platform that empowers them to effortlessly transform, restore, and upscale their images without the need for extensive technical knowledge.

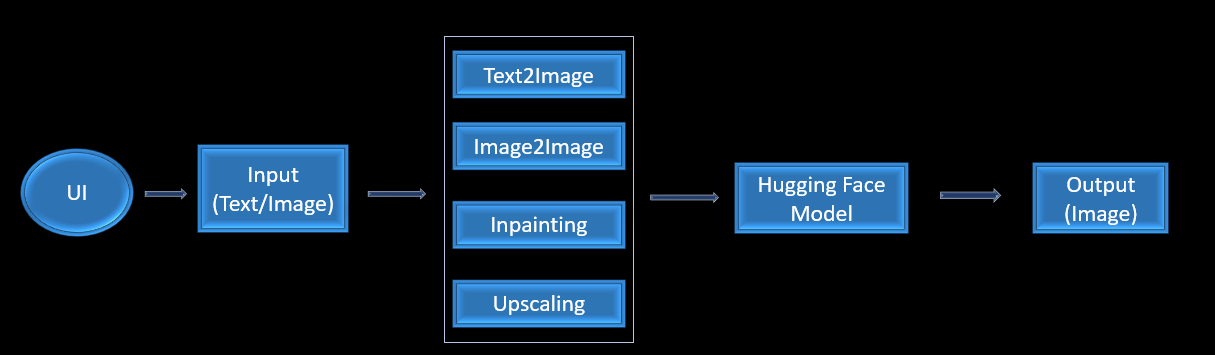
**Project overview**

Our project centers on the development of an application tailored to meet the diverse image editing needs of users. The primary objective is to create a platform that democratizes the process of enhancing and manipulating images, making it accessible to individuals with varying levels of expertise. Leveraging state-of-the-art models from Hugging Face, our application offers an array of powerful features designed to streamline the editing process and produce high-quality results.

**Requirements**

* A web browser to access the application
* Computer Internet Connection:  
     - You need a computer with a decent graphics card (GPU) and a stable internet connection.
* Web Browser:  
     - You'll access the application through a web browser like Chrome or Firefox.
* Python Environment:  
     - Make sure you have Python installed on your computer.  
     - You'll also need to install Gradio, a tool for creating the interactive interface.
* Google Colab Account:  
     - Sign up for a Google account if you don't have one.  
     - This account lets you use Google Colab, which helps with the heavy lifting of image processing using its GPU power.
* Operating System:  
     - The application works on Windows, macOS, and Linux.
* Browser Compatibility:  
     - Ensure your web browser is up-to-date for the best experience.

**Project Architecture**

****

**Code Module**

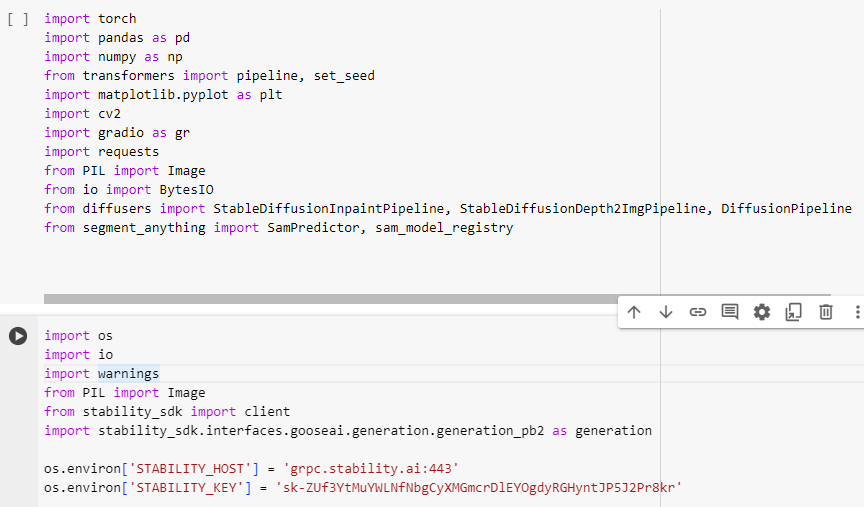
Required Packages:

Below are the required libraries to install-  


* `diffusers` and `transformers`: Libraries for working with generative AI models, with `transformers` specifically focusing on transformer-based models.
* `gradio`: A library for quickly creating UIs for machine learning models.
* `accelerate`: A library for accelerating PyTorch model training.
* `torch`, `torchvision`, `numpy`, `opencv-python`, `Pillow`, `basicsr`, `facexlib`, `gfpgan`, `tqdm`, `realesrgan`: Libraries for various tasks in deep learning, computer vision, and generative modeling.
* `stability-sdk`: A software development kit for building stable and reliable machine learning models.
* `scipy`, `ftfy`: Libraries for scientific computing and text processing.
* `segment-anything`: A library for segmenting objects in images using neural networks.
* `weights/sam\_vit\_h\_4b8939.pth`: Pre-trained weights for a segmentation model based on a Vision Transformer architecture.

Libraries to Import:

* + `torch`: PyTorch, a popular deep learning framework.
  + `pandas`: A library for data manipulation and analysis in Python.
  + `numpy`: A fundamental package for scientific computing with Python.
  + `transformers`: A library for natural language understanding (NLU) and natural language generation (NLG) using pre-trained models.
  + `matplotlib.pyplot`: A plotting library for Python programming language and its numerical mathematics extension NumPy.
  + `cv2`: OpenCV, a library of programming functions mainly aimed at real-time computer vision.



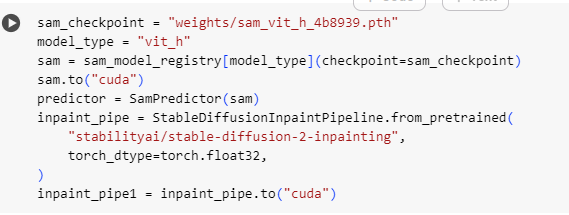
* + `gradio`: A library for creating UIs for machine learning models.
  + `requests`: A simple HTTP library for Python, allowing for sending HTTP requests.
  + `PIL.Image`: Python Imaging Library, a library for adding image processing capabilities to Python.
  + `BytesIO`: A class for reading and writing bytes data in memory.
  + `diffusers`: A library for working with generative AI models with diffusion-based methods.
  + `segment\_anything`: A library for segmenting objects in images using neural networks.
  + `stability\_sdk`: A software development kit for building stable and reliable machine learning models, with an interface for GooseAI.



This code imports several libraries and modules for various image processing tasks. Here's a brief overview:

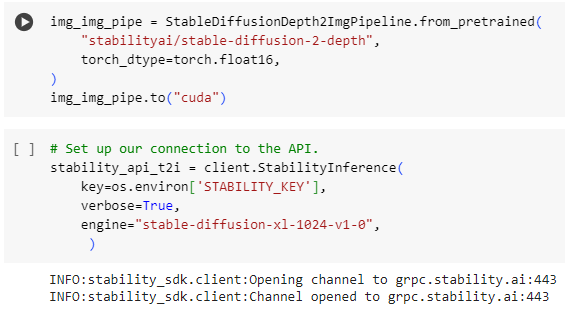
* + `gradio`: A library for creating UIs for machine learning models.
  + `cv2`: OpenCV, a library for computer vision tasks.
  + `numpy`: A fundamental package for scientific computing with Python.
  + `os`: Provides a portable way of using operating system-dependent functionality.
  + `random`: Generates random numbers.
  + `basicsr`: A collection of basic modules for super-resolution tasks.
  + `RealESRGANer`: A wrapper for the Real-ESRGAN model, which performs image super-resolution.
  + `SRVGGNetCompact`: Architecture for VGG-based models used in image super-resolution.
  + `PIL`: Python Imaging Library, a library for adding image processing capabilities to Python.
  + `google.colab`: Specific functionality related to Google Colab, an online platform for running Python code.

Main Code:



Sure, here's a simplified explanation:

* + `sam\_checkpoint = "weights/sam\_vit\_h\_4b8939.pth"`: This line loads pre-trained weights for a segmentation model based on a Vision Transformer architecture.
  + `model\_type = "vit\_h"`: Specifies the type of model being used, in this case, a Vision Transformer model.
  + `sam = sam\_model\_registry[model\_type](checkpoint=sam\_checkpoint)`: Initializes the segmentation model with the specified type and pre-trained weights.
  + `sam.to("cuda")`: Moves the segmentation model to the GPU for faster computation.
  + `predictor = SamPredictor(sam)`: Creates a predictor object for making predictions using the segmentation model.
  + `inpaint\_pipe = StableDiffusionInpaintPipeline.from\_pretrained("stabilityai/stable-diffusion-2-inpainting", torch\_dtype=torch.float32)`: Initializes a pipeline for image inpainting using diffusion-based methods.
  + `inpaint\_pipe1 = inpaint\_pipe.to("cuda")`: Moves the inpainting pipeline to the GPU for faster processing.



This code sets up two components:

1. `img\_img\_pipe`: Initializes a pipeline for converting depth maps to images using diffusion-based methods, with pre-trained weights loaded from "stabilityai/stable-diffusion-2-depth". The `torch\_dtype` parameter specifies the data type used for computation, in this case, torch.float16. Then, it moves this pipeline to the GPU for faster processing.

2. `stability\_api\_t2i`: Sets up a connection to the Stability AI API for text-to-image generation. It uses the provided API key and specifies the engine to use as "stable-diffusion-xl-1024-v1-0".



This function `text2img(prompt)` takes a text prompt as input and generates an image using the Stability AI API. Here's a breakdown of its functionality:

- It calls the `generate` method of the `stability\_api\_t2i` object to generate an image based on the provided text prompt.

- The parameters passed to the `generate` method include the prompt itself, a seed for reproducibility, the number of steps for generating the image, the scale of the configuration, width and height of the image, the number of samples to generate, and the sampler to use.

- The generated images are processed from the response and stored in the `img` variable.

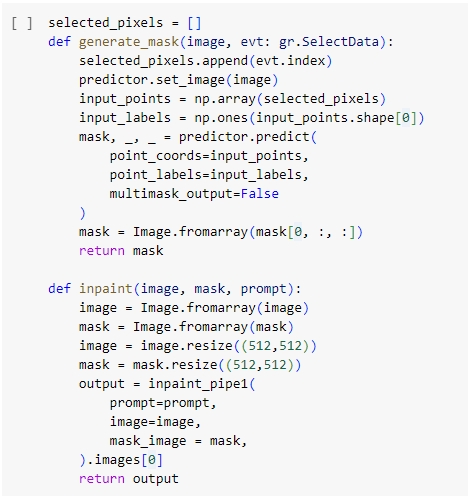
- The function returns the generated image.



This function `load\_image(image\_input, prompt)` takes an image input and a text prompt as arguments. It processes the image using the `img\_img\_pipe` pipeline, which converts depth maps to images using diffusion-based methods. Here's a breakdown of its functionality:

* + It converts the image input (provided as a NumPy array) into a PIL Image object using `Image.fromarray()`.
  + It passes the image input and the text prompt to the `img\_img\_pipe` pipeline.
  + The `negative\_prompt` parameter is set to `None`, and the `strength` parameter controls the influence of the prompt on the generated image.
  + The generated image from the pipeline is extracted from the output and returned.

This function essentially acts as a wrapper for applying a text prompt to an image generation pipeline.

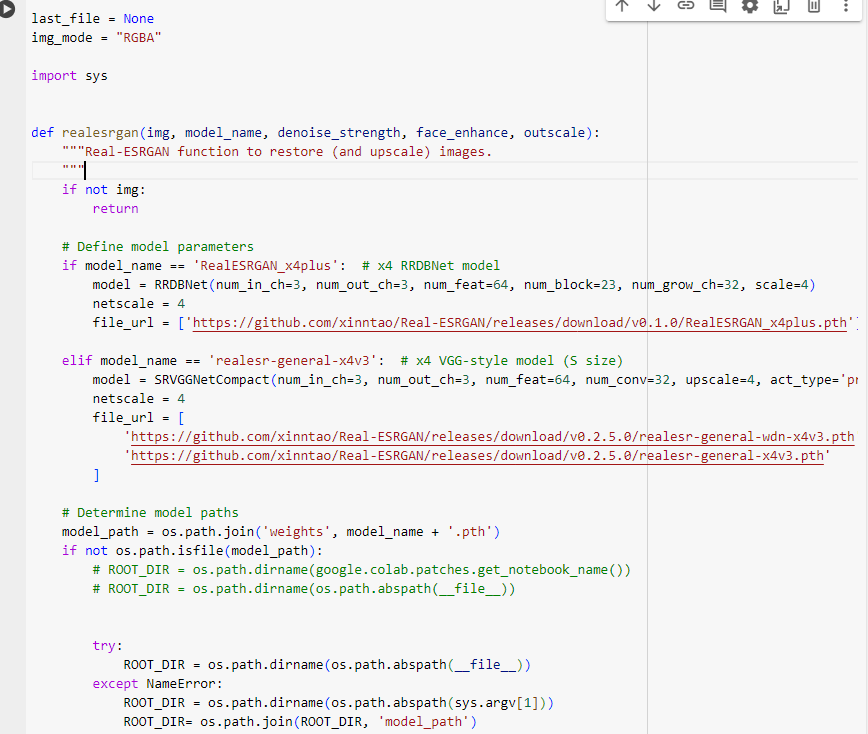


These functions are used for generating and applying masks on images and then inpainting the masked areas using the inpainting pipeline. Here's a brief explanation of each function:

1. `generate\_mask(image, evt: gr.SelectData)`: This function generates a mask based on the selected pixels on the input image. It appends the coordinates of the selected pixels to the `selected\_pixels` list, then uses the segmentation model (`predictor`) to generate a mask based on these selected pixels. The generated mask is returned as a PIL Image object.

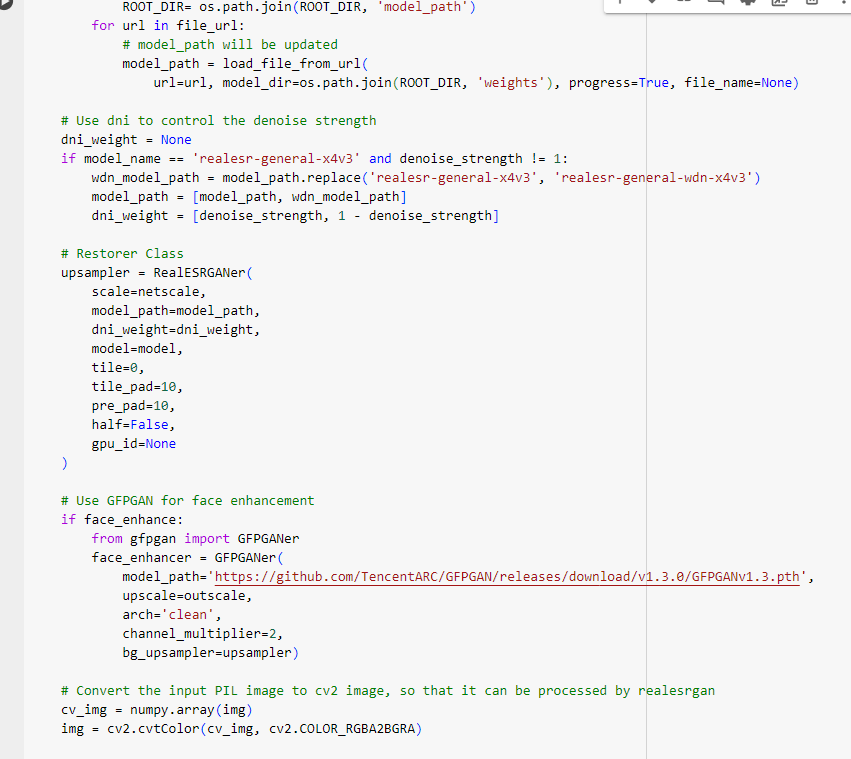
2. `inpaint(image, mask, prompt)`: This function takes an image, a mask, and a text prompt as input. It resizes the input image and mask to a fixed size (512x512). Then, it applies the inpainting pipeline (`inpaint\_pipe1`) to inpaint the image using the provided text prompt and the mask. The inpainted image is returned as a PIL Image object.

These functions together facilitate interactive inpainting of images based on user-selected regions and textual prompts.



This function `realesrgan(img, model\_name, denoise\_strength, face\_enhance, outscale)` performs image restoration and upscaling using the Real-ESRGAN model. Here's a breakdown of its functionality:

* + It first checks if the input image `img` is provided. If not, it returns.
  + Then, based on the selected `model\_name`, it initializes the corresponding Real-ESRGAN model. The available models are 'RealESRGAN\_x4plus' and 'realesr-general-x4v3'.
  + It determines the URLs for downloading the pre-trained model weights based on the selected `model\_name`.
  + It sets the `model\_path` where the downloaded model weights will be saved.
  + If the model weights are not already downloaded, it attempts to download them from the provided URLs and saves them to the `weights` directory.
  + If running in Google Colab environment, it sets the `ROOT\_DIR` to the current notebook directory. Otherwise, it sets it to the directory of the provided script file.



This portion of the code completes the setup for using the Real-ESRGAN model for image restoration and upscaling:

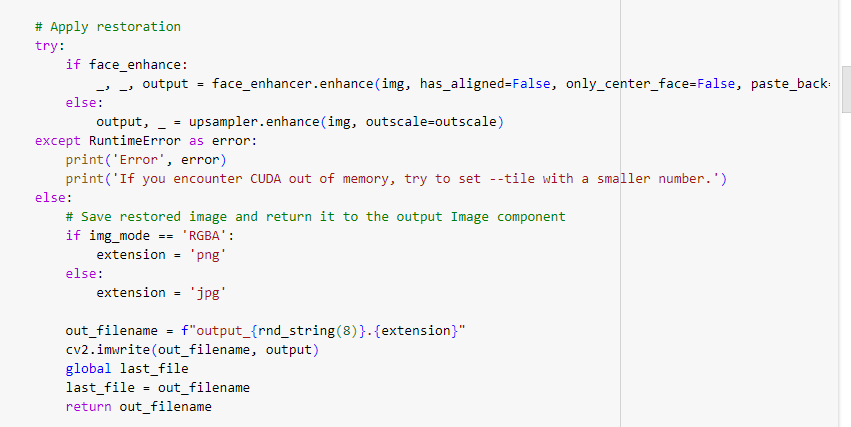
- It iterates through the list of URLs in `file\_url` and downloads the corresponding model weights using the `load\_file\_from\_url` function. The downloaded model weights are saved in the `weights` directory.

- If the selected model is 'realesr-general-x4v3' and the `denoise\_strength` parameter is not equal to 1, it modifies the model path to include both the regular and denoising models, and sets up weights for the denoising neural network (DNI).

- It initializes the `RealESRGANer` class with parameters including the scale factor, model path, DNI weights, and other configuration options.

- If `face\_enhance` is enabled, it sets up the `GFPGANer` class for face enhancement, using the pretrained GFPGAN model.

- Finally, it converts the input PIL image to a NumPy array compatible with OpenCV for further processing by Real-ESRGAN.



This code block attempts to enhance the input image using the Real-ESRGAN model or the GFPGAN model for face enhancement if `face\_enhance` is enabled. Here's what it does:

- It checks if `face\_enhance` is enabled. If so, it calls the `enhance` method of the `face\_enhancer` object to enhance the image.

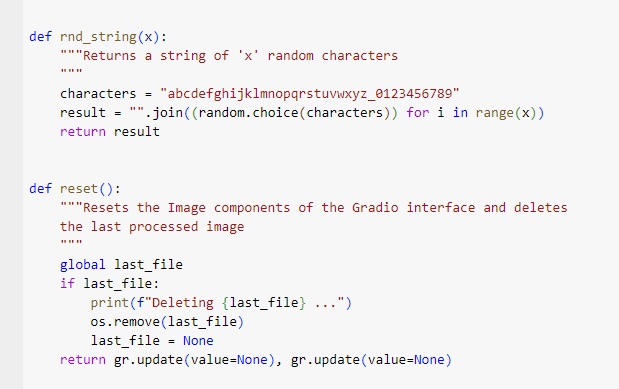
- If `face\_enhance` is not enabled, it calls the `enhance` method of the `upsampler` object to enhance the image using the Real-ESRGAN model.

- If an error occurs during processing (e.g., CUDA out of memory), it prints an error message and suggests adjusting the `--tile` parameter.

- If processing is successful, it saves the restored image to a file with a randomly generated filename and returns the filename.

Overall, this block processes the input image using either Real-ESRGAN or GFPGAN for enhancement and handles potential errors during processing.

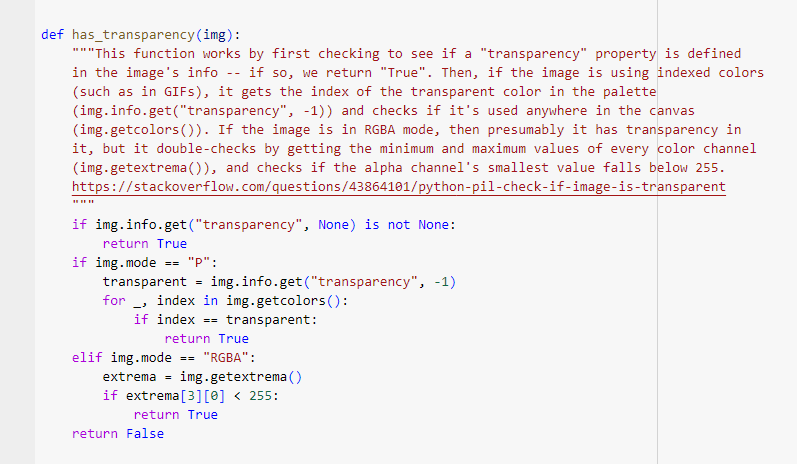
This function essentially prepares the Real-ESRGAN model for image restoration and upscaling by initializing the model, downloading the required model weights if not already downloaded, and setting up the directory for saving the model weights.



These two functions serve additional utility purposes within the Gradio interface:

1. `rnd\_string(x)`: This function generates a random string of length `x` consisting of lowercase letters, underscores, and digits. It is used, for example, to generate random filenames for saving processed images.

2. `reset()`: This function resets the image components of the Gradio interface and deletes the last processed image file. It checks if there is a `last\_file` variable storing the filename of the last processed image. If such a file exists, it deletes the file and sets `last\_file` to `None`. Finally, it updates the image components in the Gradio interface to remove the displayed images.



This function `has\_transparency(img)` checks if an image has transparency by examining its properties and color channels. Here's a breakdown of its functionality:

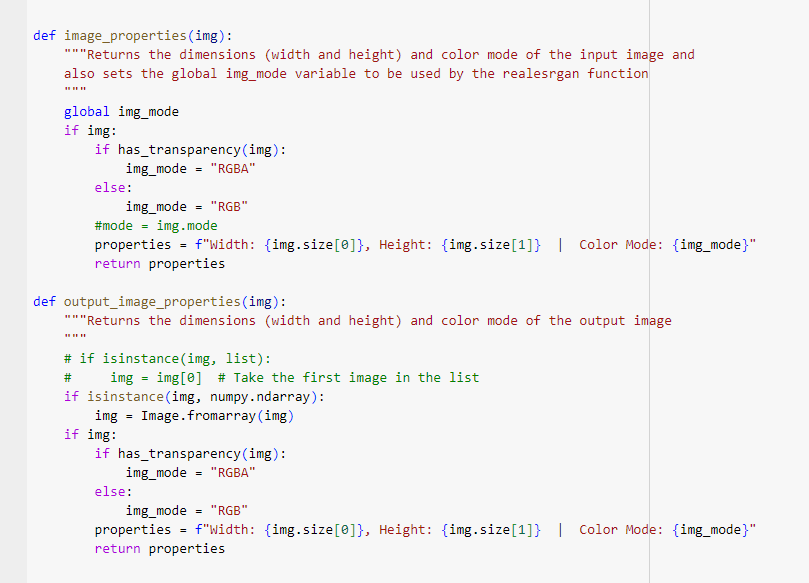
- It first checks if the image has a "transparency" property defined in its info dictionary. If so, it returns `True`.

- If the image is using indexed colors (mode "P"), it retrieves the index of the transparent color in the palette and checks if it's used anywhere in the image's canvas. If found, it returns `True`.

- If the image is in RGBA mode, it examines the minimum and maximum values of every color channel to determine if the alpha channel's smallest value falls below 255 (indicating transparency). If so, it returns `True`.

- If none of the above conditions are met, it returns `False`, indicating that the image does not have transparency.

This function provides a comprehensive check for transparency in various types of images and modes.



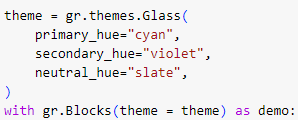
These two functions, `image\_properties(img)` and `output\_image\_properties(img)`, provide information about the properties of input and output images, respectively. Here's what each function does:

1. `image\_properties(img)`: This function takes an input image `img` as input and returns a string containing information about the image's dimensions (width and height) and color mode. It first checks if the image has transparency using the `has\_transparency` function and sets the global variable `img\_mode` accordingly. Then, it constructs and returns a string with the image properties.

2. `output\_image\_properties(img)`: This function is similar to `image\_properties`, but it is specifically designed to work with output images. It takes an image `img` as input and returns a string containing information about the output image's dimensions and color mode. It also checks for transparency in the image and sets the `img\_mode` variable accordingly.

Both functions provide concise information about the dimensions and color mode of images, which can be useful for displaying metadata in the user interface or logging purposes.

User Interface:



This code block applies a custom theme to a Gradio interface using the `Glass` theme. Here's a breakdown of the theme parameters:

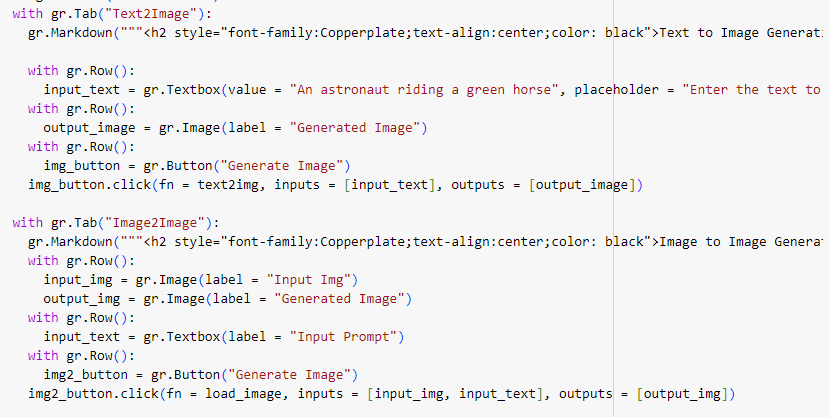
- `primary\_hue`: Sets the primary color hue to "cyan".

- `secondary\_hue`: Sets the secondary color hue to "violet".

- `neutral\_hue`: Sets the neutral color hue to "slate".

The `Glass` theme is then applied to the Gradio interface using the specified color hues, resulting in a visual style characterized by these colors.

Text 2 Image:



This code creates a Gradio interface with tab: "Text2Image"

Displays a Markdown header for "Text to Image Generation".

Provides a textbox for entering text prompts.

Displays an image component for showing the generated image.

Includes a button labeled "Generate Image" for triggering the image generation process. When clicked, it calls the text2img function with the input text as argument and updates the output image component with the generated image.

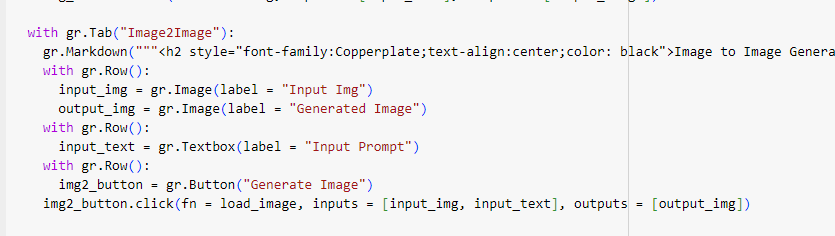


Image2Image Tab:

Displays a Markdown header for "Image to Image Generation".

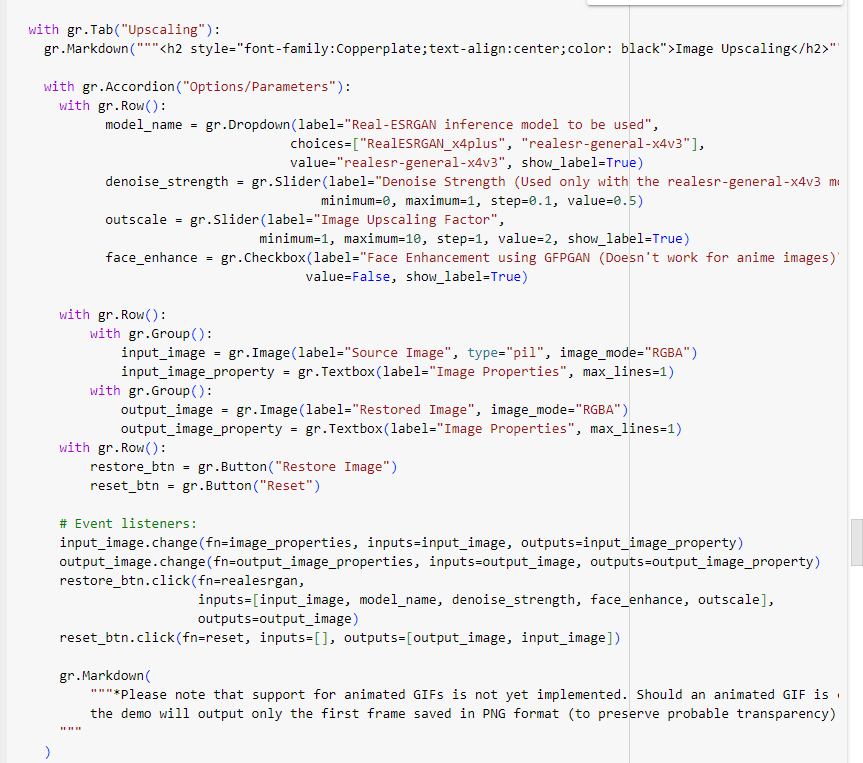
Provides an image component for displaying the input image.

Displays another image component for showing the generated image.

Includes a textbox for entering prompts related to image generation.

Includes a button labeled "Generate Image" for triggering the image generation process. When clicked, it calls the load\_image function with the input image and text prompt as arguments, and updates the output image component with the generated image.

Upscaling:



This code block creates a Gradio interface tab for image upscaling with various options and parameters. Here's a breakdown of each section:

1. \*Upscaling Tab:

- Displays a Markdown header for "Image Upscaling".

- Includes an accordion widget for organizing options and parameters.

2. Options/Parameters Accordion:

- Provides dropdowns, sliders, and checkboxes for selecting the Real-ESRGAN inference model, denoise strength (specific to the 'realesr-general-x4v3' model), image upscaling factor, and whether to enhance faces using GFPGAN.

- Displays image components for the source image and the restored image, along with textboxes showing the properties of both input and output images.

- Includes buttons labeled "Restore Image" for triggering the image restoration process and "Reset" for clearing the image components and resetting the interface.

- Attaches event listeners to the input and output image components to update their properties dynamically.

- Provides a Markdown note about the limitation regarding support for animated GIFs.

Overall, this interface allows users to select options and parameters for image upscaling, visualize input and output images, and trigger the restoration process with the click of a button.

Masking and Inpainting:



This code creates a Gradio interface tab for masking and inpainting images. Here's a breakdown of each section:

1. Inpainting Tab:

- Displays a Markdown header for "Masking and Inpainting".

2. Image Input and Output:

- Provides image components for displaying the input image, mask image, and output image.

3. Text Prompt:

- Includes a textbox for entering a prompt related to inpainting.

4. Submit Button:

- Displays a button labelled "Submit" for triggering the inpainting process.

5. Event Handlers:

- Attaches an event handler to the `input\_img1` image component, specifying the `generate\_mask` function to be called when the input image is selected. This function generates a mask based on the selected pixels on the input image.

- Attaches an event handler to the "Submit" button, specifying the `inpaint` function to be called when the button is clicked. This function performs inpainting using the input image, mask, and prompt text, and updates the output image component with the inpainted image.

Overall, this interface allows users to select an input image, generate a mask by selecting pixels, enter a prompt, and perform inpainting with the click of a button.

**Challenges and Solutions**

**1.Selecting Compatible Models:**   **Challenge:** Picking models that work well with our system was tricky.  
   **Solution:** We found suitable models from Hugging Face, a platform offering a variety of models. These models fit our system and met our needs for different image editing tasks.

**2.Integration Issues:**   **Challenge:** When putting everything together, we faced problems making the models work smoothly with each other and our hardware.  
**Solution:** We tweaked the models' settings, especially how they use CPU and GPU resources. This adjustment helped them work together seamlessly, ensuring our application runs smoothly.

**3. Utilizing RealsGAN:**  
   **Challenge:** Using RealsGAN, which boosts image quality, was tough due to its heavy demands on resources.  
   **Solution:** We optimized RealsGAN's use of our system's GPU power. This made sure it could handle the workload efficiently, making images look better without slowing down our app.

**4. Stability AI:**  
   **Challenge:** Making sure our AI-powered features are reliable and don't cause errors was a challenge.  
   **Solution:** We built safeguards into our app to catch and fix any errors. Plus, we kept a close eye on how our AI models performed, testing them thoroughly to ensure they work smoothly without any hiccups.

**Features**

**1.Text-to-Image Generation:** Utilizing advanced models from Hugging Face, our application enables users to effortlessly generate images from text descriptions. This feature allows for seamless translation of textual concepts into visual representations, opening up creative possibilities for users without the need for manual image creation.

**2.Image Transformation:** With our application, users can easily apply various styles and effects to their images, including artistic filters and colorization. This feature empowers users to enhance the aesthetic appeal of their images with just a few simple clicks.

**3. Image Restoration:** Our application incorporates advanced algorithms, such as Inpaint and AttnGAN, to restore damaged or incomplete images. By intelligently filling in missing parts while maintaining realism, this feature enables users to salvage cherished memories and preserve the integrity of their photographs.

**4. Image Upscaling:** Through the use of models like SRGAN and Real-ESRGAN, our application allows users to enhance the resolution of their images while preserving fine details. Whether enlarging a photo for printing or improving the clarity of digital images, this feature ensures that users can achieve stunning results without compromising image quality.

**Future Scope**

**1.Enhanced Model Integration:** We plan to integrate additional state-of-the-art models and algorithms to further enhance the capabilities of our application. By continually updating our model library, we aim to stay at the forefront of image editing technology and provide users with access to the latest advancements in the field.

**2.Collaborative Editing Features:** In future iterations, we envision incorporating collaborative editing features that enable users to work together on editing projects in real-time. This functionality would facilitate seamless collaboration among individuals, whether for professional collaborations or creative endeavours.

**3.Customization Options:** To cater to the diverse preferences and artistic styles of our users, we plan to introduce enhanced customization options for image editing. This may include the ability to create custom filters, adjust blending modes, and fine-tune editing parameters to achieve personalized results.

**4.Mobile Integration:** Recognizing the growing trend towards mobile computing, we aim to develop a mobile version of our application that offers the same powerful features and intuitive interface optimized for smartphone and tablet devices. This expansion would enable users to edit images on the go, providing greater flexibility and convenience.

**Conclusion**

In summary, our project makes editing images easy for everyone. We've created a simple app that lets you enhance, transform, restore, and improve the quality of your pictures without needing any special skills. By using advanced technology and overcoming challenges like picking the right models and making them work together smoothly, we've made image editing accessible to everyone.

Looking forward, we're excited to keep improving our app. We'll add more features, make it work even better on different computers, and listen to feedback from users to make sure it's always easy and fun to use.

 Overall, our project helps people make their pictures look great without any hassle. It's all about making image editing simple and enjoyable for everyone.