NANYANG TECHNOLOGICAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Diffusion Models for Intelligent Image Editing and Inpainting

Lee Yu Quan

Supervisor: Prof Zhang Hanwang

School of Computer Science and Engineering

A Final Year Project report
presented to Nanyang Technological University
in partial fulfilment of the requirements for the
degree of Bachelor of Engineering

2025/2026

Acknowledgements

Over the course of two semesters working on this final year project, I would like to express my appreciation to everyone who has encouraged me and offered their guidance, helping make this project possible.

I would like to extend my sincere gratitude to my supervisor, Prof Zhang Hanwang, for granting me the freedom to steer the direction of this project. His trust and openness allowed me to explore a wide variety of diffusion models and techniques in the field of image generation, which greatly enriched both the project and my learning experience.

Lastly, I would like to thank my examiner, Prof (placeholder), for taking the time to review and evaluate this final year project.

Lee Yu Quan
March 2026

Contents

List of Figures

List of Tables

1 Introduction

1.1 Abstract

1.2 Background and Motivation

Large language models are the most discussed aspect of generative AI today, but image generation is not far behind. According to Grand View Research, the global AI image generator market was valued at USD 349.6 million in 2023 and is projected to grow at a compound annual growth rate (CAGR) of 17.7% to reach USD 1.08 billion by 2030 (Grand View Research, 2023). Yet, there remains a lack of comprehensive software that caters to this growing demand in an accessible manner.

Traditional methods in image inpainting, such as patch-based and exemplar-based approaches, have notable limitations in generating semantically meaningful content, particularly in high-resolution or complex scenarios (Ma et al., 2023). These methods often struggle with boundary artefacts when dealing with large masked regions due to insufficient constraints, resulting in visible seams and structurally inconsistent outputs (Ma et al., 2023). Such limitations create significant accessibility barriers, as current solutions frequently require extensive technical expertise and expensive software licences, effectively restricting advanced image editing capabilities to professional users.

The introduction of deep learning techniques, particularly diffusion models (Ho et al., 2020; Song et al., 2021; Rombach et al., 2022), has led to significant improvements in image generation quality and semantic comprehension, enabling capabilities that were previously difficult or impossible to automate. A detailed review of these developments is presented in Section 3.

Despite these breakthroughs, critical gaps persist between state-of-the-art image editing models and users' practical needs. Existing solutions face four key limitations: (1) fragmented ecosystems requiring users to switch between different applications for different editing tasks, (2) high complexity barriers that make advanced editing tools inaccessible to non-expert users, (3) reliance on command-line tools, Python programming, and GPU-equipped hardware, and (4) a lack of unified platforms that integrate multiple diffusion model capabilities within a single interface.

This project, DiffusionDesk, addresses these gaps by deploying a web-based platform that integrates diffusion models for inpainting, style transfer, and image restoration within a single, user-friendly interface. Developed as a Final Year Project at Nanyang Technological University under the supervision of Prof Zhang Hanwang, the application provides three core image editing capabilities:

1. **Inpainting** – removing or replacing objects within selected regions of an image using models such as Stable Diffusion, Stable Diffusion XL, Kandinsky, and FLUX.1 Fill.

2. **Style Transfer** – applying artistic styles such as anime, oil painting, and watercolour to images using diffusion-based image-to-image translation.

3. **Restoration** – enhancing image quality through face restoration (CodeFormer, GFPGAN) and image upscaling (Real-ESRGAN).

By exposing these models through a FastAPI backend and a React-based browser frontend, DiffusionDesk aims to make diffusion-based image editing accessible to users without requiring direct interaction with the underlying models or command-line tools.

1.3 Project Objective

The objective of this project is to design and deploy a web-based image editing platform that leverages open-source diffusion models to provide intelligent inpainting, style transfer, and image restoration capabilities. The specific objectives are as follows:

1. To develop a responsive web application that integrates multiple diffusion models for image editing within a unified interface.

2. To implement an inpainting feature that enables users to selectively remove or replace objects in images using state-of-the-art diffusion models, including Stable Diffusion, Stable Diffusion XL, Kandinsky, and FLUX.1 Fill.

3. To implement a style transfer feature that allows users to apply artistic styles to images through diffusion-based image-to-image translation.

4. To implement an image restoration feature that enhances image quality through face restoration and upscaling using CodeFormer, GFPGAN, and Real-ESRGAN.

5. To design an intuitive user interface that enables non-expert users to perform advanced image editing tasks without requiring technical expertise in machine learning or programming.

6. To evaluate the system's performance through processing speed benchmarks, output quality assessments, and usability considerations.

1.4 Limitations

This project is subject to the following limitations:

1. **Open-source models only** – The application exclusively utilises open-source diffusion models available through the Hugging Face ecosystem. Proprietary or commercially licensed models are not included, which may limit the range of available capabilities compared to commercial solutions.

2. **GPU resource constraints** – Diffusion model inference is computationally intensive and requires GPU acceleration. The available GPU memory (VRAM) constrains the size and complexity of models that can be loaded simultaneously. Quantisation techniques (4-bit, 8-bit) are employed to mitigate this, but may result in slight quality degradation.

3. **Supported image formats** – The application supports JPEG, JPG, and PNG image formats only. Other formats such as TIFF, BMP, WebP, or RAW are not supported.

4. **No mobile application** – The platform is designed as a web application accessible through desktop and mobile browsers. A dedicated native mobile application is not within the project scope.

5. **No video processing** – The system processes individual images only. Video frame processing, video inpainting, or video style transfer are not supported.

6. **No 3D image manipulation** – The application is limited to 2D image editing. 3D reconstruction, 3D-aware editing, or depth-based manipulation are not included.

7. **Inference only** – The project focuses on model inference using pre-trained models. Model training, fine-tuning, or custom model development are outside the project scope.

1.5 Project Scope

The scope of this project encompasses the following:

1.5.1 In Scope

- Development of a web-based frontend using React, TypeScript, and Tailwind CSS that provides an intuitive user interface for all three editing features.
- Development of a backend API using FastAPI and PyTorch that serves diffusion model inference for inpainting, style transfer, and image restoration.
- Implementation of inpainting using Stable Diffusion Inpainting, Stable Diffusion XL Inpainting, Kandinsky Inpainting, and FLUX.1 Fill models.
- Implementation of style transfer using SDXL image-to-image generation with artistic style prompts.
- Implementation of image restoration using CodeFormer, GFPGAN (face restoration), and Real-ESRGAN (image upscaling).
- Support for JPEG, JPG, and PNG image formats.

- VRAM optimisation through model quantisation (4-bit, 8-bit) and CPU offloading to accommodate varying GPU configurations.
- Deployment and testing on cloud GPU environments such as Google Colab.

1.5.2 Out of Scope

- Native mobile application development.
- Video processing, video inpainting, or video style transfer.
- 3D image manipulation or depth-based editing.
- Model training, fine-tuning, or custom model development.
- User authentication, user account management, or multi-user collaboration features.
- Image formats other than JPEG, JPG, and PNG.

Success will be measured through processing speed benchmarks, output quality assessments, and user experience evaluation across the three core features.

2 Project Schedule

This section outlines the project timeline, work breakdown structure, and risk management plan for DiffusionDesk. The project spans two semesters of Academic Year 2025/2026, with key milestones aligned to the FYP submission deadlines.

2.1 Project Timeline

Table 1 presents the key milestones and deliverables for the project.

Table 1: Project Timeline and Milestones

Date	**Week**	**Milestone**
11 Aug 2025	Sem 1, Wk 1	FYP officially commences
1 Sep 2025	Sem 1, Wk 4	Submission of Project Plan to Supervisor
Oct 2025	Sem 1, Wk 8–10	Backend API and diffusion service implementation
Nov 2025	Sem 1, Wk 12–13	Frontend development and API integration
26 Jan 2026	Sem 2, Wk 3	Submission of Interim Report
Feb 2026	Sem 2, Wk 5–7	Feature refinement and testing on cloud GPU
23 Mar 2026	Sem 2, Wk 10	Submission of Final Report
17 Apr 2026	Sem 2, Wk 13	Submission of Amended Final Report
8–13 May 2026	–	Oral Presentation (20 min + 10 min Q&A)

2.2 Work Breakdown

This section provides a structured breakdown of the project activities, their descriptions, estimated effort, and dependencies. The work breakdown structure facilitates progress tracking and resource allocation throughout the development lifecycle.

Table 2: Work Breakdown Structure

ID	**Activity**	**Description**	**Effort (Days)**	**Depends On**
1.1	Literature Review	Review foundational papers on diffusion models (DDPM, DDIM, LDM) and existing inpainting, style transfer, and restoration techniques.	10	–

ID	Activity	Description	Effort (Days)	Depends On
1.2	Technology Evaluation	Evaluate and select appropriate frameworks, libraries, and pre-trained models from the Hugging Face ecosystem.	5	1.1
1.3	Requirements Specification	Define functional and non-functional requirements based on project objectives and supervisor feedback.	4	1.2
2.1	Backend Architecture Design	Design the FastAPI backend structure, including API endpoints, service layers, and model management strategy.	6	1.3
2.2	Inpainting Service	Implement the inpainting service supporting SD, SDXL, Kandinsky, and FLUX.1 Fill models with quantisation support.	15	2.1
2.3	Style Transfer Service	Implement the style transfer service using SDXL img2img with configurable style prompts and parameters.	10	2.1
2.4	Restoration Service	Implement face restoration (CodeFormer, GFPGAN) and image upscaling (Real-ESRGAN) services.	10	2.1
2.5	VRAM Optimisation	Implement model quantisation (4-bit, 8-bit) and CPU offloading strategies to support varying GPU configurations.	8	2.2, 2.3
3.1	Frontend UI Design	Design the user interface layout, including wireframes for the Home, Inpainting, Style Transfer, and Restoration tabs.	5	1.3
3.2	Frontend Implementation	Develop the React frontend with TypeScript and Tailwind CSS, implementing all UI components and tab navigation.	15	3.1
3.3	Canvas and Mask Drawing	Implement the interactive canvas component for users to draw inpainting masks on uploaded images.	8	3.2

ID	Activity	Description	Effort (Days)	Depends On
3.4	API Integration	Connect the frontend to the backend API, implementing image upload, processing requests, and result display.	6	2.2–2.4, 3.2
4.1	Cloud Deployment Testing	Deploy and test the backend on Google Colab with ngrok tunnelling to validate GPU inference performance.	5	3.4
4.2	Feature Testing	Conduct end-to-end testing of all features, addressing bugs and refining based on test results.	10	4.1
4.3	Performance Benchmarking	Measure and document inference times, memory usage, and output quality across different models.	5	4.2
5.1	Supervisor Meetings	Regular meetings with the supervisor to report progress, seek guidance, and align on project direction.	8	All
5.2	Interim Report Writing	Prepare and submit the interim report documenting progress, challenges, and preliminary results.	5	2.5, 3.4
5.3	Final Report Writing	Prepare the final report with comprehensive documentation of system design, implementation, and evaluation.	15	4.3, 5.2
5.4	Oral Presentation Prep	Prepare presentation slides and rehearse for the oral examination.	5	5.3

2.3 Risk Management

This section identifies potential risks that may impact the project's success and outlines mitigation strategies. Each risk is assessed based on its probability of occurrence, potential impact, and overall risk level. Proactive risk management ensures that challenges are anticipated and addressed promptly.

Table 3: Risk Management Plan

Risk	**Mitigation Strategy**	**Prob.**	**Impact**	**Level**
Insufficient GPU memory for large models	Implement model quantisation (4-bit, 8-bit) using bitsandbytes and enable CPU offloading. Prioritise smaller models when VRAM is limited.	High	High	High
Slow model inference	Use DDIM sampling with reduced steps (20–50), enable attention slicing, and implement model caching to reduce loading overhead.	Medium	High	Medium
Low-quality or artefacted outputs	Fine-tune prompt engineering, adjust guidance scale and denoising strength, and provide users with parameter controls for iterative refinement.	Medium	Medium	Medium
Diffusers library breaking changes	Pin specific library versions in requirements.txt and test compatibility before updating dependencies.	Medium	Medium	Medium
Cloud GPU constraints (Colab limits)	Design for stateless operation, allowing sessions to restart without data loss. Document alternative deployment options (NTU HPC, local GPU).	Medium	Medium	Medium
Frontend-backend integration issues	Define clear API contracts using OpenAPI, implement comprehensive error handling, and test integration incrementally.	Medium	High	Medium
Scope creep	Adhere strictly to defined scope. Evaluate new requests against timeline constraints and prioritise core features.	Medium	Medium	Medium
Unfamiliarity with diffusion architectures	Conduct thorough literature review early. Leverage tutorials, documentation, and pre-trained models to accelerate learning.	Low	High	Low
Time management challenges	Maintain detailed project schedule with milestones. Allocate buffer time and communicate proactively with supervisor.	Low	High	Low

3 Literature Review

3.1 Diffusion Models

The foundation of modern image generation lies in diffusion models, which produce images through an iterative denoising process. This subsection reviews the key developments that underpin the models used in this project.

3.1.1 Denoising Diffusion Probabilistic Models (DDPM)

Ho et al. (2020) proposed Denoising Diffusion Probabilistic Models (DDPMs), which generate images by treating the process as a series of denoising steps grounded in nonequilibrium thermodynamics. The forward process gradually adds Gaussian noise to an image over T timesteps until the image becomes pure noise. The reverse process then learns to denoise step by step, recovering a clean image from random noise. DDPMs demonstrated image quality that surpassed the then-dominant Generative Adversarial Networks (GANs), producing diverse, high-fidelity samples without the training instability commonly associated with GANs. However, the original DDPM formulation required a large number of denoising steps (typically $T = 1000$), resulting in slow sampling speeds.

3.1.2 Denoising Diffusion Implicit Models (DDIM)

Song et al. (2021) addressed the slow sampling limitation of DDPMs by proposing Denoising Diffusion Implicit Models (DDIMs). DDIMs reformulate the reverse diffusion process as a non-Markovian process, meaning that each denoising step can depend on the original noisy input rather than solely on the immediately preceding step. This reformulation allows for deterministic sampling and, crucially, enables the use of a subsequence of only 20–100 steps while maintaining comparable image quality. The result is a 10–50$\times$ speedup over DDPMs, making diffusion-based generation significantly more practical for interactive applications.

3.1.3 Latent Diffusion Models (LDM)

Rombach et al. (2022) proposed Latent Diffusion Models (LDMs), which achieved an optimal balance between generative quality and computational efficiency. Rather than performing diffusion directly in pixel space, LDMs first encode images into a lower-dimensional latent representation using a pre-trained autoencoder, then apply the diffusion process within this compressed latent space. This approach alleviates critical computational bottlenecks, substantially reducing memory and computation requirements while preserving high image quality. LDMs form the basis of the widely adopted Stable Diffusion family of models, including the inpainting and image-to-image variants used in this project.

3.1.4 Enabling Technologies for Conditional Generation

The diffusion models reviewed above provide the foundational denoising mechanism, but practical text-to-image systems require additional components for conditional generation. This subsection reviews three key enabling technologies that bridge the gap between unconditional diffusion and controllable image synthesis.

3.1.4.1 Classifier-Free Guidance Ho and Salimans (2022) introduced classifier-free guidance, a technique that enables conditional diffusion models to produce outputs strongly aligned with input conditions (e.g., text prompts) without requiring a separate classifier network. The method trains a single neural network to perform both conditional and unconditional generation by randomly dropping the conditioning signal during training. At inference time, the model's output is extrapolated away from the unconditional prediction towards the conditional prediction, controlled by a guidance scale parameter w:

$$\tilde{\epsilon}_\theta(z_t, c) = \epsilon_\theta(z_t, \emptyset) + w \cdot (\epsilon_\theta(z_t, c) - \epsilon_\theta(z_t, \emptyset)) \tag{1}$$

where $\epsilon_\theta(z_t, c)$ is the conditional prediction, $\epsilon_\theta(z_t, \emptyset)$ is the unconditional prediction, and w is the guidance scale. Higher values of w produce outputs more faithful to the conditioning but may reduce diversity and introduce artefacts. This parameter is exposed in DiffusionDesk as "guidance scale" and is fundamental to all inpainting and style transfer operations.

3.1.4.2 CLIP Text-Image Alignment Radford et al. (2021) developed CLIP (Contrastive Language-Image Pre-training), a model trained on 400 million image-text pairs to learn a joint embedding space for images and text. CLIP's text encoder transforms natural language prompts into dense vector representations that can guide image generation. In Stable Diffusion and similar models, the CLIP text encoder processes user prompts into conditioning embeddings that direct the denoising process. This architecture enables intuitive text-based control: users describe their desired output in natural language, and the model generates images aligned with that description. SDXL extends this by using dual text encoders (CLIP ViT-L and OpenCLIP ViT-G) for richer text understanding.

3.1.4.3 Diffusion Transformers (DiT) While Stable Diffusion and SDXL use U-Net architectures as their denoising backbone, Peebles and Xie (2023) demonstrated that Vision Transformers can serve as effective diffusion backbones. Diffusion Transformers (DiT) replace the convolutional U-Net with a transformer architecture, operating on sequences of latent patches. This approach scales more efficiently with model size and has led to state-of-the-art image generation quality. FLUX.1, developed by Black Forest Labs, adopts a DiT-based architecture with flow matching (a generalisation of diffusion that learns direct probability paths rather than score functions), achieving superior quality at the cost of increased computational requirements.

The architectural shift from U-Net to DiT represents a significant evolution in diffusion model design.

Having established the theoretical foundations—from DDPM's denoising mechanism, through DDIM's accelerated sampling, to LDM's latent space operation and the enabling technologies of classifier-free guidance, CLIP conditioning, and transformer backbones—the following sections examine how these principles are instantiated in practical models for specific image editing tasks.

3.2 Diffusion-Based Image Inpainting

Image inpainting is the task of filling in missing or masked regions of an image with plausible content. Traditional approaches include patch-based methods that copy similar patches from elsewhere in the image, and exemplar-based techniques that iteratively fill regions based on surrounding texture and structure. While effective for simple cases, these methods struggle with large masked regions, complex scenes, and semantically meaningful content generation—they cannot, for example, intelligently fill a masked region with a contextually appropriate object.

Deep learning approaches, particularly diffusion models, have transformed inpainting by learning rich semantic priors from large datasets. Rather than relying on local texture matching, diffusion-based inpainting models understand scene context and can generate novel content that is both visually coherent and semantically meaningful. This subsection reviews four diffusion-based inpainting models implemented in DiffusionDesk, each building upon the theoretical foundations established in Section 3.1.

3.2.1 Problem Definition

Given an input image x and a binary mask m indicating the region to be inpainted, the goal is to generate an output image $\hat{x}$ where the masked region contains plausible content consistent with the surrounding context. In diffusion-based inpainting, the model is additionally conditioned on a text prompt c that guides the generation. The inpainting process can be formulated as:

$$\hat{x} = f(x, m, c; \theta) \tag{2}$$

where f is the inpainting model with parameters θ. The model must preserve the unmasked regions exactly while generating coherent content in the masked areas.

3.2.2 Inpainting Model Architectures

3.2.2.1 Stable Diffusion Inpainting Stable Diffusion Inpainting (Rombach et al., 2022) extends the base Stable Diffusion v1.5 model for inpainting by modifying the input architecture. While the original model accepts a 4-channel latent input, the inpainting variant accepts 9

channels: 4 for the noisy latent, 4 for the encoded masked image, and 1 for the downsampled mask. This architectural modification allows the model to explicitly condition on both the masked image content and the mask geometry.

The model inherits the core DDPM/DDIM denoising mechanism and operates in the latent space as per LDM principles. Text conditioning is provided through the CLIP ViT-L/14 text encoder, and classifier-free guidance controls the fidelity to the text prompt. With approximately 860 million parameters and VRAM requirements of 5–7 GB in FP16 precision, Stable Diffusion Inpainting offers a good balance of quality and computational efficiency, making it suitable for users with limited GPU resources.

3.2.2.2 Stable Diffusion XL Inpainting SDXL Inpainting builds upon the base SDXL architecture, which introduced several improvements over Stable Diffusion v1.5: a larger U-Net backbone with approximately 2.6 billion parameters, native 1024×1024 resolution support, and dual text encoders (CLIP ViT-L and OpenCLIP ViT-G) for enhanced prompt understanding. These improvements translate to higher-quality inpainting results with better detail preservation and more accurate text-to-image alignment.

The theoretical foundations remain consistent: SDXL Inpainting uses DDPM-based denoising, supports DDIM and other accelerated schedulers, operates in latent space, and employs classifier-free guidance. However, the increased model capacity comes at a computational cost—SDXL Inpainting requires 10–12 GB VRAM in FP16 precision. To address this, DiffusionDesk implements 8-bit and 4-bit quantisation using the bitsandbytes library, reducing VRAM requirements to approximately 6 GB and 4 GB respectively, with minimal quality degradation.

3.2.2.3 Kandinsky 2.2 Inpainting Kandinsky 2.2 employs a distinct two-stage architecture inspired by unCLIP/DALL-E 2. The first stage ("prior") maps the text prompt to a CLIP image embedding, predicting what the image embedding should look like given the text. The second stage ("decoder") generates the actual image conditioned on both the text and the predicted image embedding. This architecture leverages CLIP's powerful joint text-image space to guide generation.

For inpainting, Kandinsky conditions the decoder on the masked image and mask geometry in addition to the CLIP embeddings. With approximately 2 billion parameters across both stages and VRAM requirements of 6–8 GB, Kandinsky offers comparable resource usage to Stable Diffusion while providing different aesthetic characteristics. The two-stage approach can produce results with strong text alignment due to the explicit CLIP image embedding conditioning.

3.2.2.4 FLUX.1 Fill FLUX.1 Fill, developed by Black Forest Labs, represents a significant architectural departure from the U-Net-based models. Built on the Diffusion Transformer

(DiT) architecture, FLUX.1 uses a transformer backbone operating on sequences of latent patches rather than a convolutional U-Net. Additionally, FLUX.1 employs flow matching rather than traditional score-based diffusion, learning direct probability paths between noise and data distributions.

With approximately 12 billion parameters and a T5-XXL text encoder (rather than CLIP), FLUX.1 Fill achieves state-of-the-art inpainting quality with exceptional detail, coherence, and prompt following. However, this quality comes at significant computational cost: full BF16 inference requires 22–24 GB VRAM. DiffusionDesk implements NF4 quantisation to reduce this to approximately 10 GB, enabling deployment on consumer GPUs like the NVIDIA T4 available in Google Colab.

3.2.3 Model Comparison

Table 4 summarises the key characteristics of the four inpainting models implemented in DiffusionDesk.

Table 4: Comparison of Inpainting Models

Criteria	**SD Inpainting**	**SDXL Inpainting**	**Kandinsky 2.2**	**FLUX.1 Fill**
Architecture	U-Net (LDM)	U-Net (LDM)	Prior + Decoder	DiT (Transformer)
Parameters	∼860M	∼2.6B	∼2B	∼12B
Text Encoder	CLIP ViT-L	CLIP + OpenCLIP	CLIP	T5-XXL
VRAM (FP16)	5–7 GB	10–12 GB	6–8 GB	22–24 GB
VRAM (4-bit)	N/A	∼4 GB	N/A	∼10 GB
Quality	Good	Very Good	Good	Excellent
Theory Basis	DDPM, DDIM, LDM, CFG	DDPM, DDIM, LDM, CFG	DDPM, DDIM, LDM, CLIP	Flow Matching, DiT

3.2.4 Selection Justification

DiffusionDesk includes all four inpainting models to provide users with flexibility across different use cases and hardware constraints:

- **SD Inpainting**: Entry-level option for users with limited VRAM (<8 GB). Provides good quality results with fast inference.
- **SDXL Inpainting**: Recommended default for users with mid-range GPUs (8–12 GB). Offers improved quality and prompt understanding with quantisation options for constrained environments.

- **Kandinsky 2.2**: Alternative architecture providing different aesthetic characteristics. Useful when SDXL results are unsatisfactory for specific prompts.
- **FLUX.1 Fill**: State-of-the-art quality for users with high-end GPUs or when quantised deployment is acceptable. Best choice for complex inpainting tasks requiring maximum coherence.

The diffusion architectures examined for inpainting can also be applied to style transfer. Rather than filling masked regions, style transfer leverages the same denoising process to transform an entire image's aesthetic while preserving its structure.

3.3 Diffusion-Based Style Transfer

Style transfer aims to render an image in a different artistic style while preserving its underlying content and structure. This section reviews the evolution of neural style transfer techniques and explains how diffusion models provide a powerful and flexible approach through the image-to-image (img2img) mechanism.

3.3.1 Evolution of Neural Style Transfer

Gatys et al. (2016) pioneered neural style transfer by demonstrating that convolutional neural networks (CNNs) encode both content and style information in their hierarchical feature representations. Their optimisation-based approach iteratively modifies an image to match the content features of a source image and the style features (captured via Gram matrices) of a reference style image. While producing impressive results, this method requires a slow optimisation process for each image pair.

Subsequent work developed feed-forward style transfer networks that train a single network per style, enabling real-time inference. However, these methods are limited to pre-defined styles and cannot generalise to arbitrary style descriptions. More recent approaches explored arbitrary style transfer using adaptive instance normalisation or attention mechanisms, but these often struggle with complex style semantics or structural preservation.

Diffusion models offer a compelling alternative: rather than explicitly separating content and style features, they leverage the denoising process with text conditioning to apply stylistic transformations described in natural language. This approach supports arbitrary styles without retraining and benefits from the semantic understanding encoded in large-scale text-image models.

3.3.2 Image-to-Image Mechanism

Diffusion-based style transfer operates through the image-to-image (img2img) pipeline, which differs fundamentally from text-to-image generation. Rather than starting from pure random

noise, img2img begins by adding controlled noise to the input image and then denoises with a style-describing prompt. This process can be understood as follows:

1. **Encoding**: The input image is encoded into latent space using the VAE encoder.
2. **Noise Addition**: Gaussian noise is added to the latent representation according to a specified "denoising strength" parameter $s \in [0, 1]$. Higher values add more noise, starting the denoising from a later timestep.
3. **Conditional Denoising**: The noised latent is denoised using the diffusion model, conditioned on a text prompt describing the desired style (e.g., "oil painting style", "anime illustration").
4. **Decoding**: The denoised latent is decoded back to pixel space.

The denoising strength parameter directly controls the trade-off between structure preservation and style application:

- **Low strength (0.2–0.4)**: Subtle stylisation; original composition, shapes, and details are largely preserved.
- **Medium strength (0.5–0.7)**: Moderate stylisation; structural elements remain recognisable but undergo significant aesthetic transformation.
- **High strength (0.8–1.0)**: Aggressive stylisation; the output may diverge substantially from the input, using it primarily as compositional guidance.

This mechanism directly leverages the DDPM/DDIM denoising process: the noise schedule determines at which timestep the denoising begins, and classifier-free guidance ensures the output adheres to the style prompt.

3.3.3 Approach Selection

DiffusionDesk implements style transfer using SDXL img2img for the following reasons:

- **Quality**: SDXL's larger capacity and dual text encoders produce higher-quality stylisations with better detail and prompt following compared to SD 1.5.
- **Resolution**: Native 1024×1024 support enables high-resolution style transfer without tiling artefacts.
- **Flexibility**: Text-based style conditioning supports arbitrary styles (anime, oil painting, watercolour, pencil sketch, cyberpunk, etc.) without requiring style reference images.
- **Parameter Control**: Exposing denoising strength, guidance scale, and inference steps gives users fine-grained control over the structure-style trade-off.

- **Quantisation Support**: 8-bit and 4-bit quantisation via bitsandbytes enables deployment on resource-constrained GPUs.

DiffusionDesk provides preset style prompts (anime, oil painting, watercolour, pencil sketch, digital art) while allowing users to specify custom prompts for specialised styles.

While diffusion models excel at generative tasks like inpainting and style transfer, image restoration requires specialised architectures optimised for specific degradation types. The following section examines restoration models that complement diffusion-based editing.

3.4 Image Restoration

Image restoration addresses the inverse problem of recovering high-quality images from degraded observations. Common degradations include compression artefacts, noise, blur, low resolution, and facial damage (wrinkles, scratches, low quality). Unlike the generative tasks of inpainting and style transfer, restoration aims to recover or enhance existing content rather than synthesise new content.

Notably, the restoration models reviewed in this section are *not* diffusion models. They employ distinct architectures—codebook-based transformers, generative adversarial networks (GANs), and enhanced super-resolution networks—each optimised for specific restoration tasks. DiffusionDesk includes these models to provide a comprehensive image editing pipeline that addresses both generative and restorative user needs.

3.4.1 Face Restoration

Face restoration is a specialised task that recovers high-quality facial details from degraded face images. The challenge lies in reconstructing plausible facial features (eyes, nose, mouth) while maintaining the subject's identity. DiffusionDesk implements two complementary approaches: CodeFormer and GFPGAN.

3.4.1.1 CodeFormer Zhou et al. (2022) proposed CodeFormer, a transformer-based face restoration method that leverages a learned discrete codebook of high-quality facial features. The approach consists of three components:

1. **Codebook Learning**: A vector-quantised autoencoder learns a discrete codebook capturing diverse high-quality facial features from a large face dataset.

2. **Code Prediction**: Given a degraded face image, a transformer predicts the sequence of codebook indices that best represent the underlying face.

3. **Controllable Decoding**: The decoder reconstructs the face from the predicted codes, with a fidelity parameter $w \in [0, 1]$ controlling the trade-off between restoration quality and identity preservation.

The fidelity parameter is particularly valuable: setting $w = 0$ produces maximum restoration quality (potentially altering facial features), while $w = 1$ maximally preserves the input identity (with less aggressive restoration). Users can adjust this parameter based on whether they prioritise visual quality or identity fidelity.

CodeFormer excels at restoring severely degraded faces, producing natural-looking results with realistic textures. It handles diverse degradation types robustly and provides controllable output through the fidelity parameter.

3.4.1.2 GFPGAN Wang et al. (2021a) developed GFPGAN (Generative Facial Prior GAN), which incorporates rich facial priors from a pre-trained face generation model (StyleGAN2) to assist face restoration. The key innovations include:

1. **Generative Facial Prior**: Channel-split spatial feature transforms inject pre-trained StyleGAN2 features to provide high-quality facial priors.

2. **Facial Component Dictionaries**: Pre-computed dictionaries of facial components (left eye, right eye, mouth) enable component-specific enhancement.

3. **Identity-Preserving Loss**: An identity loss term encourages the restored face to match the input identity.

GFPGAN tends to produce sharper results than CodeFormer in some cases but may occasionally alter facial features more aggressively. It performs particularly well on moderately degraded faces and old photographs.

3.4.1.3 Face Restoration Comparison Table 5 compares CodeFormer and GFPGAN across key characteristics.

Table 5: Comparison of Face Restoration Models

Criteria	**CodeFormer**	**GFPGAN**
Architecture	Transformer + Codebook	GAN + StyleGAN2 Prior
Controllability	Fidelity parameter (0–1)	Fixed
Severe Degradation	Excellent	Good
Identity Preservation	Controllable	Moderate
Output Sharpness	Natural	Sharp
VRAM Usage	2–4 GB	2–4 GB

DiffusionDesk includes both models to accommodate different user preferences and degradation scenarios. CodeFormer is recommended for severely degraded images or when identity preservation is critical, while GFPGAN is suitable for general enhancement of moderately degraded faces.

3.4.2 Image Upscaling

Image upscaling (super-resolution) increases image resolution while adding plausible high-frequency details. Classical interpolation methods (bicubic, bilinear) produce blurry results, while deep learning approaches can hallucinate realistic details.

3.4.2.1 Real-ESRGAN Wang et al. (2021b) extended ESRGAN (Enhanced Super-Resolution GAN) to handle real-world degradations. Unlike prior methods trained on synthetic bicubic downsampling, Real-ESRGAN models a comprehensive degradation pipeline including blur, noise, compression, and their combinations. This training strategy enables robust performance on diverse real-world images.

Key characteristics of Real-ESRGAN include:

- **RRDB Architecture**: Residual-in-Residual Dense Blocks provide powerful feature extraction for upscaling.
- **Real-World Degradation Model**: Training includes blur kernels, noise injection, JPEG compression, and resize operations in various orders.
- **Scale Options**: Supports $2\times$ and $4\times$ upscaling with pre-trained models.
- **Face Enhancement Integration**: A face-enhanced variant (Real-ESRGAN + GFPGAN) applies face restoration after upscaling for improved facial detail.

Real-ESRGAN produces sharp, detailed upscaled images with minimal artefacts, making it suitable for enhancing low-resolution photographs, enlarging images for printing, or improving image quality before further editing.

3.4.3 Selection Justification

DiffusionDesk includes CodeFormer, GFPGAN, and Real-ESRGAN to provide comprehensive restoration capabilities:

- **CodeFormer**: Primary face restoration model offering controllable quality-fidelity trade-off.
- **GFPGAN**: Alternative face restoration for users preferring sharper outputs or as a fallback when CodeFormer results are unsatisfactory.
- **Real-ESRGAN**: Essential for image upscaling, complementing face restoration for full-image enhancement pipelines.

These restoration models integrate seamlessly with diffusion-based editing: users can upscale low-resolution images before inpainting, or restore faces after style transfer to recover facial details.

3.5 Technology Stack

This section reviews and justifies the technology choices for developing DiffusionDesk. Each subsection follows a structured approach: exploring available options, comparing them against relevant criteria, and justifying the final selection.

3.5.1 Machine Learning Framework Selection

3.5.1.1 Options Explored Two major deep learning frameworks were considered for model inference:

- **PyTorch**: Developed by Meta AI, known for dynamic computation graphs, Pythonic API, and strong research community adoption.
- **TensorFlow**: Developed by Google, known for production deployment tools, static graph optimisation, and TensorFlow Serving.

Table 6: ML Framework Comparison

Criteria	**PyTorch**	**TensorFlow**
Diffusion Model Support	Excellent (Diffusers, native)	Limited (some ports exist)
Pre-trained Models	Extensive (Hugging Face Hub)	Moderate
Dynamic Graphs	Native	TF2 eager mode (added later)
Debugging	Straightforward (Python)	More complex (graph mode)
Research Adoption	Dominant in generative AI	Strong in production ML
Quantisation Libraries	bitsandbytes, GPTQ, AWQ	TensorFlow Lite

3.5.1.2 Comparison

3.5.1.3 Decision PyTorch was selected as the ML framework for the following reasons:

- **Diffusers Library**: The Hugging Face Diffusers library, which provides unified pipeline abstractions for all diffusion models used in this project, is built on PyTorch.
- **Model Availability**: All target models (SD, SDXL, Kandinsky, FLUX.1, CodeFormer, GFPGAN, Real-ESRGAN) have official PyTorch implementations available on Hugging Face Hub.

- **Quantisation Support**: The bitsandbytes library for 8-bit and 4-bit quantisation is PyTorch-native and essential for deploying large models on resource-constrained GPUs.
- **Research Alignment**: PyTorch dominates generative AI research, ensuring access to the latest models and techniques.

3.5.1.4 Supporting Libraries The following PyTorch ecosystem libraries are utilised:

- **Diffusers**: Unified pipeline API for diffusion models, scheduler implementations (DDIM, DPM++, Euler), and model loading utilities.
- **Transformers**: Text encoder loading (CLIP, T5) for conditioning diffusion models.
- **bitsandbytes**: 8-bit (LLM.int8) and 4-bit (NF4) quantisation for reduced VRAM usage.
- **Accelerate**: Device placement and mixed-precision inference utilities.

3.5.2 Backend Framework Selection

3.5.2.1 Options Explored Three Python web frameworks were considered for the API backend:

- **FastAPI**: Modern, async-first framework with automatic OpenAPI documentation.
- **Flask**: Lightweight, mature framework with extensive ecosystem.
- **Django**: Full-featured framework with ORM, admin interface, and batteries-included philosophy.

Table 7: Backend Framework Comparison

Criteria	**FastAPI**	**Flask**	**Django**
Async Support	Native (ASGI)	Extension (async views)	Limited (ASGI adapter)
Type Hints	Native (Pydantic)	Manual	Manual
Auto Documentation	OpenAPI + Swagger UI	Extension required	Extension required
Performance	High (async I/O)	Moderate	Moderate
Learning Curve	Low	Low	Moderate
ML Ecosystem Fit	Excellent	Good	Moderate (ORM overhead)

3.5.2.2 Comparison

3.5.2.3 Decision

FastAPI was selected for the following reasons:

- **Async Support**: Native async/await enables efficient handling of long-running inference requests without blocking other clients.
- **Pydantic Integration**: Type-validated request/response models ensure robust API contracts and clear documentation.
- **Automatic Documentation**: Built-in Swagger UI and OpenAPI specification generation facilitates frontend integration and testing.
- **Performance**: ASGI-based architecture provides high throughput suitable for inference workloads.
- **Simplicity**: No unnecessary features (ORM, admin) that add complexity without benefit for an ML inference API.

3.5.3 Frontend Framework Selection

3.5.3.1 Options Explored

Three major frontend frameworks were considered:

- **React**: Component-based library by Meta with extensive ecosystem.
- **Vue**: Progressive framework with gentle learning curve.
- **Angular**: Full-featured framework by Google with strong typing.

Table 8: Frontend Framework Comparison

Criteria	**React**	**Vue**	**Angular**
TypeScript Support	Excellent	Good	Native
Component Model	Functional + Hooks	Options/Composition API	Class-based
Ecosystem Size	Largest	Large	Large
Learning Curve	Moderate	Low	Steep
Canvas Libraries	Extensive (Fabric.js, Konva)	Good	Limited
Build Tooling	Vite, Next.js, CRA	Vite, Nuxt	Angular CLI

3.5.3.2 Comparison

3.5.3.3 Decision

React with TypeScript was selected for the following reasons:

- **Ecosystem**: The largest ecosystem provides libraries for every need, including canvas manipulation (critical for mask drawing in inpainting).
- **TypeScript Integration**: Mature TypeScript support enables type-safe development with excellent IDE support.
- **Hooks**: Functional components with hooks provide clean state management without class complexity.
- **Community**: Extensive documentation, tutorials, and community support accelerate development.

3.5.3.4 Supporting Tools

- **Vite**: Fast build tool with hot module replacement, significantly faster than Create React App.
- **Tailwind CSS**: Utility-first CSS framework enabling rapid UI development without writing custom CSS.
- **Canvas API**: Native HTML5 canvas for mask drawing, integrated via React refs.

3.5.4 Deployment Strategy

3.5.4.1 Options Explored

Three deployment environments were evaluated:

- **Google Colab**: Free cloud Jupyter environment with T4 GPU access.
- **NTU HPC Cluster**: University-provided high-performance computing resources.
- **Cloud VM**: Commercial cloud instances (AWS, GCP, Azure) with GPU support.

Table 9: Deployment Environment Comparison

Criteria	**Google Colab**	**NTU HPC**	**Cloud VM**
Cost	Free (T4)	Free (for students)	$1–3/hour (GPU)
GPU Availability	T4 (15GB)	V100/A100	Various
Session Limits	12 hours, idle timeout	Queue-based	Persistent
Public Access	ngrok tunnelling	VPN required	Direct
Setup Complexity	Low (notebook)	Moderate (job scripts)	High (infrastructure)

3.5.4.2 Comparison

3.5.4.3 Decision A hybrid deployment strategy was adopted:

- **Development/Demo**: Google Colab with ngrok tunnelling for rapid prototyping and demonstration. The T4 GPU (15 GB VRAM) supports all models with quantisation.
- **Extended Testing**: NTU HPC for longer-running experiments requiring V100/A100 GPUs without session limits.
- **Frontend Hosting**: Static frontend deployed separately (e.g., Vercel, Netlify) to avoid session timeout issues.

3.5.4.4 Quantisation Strategy To enable deployment on memory-constrained environments like Colab's T4 GPU, DiffusionDesk implements dynamic quantisation:

- **8-bit (LLM.int8)**: Reduces VRAM by $\sim$50% with minimal quality impact. Suitable for SDXL on 8–12 GB GPUs.
- **4-bit (NF4)**: Reduces VRAM by $\sim$75% with some quality trade-off. Enables FLUX.1 Fill on T4 GPU.
- **CPU Offloading**: Sequential layer offloading to CPU RAM when GPU VRAM is exhausted, trading speed for memory.

Users can select quantisation level per model based on their hardware constraints and quality requirements.

4 Software Requirements

This section presents the software requirements for DiffusionDesk, a web-based image editing application powered by diffusion models. The requirements were gathered through a combination of literature review of existing image editing tools, analysis of diffusion model capabilities, and consideration of the target deployment environment (Google Colab with NVIDIA T4 GPU). The requirements are organised into use cases, functional requirements, and non-functional requirements.

4.1 Use Case Diagram

Figure 1 presents the use case diagram for DiffusionDesk, illustrating the interactions between the User actor and the system's core functionalities. The system boundary encompasses ten use cases spanning the three main feature areas: Inpainting, Style Transfer, and Restoration.

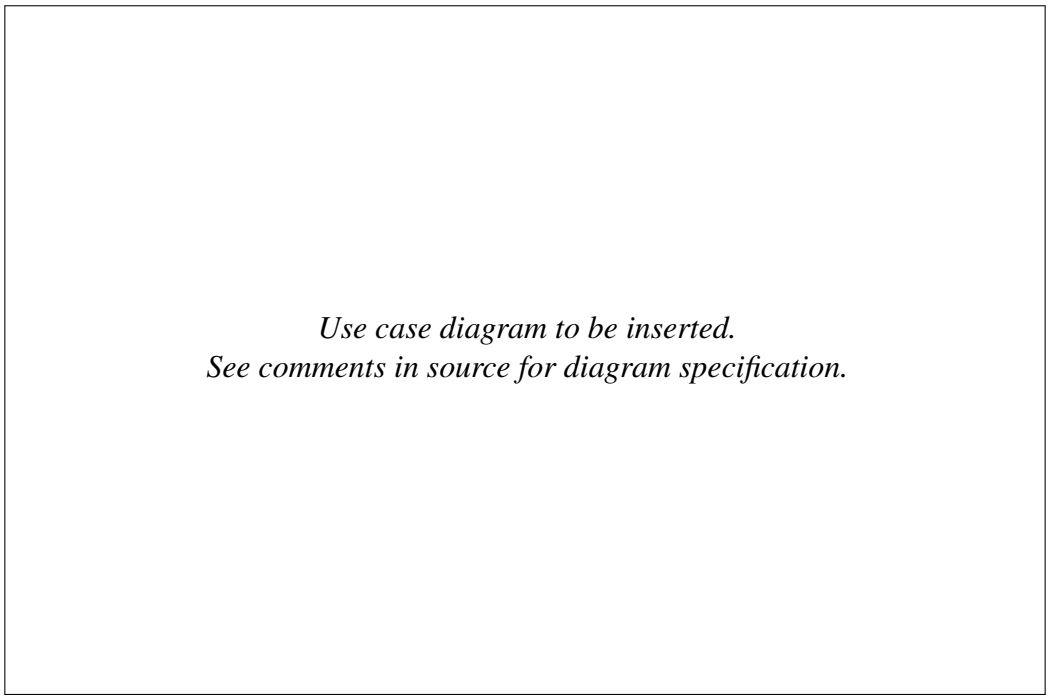

Figure 1: Use Case Diagram for DiffusionDesk

The User actor interacts directly with three primary use cases: UC-2 (Perform Inpainting), UC-4 (Apply Style Transfer), and UC-6 (Restore Image). The remaining use cases are reached indirectly through include and extend relationships. UC-10 (Load Diffusion Model) is associated with the System actor, as it is triggered automatically when a model is needed for inference. Key relationships include:

- **Include relationships**: UC-2, UC-4, and UC-6 all include UC-1 (Upload Image), as an image must be uploaded before any editing operation. UC-2 also includes UC-3 (Draw

Inpainting Mask), as a mask is required for inpainting. All editing operations include UC-10 (Load Diffusion Model).

- **Extend relationships**: UC-7 (Select Model) and UC-8 (Configure Generation Parameters) extend the editing use cases, as users may optionally change the model or adjust parameters. UC-5 (Select Style Preset) extends UC-4 (Apply Style Transfer). UC-9 (Download Result Image) extends all three editing use cases, as the user may optionally download the generated result.

4.1.1 Use Case Descriptions

The following tables provide detailed descriptions for each use case identified in the use case diagram. Each table follows a standardised format documenting the use case metadata, flow of events, and related information.

UC-1: Upload Image

Use Case ID	UC-1
Use Case Name	Upload Image
Created By	Lee Yu Quan
Date Created	8 February 2026
Actor	User
Description	The user uploads an image from their local device to the application for subsequent editing operations (inpainting, style transfer, or restoration).
Preconditions	1. The user has navigated to one of the editing tabs (Inpainting, Style Transfer, or Restoration). 2. The user has an image file in a supported format (JPEG, PNG, WebP).
Postconditions	1. The uploaded image is displayed as a preview in the editing canvas. 2. The image is ready for subsequent editing operations.
Priority	High
Frequency of Use	Every editing session

Flow of Events	1. The user clicks the upload area or drag-and-drop zone. 2. The system opens a file selection dialog. 3. The user selects an image file. 4. The system validates the file format and size. 5. The system displays the image preview in the canvas.
Alternative Flows	3a. The user drags and drops an image file onto the upload area instead of using the file dialog.
Exceptions	4a. The file format is not supported: the system displays an error message listing supported formats. 4b. The file exceeds the maximum size limit: the system displays an error with the size constraint.
Includes	–
Special Requirements	Image files must not exceed 10 MB. Supported formats: JPEG, PNG, WebP.
Assumptions	The user has a compatible web browser with JavaScript enabled.
Notes and Issues	Large images may be resized on the backend to fit within model input constraints.

UC-2: Perform Inpainting

Use Case ID	UC-2
Use Case Name	Perform Inpainting
Created By	Lee Yu Quan
Date Created	8 February 2026
Actor	User
Description	The user fills in or replaces a masked region of an uploaded image using a diffusion-based inpainting model. The user provides a text prompt describing the desired content for the masked area.

Preconditions	1. An image has been uploaded (UC-1). 2. A mask has been drawn over the region to inpaint (UC-3). 3. The backend server is running and a GPU is available.
Postconditions	1. The inpainted image is displayed alongside the original for comparison. 2. The result image is available for download (UC-9).
Priority	High
Frequency of Use	Frequent
Flow of Events	1. The user navigates to the Inpainting tab. 2. The user uploads an image (UC-1). 3. The user draws a mask over the region to edit (UC-3). 4. The user enters a text prompt describing the desired content. 5. The user optionally selects a model (UC-7) and configures parameters (UC-8). 6. The user clicks the “Generate” button. 7. The system sends the image, mask, prompt, and parameters to the backend. 8. The system loads the selected model if not already cached (UC-10). 9. The system performs inpainting inference and returns the result. 10. The system displays the inpainted image.
Alternative Flows	4a. The user enters a negative prompt to specify undesired content. 10a. The user is unsatisfied and regenerates with different parameters.
Exceptions	8a. The model fails to load due to insufficient GPU VRAM: the system suggests a lower quantisation level or smaller model. 9a. Inference fails or times out: the system displays an error message and allows the user to retry.
Includes	UC-1 (Upload Image), UC-3 (Draw Inpainting Mask), UC-10 (Load Diffusion Model)

Special Requirements	Requires an active GPU backend. Inference time varies by model (10–120 seconds).
Assumptions	The backend has sufficient VRAM for the selected model and quantisation level.
Notes and Issues	FLUX.1 Fill models require NF4 quantisation to run on T4 GPUs.

UC-3: Draw Inpainting Mask

Use Case ID	UC-3
Use Case Name	Draw Inpainting Mask
Created By	Lee Yu Quan
Date Created	8 February 2026
Actor	User
Description	The user draws a binary mask on the uploaded image to indicate the region that should be inpainted. White pixels in the mask denote the area to be filled.
Preconditions	1. An image has been uploaded and displayed in the Inpainting tab.
Postconditions	1. A mask overlay is visible on the image canvas. 2. The mask data is ready to be sent with the inpainting request.
Priority	High
Frequency of Use	Every inpainting operation
Flow of Events	1. The user selects the brush tool on the canvas. 2. The user adjusts the brush size using the slider control. 3. The user draws over the region to be inpainted by clicking and dragging on the canvas. 4. The system renders the mask overlay in a semi-transparent colour. 5. The user reviews the mask coverage.

Alternative Flows	3a. The user uses the eraser tool to remove parts of the mask. 3b. The user clicks “Clear Mask” to reset the entire mask.
Exceptions	3a. No image is loaded: the canvas tools are disabled.
Includes	–
Special Requirements	The canvas must support touch input for tablet/touchscreen devices.
Assumptions	The user can use a mouse or touchscreen to draw on the canvas.
Notes and Issues	The mask is internally represented as a black-and-white image matching the uploaded image dimensions.

UC-4: Apply Style Transfer

Use Case ID	UC-4
Use Case Name	Apply Style Transfer
Created By	Lee Yu Quan
Date Created	8 February 2026
Actor	User
Description	The user applies an artistic style (e.g., anime, oil painting, watercolour, pixel art) to an uploaded image using a diffusion-based image-to-image pipeline.
Preconditions	1. An image has been uploaded (UC-1). 2. The backend server is running and a GPU is available.
Postconditions	1. The stylised image is displayed alongside the original. 2. The result image is available for download (UC-9).
Priority	High
Frequency of Use	Frequent

Flow of Events	1. The user navigates to the Style Transfer tab. 2. The user uploads an image (UC-1). 3. The user selects a style preset (UC-5) or enters a custom style prompt. 4. The user optionally selects a model (UC-7) and adjusts the strength parameter to control style intensity. 5. The user optionally configures additional parameters (UC-8). 6. The user clicks the "Generate" button. 7. The system sends the image, prompt, and parameters to the backend. 8. The system loads the model if not already cached (UC-10). 9. The system performs image-to-image inference and returns the result. 10. The system displays the stylised image.
Alternative Flows	3a. The user enters a custom prompt instead of selecting a preset. 10a. The user adjusts the strength and regenerates to fine-tune the result.
Exceptions	8a. Model fails to load: the system suggests a lower quantisation level. 9a. Inference fails: the system displays an error message.
Includes	UC-1 (Upload Image), UC-10 (Load Diffusion Model)
Special Requirements	Requires an active GPU backend.
Assumptions	The backend has sufficient VRAM for the SDXL img2img model.
Notes and Issues	Higher strength values produce more stylised results but may lose original content details. A strength of 0.5–0.7 is recommended for most styles.

UC-5: Select Style Preset

Use Case ID	UC-5
Use Case Name	Select Style Preset

Created By	Lee Yu Quan
Date Created	8 February 2026
Actor	User
Description	The user selects a predefined artistic style from a list of presets. Each preset maps to a curated prompt optimised for that style.
Preconditions	1. The user is on the Style Transfer tab.
Postconditions	1. The selected style preset is applied to the prompt field. 2. The style is ready to be used in the next generation.
Priority	Medium
Frequency of Use	Frequent
Flow of Events	1. The user views the list of available style presets (e.g., Anime, Oil Painting, Watercolour, Pixel Art, Cinematic, Pencil Sketch). 2. The user clicks on a style preset. 3. The system populates the prompt field with the preset's curated prompt.
Alternative Flows	3a. The user modifies the auto-populated prompt to customise the style further.
Exceptions	–
Includes	–
Special Requirements	–
Assumptions	The list of presets is fetched from the backend API.
Notes and Issues	Presets can be extended by adding new entries to the backend configuration.

UC-6: Restore Image

Use Case ID	UC-6
Use Case Name	Restore Image
Created By	Lee Yu Quan

Date Created	8 February 2026
Actor	User
Description	The user restores or enhances a degraded image using one of the available restoration models: CodeFormer and GFP-GAN for face restoration, or Real-ESRGAN for general image upscaling and enhancement.
Preconditions	1. An image has been uploaded (UC-1). 2. The backend server is running and a GPU is available.
Postconditions	1. The restored image is displayed alongside the original. 2. The result image is available for download (UC-9).
Priority	High
Frequency of Use	Frequent
Flow of Events	1. The user navigates to the Restoration tab. 2. The user uploads an image (UC-1). 3. The user optionally selects a restoration model (UC-7) (e.g., CodeFormer, GFPGAN, or Real-ESRGAN). 4. The user optionally adjusts model-specific parameters (UC-8) (e.g., fidelity weight for CodeFormer, upscale factor for Real-ESRGAN). 5. The user clicks the “Restore” button. 6. The system sends the image and parameters to the backend. 7. The system loads the selected model (UC-10). 8. The system performs restoration inference and returns the result. 9. The system displays the restored image.
Alternative Flows	3a. The user tries a different restoration model on the same image.
Exceptions	3a. Face restoration models (CodeFormer, GFPGAN) produce suboptimal results on images without detectable faces: the system returns the image with minimal changes. 8a. Inference fails: the system displays an error message.
Includes	UC-1 (Upload Image), UC-10 (Load Diffusion Model)

Special Requirements	Requires an active GPU backend. Face restoration models work best on images containing human faces.
Assumptions	The uploaded image contains content suitable for the selected restoration model.
Notes and Issues	Real-ESRGAN can upscale images by $2\times$ or $4\times$, which increases output resolution and file size.

UC-7: Select Model

Use Case ID	UC-7
Use Case Name	Select Model
Created By	Lee Yu Quan
Date Created	8 February 2026
Actor	User
Description	The user selects a specific AI model from the list of available models for the current editing operation. Different models offer varying trade-offs between quality, speed, and VRAM requirements.
Preconditions	1. The user is on an editing tab (Inpainting, Style Transfer, or Restoration).
Postconditions	1. The selected model is stored and will be used for the next generation request.
Priority	Medium
Frequency of Use	Occasional
Flow of Events	1. The user views the model selection dropdown. 2. The system displays available models with their names. 3. The user selects a model from the dropdown. 4. The system updates the selected model for subsequent requests.
Alternative Flows	1a. The user keeps the default model selection.
Exceptions	–

Includes	–
Special Requirements	The model list should be fetched dynamically from the backend API.
Assumptions	The backend returns a list of models appropriate for the current task.
Notes and Issues	Available inpainting models include SD Inpainting, SDXL Inpainting, Kandinsky, and FLUX.1 Fill. Style transfer uses SDXL img2img. Restoration models include CodeFormer, GFPGAN, and Real-ESRGAN.

UC-8: Configure Generation Parameters

Use Case ID	UC-8
Use Case Name	Configure Generation Parameters
Created By	Lee Yu Quan
Date Created	8 February 2026
Actor	User
Description	The user adjusts generation parameters that control the behaviour and quality of the AI inference, such as guidance scale, number of inference steps, strength, and random seed.
Preconditions	1. The user is on an editing tab with parameter controls visible.
Postconditions	1. The configured parameters are stored and will be sent with the next generation request.
Priority	Medium
Frequency of Use	Occasional

Flow of Events	1. The user expands the advanced parameters section. 2. The user adjusts one or more parameters using sliders or input fields: • Guidance scale (1.0–20.0) • Number of inference steps (1–100) • Strength (0.0–1.0, for img2img/style transfer) • Random seed (integer, or -1 for random) • Quantisation level (none, 8-bit, 4-bit) 3. The system validates the parameter values against allowed ranges. 4. The system updates the stored parameters.
Alternative Flows	1a. The user does not expand the advanced section and uses default values.
Exceptions	3a. A parameter value is out of range: the system clamps it to the nearest valid value.
Includes	–
Special Requirements	Parameter controls must provide sensible default values.
Assumptions	The user understands the effect of advanced parameters or is satisfied with defaults.
Notes and Issues	Increasing inference steps improves quality but increases generation time. Guidance scale controls prompt adherence.

UC-9: Download Result Image

Use Case ID	UC-9
Use Case Name	Download Result Image
Created By	Lee Yu Quan
Date Created	8 February 2026
Actor	User
Description	The user downloads the generated or restored image to their local device.

Preconditions	1. A result image has been generated by one of the editing operations.
Postconditions	1. The result image is saved to the user's local device.
Priority	Medium
Frequency of Use	After each successful generation
Flow of Events	1. The user clicks the "Download" button below the result image. 2. The system triggers a file download in the browser. 3. The image is saved to the user's default download directory.
Alternative Flows	1a. The user right-clicks the image and selects "Save Image As" from the browser context menu.
Exceptions	–
Includes	–
Special Requirements	The downloaded image should be in PNG format to preserve quality.
Assumptions	The user's browser supports programmatic file downloads.
Notes and Issues	The filename includes a timestamp and operation type for easy identification.

UC-10: Load Diffusion Model

Use Case ID	UC-10
Use Case Name	Load Diffusion Model
Created By	Lee Yu Quan
Date Created	8 February 2026
Actor	System
Description	The system loads the requested diffusion model into GPU memory, applying quantisation if specified. Previously loaded models are cached to avoid redundant loading.

Preconditions	1. An editing operation has been requested (UC-2, UC-4, or UC-6). 2. The requested model is not already cached in GPU memory.
Postconditions	1. The model is loaded into GPU memory and ready for inference. 2. The model is cached for subsequent requests.
Priority	High
Frequency of Use	On first use of each model per session
Flow of Events	1. The system receives a generation request specifying a model. 2. The system checks if the model is already cached in memory. 3. If not cached, the system downloads the model weights from Hugging Face Hub (or loads from local cache). 4. The system applies the requested quantisation level (none, 8-bit, or 4-bit). 5. The system loads the model onto the GPU. 6. The system caches the loaded model for future requests. 7. The system returns control to the calling use case for inference.
Alternative Flows	2a. The model is already cached: skip to step 7. 4a. If the model requires CPU offloading, the system enables sequential offloading.
Exceptions	3a. Model weights are unavailable (network error or missing from Hub): the system returns an error to the user. 5a. Insufficient GPU VRAM: the system clears cached models and retries, or returns an error suggesting a lower quantisation level.
Includes	–
Special Requirements	Model loading can take 30–120 seconds depending on model size and network speed. A progress indicator should be shown to the user.

Assumptions	The backend has internet access to download model weights on first use.
Notes and Issues	Switching between large models (e.g., SDXL to FLUX.1) may require clearing the previous model from VRAM first.

4.2 Functional and Non-Functional Requirements

The functional and non-functional requirements for DiffusionDesk were derived from the use case analysis above, supplemented by best practices in web application development and the specific constraints of deploying diffusion models on consumer-grade GPUs. Functional requirements define the system's behaviour, while non-functional requirements specify quality attributes and constraints.

4.2.1 Functional Requirements

The functional requirements are organised by use case, with each requirement identified by a unique code in the format **FR-XX**.

Upload Image

FR-1. The system shall accept image uploads in JPEG, PNG, and WebP formats.

FR-2. The system shall reject image files exceeding 10 MB and display an appropriate error message.

FR-3. The system shall display a preview of the uploaded image in the editing canvas.

FR-4. The system shall support drag-and-drop image upload in addition to the file selection dialog.

FR-5. The system shall resize images exceeding the model's maximum input resolution while preserving the aspect ratio.

Perform Inpainting

FR-6. The system shall provide a selection of inpainting models (SD Inpainting, SDXL Inpainting, Kandinsky Inpainting, FLUX.1 Fill).

FR-7. The system shall accept a text prompt describing the desired content for the masked region.

FR-8. The system shall accept an optional negative prompt to specify undesired content.

FR-9. The system shall send the image, mask, prompt, negative prompt, and generation parameters to the backend API.

FR-10. The system shall display the inpainted result image alongside the original for visual comparison.

Draw Inpainting Mask

FR-11. The system shall provide a canvas-based brush tool for drawing masks over the uploaded image.

FR-12. The system shall allow the user to adjust the brush size via a slider control.

FR-13. The system shall render the mask as a semi-transparent overlay on the image.

FR-14. The system shall provide a "Clear Mask" button to reset the entire mask.

FR-15. The system shall generate a binary mask image (black background, white mask) matching the uploaded image dimensions.

Apply Style Transfer

FR-16. The system shall support style transfer via diffusion-based image-to-image generation.

FR-17. The system shall provide predefined style presets (e.g., Anime, Oil Painting, Watercolour, Pixel Art, Cinematic, Pencil Sketch).

FR-18. The system shall allow the user to enter a custom style prompt.

FR-19. The system shall provide a strength parameter (0.0–1.0) to control the intensity of the style application.

FR-20. The system shall display the stylised result image alongside the original for comparison.

Restore Image

FR-21. The system shall support face restoration using CodeFormer and GFPGAN models.

FR-22. The system shall support general image upscaling and enhancement using Real-ESRGAN.

FR-23. The system shall provide model-specific parameters (e.g., fidelity weight for CodeFormer, upscale factor for Real-ESRGAN).

FR-24. The system shall display the restored result image alongside the original for comparison.

Select Model

FR-25. The system shall fetch the list of available models from the backend API dynamically.

FR-26. The system shall display the available models in a dropdown selection menu.

FR-27. The system shall set a sensible default model for each editing tab.

Configure Generation Parameters

FR-28. The system shall provide controls for adjusting guidance scale (1.0–20.0).

FR-29. The system shall provide controls for adjusting the number of inference steps (1–100).

FR-30. The system shall provide controls for adjusting strength (0.0–1.0) for image-to-image operations.

FR-31. The system shall provide a seed input field for reproducible generation (-1 for random).

FR-32. The system shall provide quantisation level selection (none, 8-bit, 4-bit) where applicable.

FR-33. The system shall validate parameter values and clamp them to valid ranges.

FR-34. The system shall provide sensible default values for all parameters.

Download Result Image

FR-35. The system shall provide a download button for each generated result image.

FR-36. The system shall download the image in PNG format with a descriptive filename.

Load Diffusion Model

FR-37. The system shall load diffusion models on demand when a generation request is received.

FR-38. The system shall cache loaded models in GPU memory to avoid redundant loading.

FR-39. The system shall apply quantisation (8-bit or 4-bit) when specified by the user.

FR-40. The system shall clear previously cached models when GPU VRAM is insufficient for a new model.

FR-41. The system shall display a loading indicator while a model is being loaded.

4.2.2 Non-Functional Requirements

The non-functional requirements define the quality attributes and constraints of DiffusionDesk. Each requirement is identified by a unique code in the format **NFR-XX**.

Usability

NFR-1. The user interface shall be intuitive and require no prior experience with diffusion models to use basic features.

NFR-2. The system shall provide clear labels, tooltips, and instructions for all controls.

NFR-3. The system shall maintain a consistent visual design across all tabs using Tailwind CSS.

NFR-4. The system shall provide real-time feedback during image generation, including progress indicators and status messages.

NFR-5. The system shall organise features into clearly labelled tabs (Inpainting, Style Transfer, Restoration) for easy navigation.

NFR-6. The system shall display error messages in a user-friendly format, avoiding raw technical error traces.

Performance

NFR-7. The frontend shall load within 3 seconds on a standard broadband connection.

NFR-8. Image upload and preview rendering shall complete within 2 seconds for files under 5 MB.

NFR-9. Inpainting and style transfer inference shall complete within 120 seconds on a T4 GPU with 4-bit quantisation.

NFR-10. Image restoration shall complete within 30 seconds on a T4 GPU.

NFR-11. The system shall display generation progress to the user so that long-running operations do not appear unresponsive.

Reliability

NFR-12. The system shall handle backend errors gracefully and display meaningful error messages to the user.

NFR-13. The system shall recover from GPU out-of-memory errors by suggesting alternative models or quantisation levels.

NFR-14. The frontend shall remain functional even if the backend is temporarily unavailable, allowing image upload and parameter configuration.

NFR-15. The system shall validate all user inputs on both the frontend and backend to prevent invalid requests.

Maintainability

NFR-16. The system shall follow a modular architecture with clear separation between frontend, backend API, and ML inference services.

NFR-17. The backend shall use a router-service pattern with low coupling between API endpoints and model inference logic.

NFR-18. The frontend shall use reusable React components with well-defined props interfaces.

NFR-19. New models shall be addable to the system by extending the backend service configuration without modifying existing code.

Compatibility

NFR-20. The frontend shall be compatible with the latest versions of Google Chrome, Mozilla Firefox, Microsoft Edge, and Apple Safari.

NFR-21. The frontend shall be responsive and functional on screen widths from 1024 pixels and above.

NFR-22. The backend shall run on NVIDIA GPUs with CUDA support (compute capability 7.0 or higher).

NFR-23. The system shall support deployment on Google Colab with T4 GPU using ngrok for backend exposure.

5 Planning and Design

This section presents the planning and design decisions made during the development of DiffusionDesk. It covers the software development methodology adopted, the system architecture, and the user interface wireframe designs that guided the implementation.

5.1 Project Development Methodology

The software development methodology adopted for DiffusionDesk is the **Iterative and Incremental Development** methodology. This approach was chosen because the project involves integrating multiple AI models with varying hardware requirements, where each feature (inpainting, style transfer, restoration) can be developed and tested as an independent increment. The iterative nature allows for continuous refinement based on testing feedback, which is particularly important when working with diffusion models where output quality depends heavily on parameter tuning and model selection.

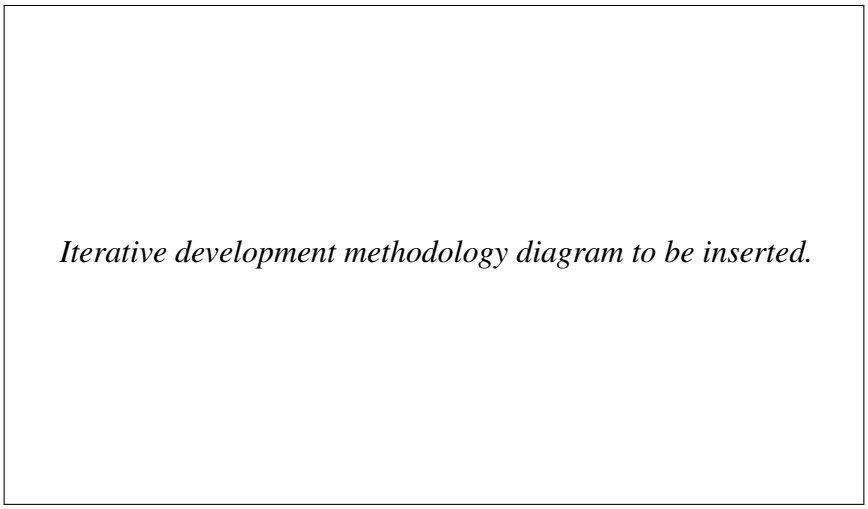

Figure 2: Iterative and Incremental Development Methodology

The project was divided into four main iterations, each building upon the previous:

1. **Iteration 1 — Project Setup and Backend Foundation**: Established the FastAPI backend structure with the router-service architecture pattern. Set up the project repository, defined API schemas using Pydantic, and implemented the health check endpoint. The frontend scaffolding was created using React, TypeScript, and Vite with Tailwind CSS for styling.

2. **Iteration 2 — Inpainting Feature**: Implemented the inpainting service with support for multiple diffusion models (SD Inpainting, SDXL Inpainting, Kandinsky, FLUX.1 Fill). Developed the frontend `InpaintingTab` component with the interactive `MaskCanvas` for drawing masks. Integrated quantisation support (8-bit and 4-bit) using bitsandbytes to enable deployment on memory-constrained GPUs.

3. **Iteration 3 — Style Transfer and Restoration Features**: Added the style transfer service using SDXL img2img pipelines with predefined style presets. Implemented the restoration service integrating CodeFormer, GFPGAN, and Real-ESRGAN models. Developed the corresponding frontend tabs (`StyleTransferTab` and `RestorationTab`).

4. **Iteration 4 — Integration Testing and Deployment**: Configured Google Colab deployment with ngrok for backend exposure. Tested all three features end-to-end on T4 GPU hardware. Refined parameter defaults and error handling based on testing results.

At the end of each iteration, the working software was reviewed and evaluated against the project requirements. Feedback from testing on actual GPU hardware informed the subsequent iteration, particularly regarding VRAM constraints and quantisation strategies. This iterative approach proved well-suited for the project, as requirements around model support and parameter ranges evolved as new models were tested and evaluated.

5.2 System Architecture

DiffusionDesk adopts a **client-server architecture** with a clear separation between the frontend application and the backend API server. The frontend is a single-page application (SPA) built with React that communicates with the backend via RESTful HTTP requests. The backend is a FastAPI server that handles API routing, request validation, and delegates ML inference to dedicated service modules. Figure 3 illustrates the overall system architecture.

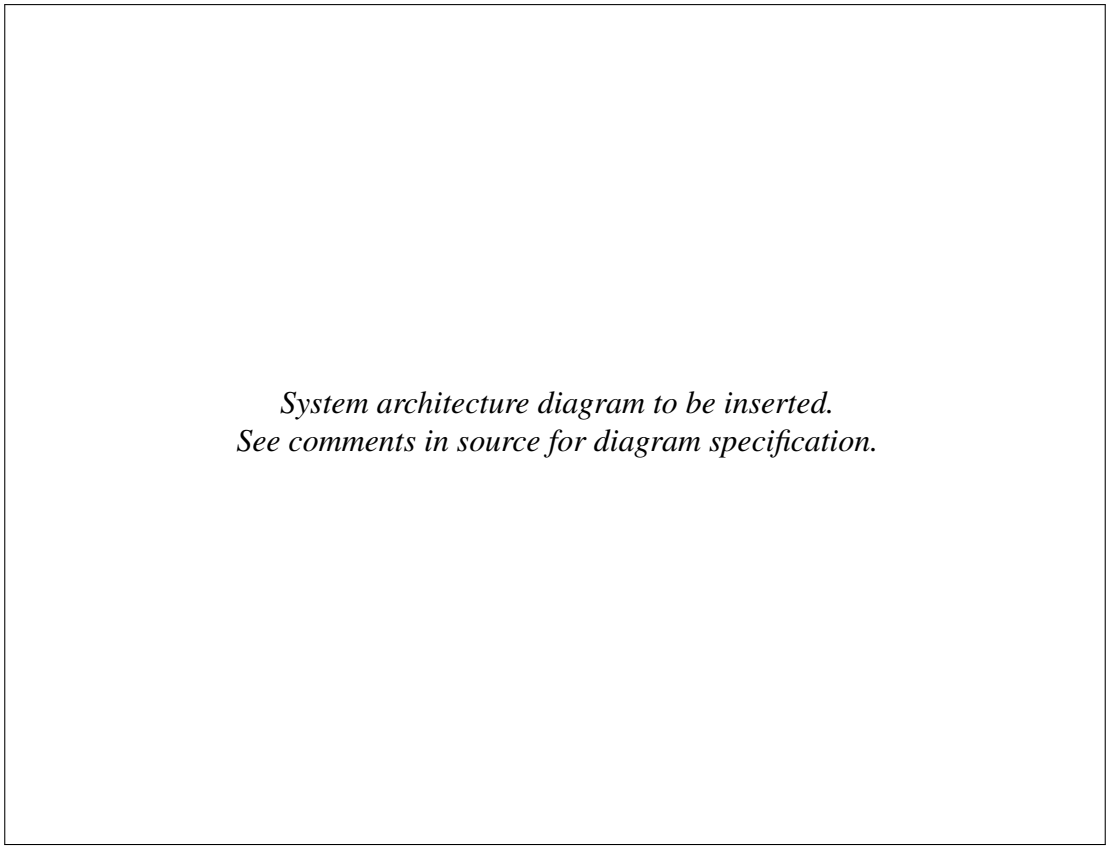

Figure 3: System Architecture of DiffusionDesk

The architecture consists of two main components: the **Frontend Application** and the **Backend API Server**, connected via HTTP/JSON communication. This separation enables independent deployment — the frontend can be hosted on static hosting platforms such as Vercel or Netlify, while the backend runs on a GPU-equipped machine such as Google Colab.

5.2.1 Frontend Architecture

The frontend is a React 19 single-page application built with TypeScript and bundled using Vite. Tailwind CSS provides utility-first styling with a dark theme (slate-900 background with indigo accents). The application is organised into the following layers:

- **App Component** (`App.tsx`): The root component manages tab navigation through an `activeTab` state variable. Four tabs are available: Home, Inpainting, Style Transfer, and Restoration. Each tab renders its corresponding component.
- **Tab Components** (`components/`): Each editing feature is encapsulated in its own tab component (`InpaintingTab.tsx`, `StyleTransferTab.tsx`, `RestorationTab.tsx`). Each tab manages its own local state for image data, generation parameters, and UI state (loading, errors, results). This design ensures that tabs are independent and do not share mutable state.

- **Shared Components**: Reusable components include `ImageUpload` (drag-and-drop file upload with preview) and `MaskCanvas` (HTML5 Canvas-based interactive drawing tool for creating inpainting masks). These are used across multiple tabs to maintain consistency.

- **API Client** (`api/`): The `imageApi.ts` module provides typed API functions (`inpaintImage()`, `styleTransfer()`, `restoreImage()`) that handle base64 encoding, request construction, and response parsing. The `config.ts` module reads the API base URL from the `VITE_API_URL` environment variable, enabling seamless switching between local and cloud-deployed backends.

5.2.2 Backend Architecture

The backend follows a **three-layer router-service-schema architecture** built on FastAPI. This pattern separates concerns cleanly and mirrors the structure used in production web services:

- **Router Layer** (`routers/`): Three router modules (`inpainting.py`, `style_transfer.py`, `restoration.py`) define the API endpoints. Each router receives HTTP requests, extracts validated parameters from Pydantic schemas, delegates processing to the corresponding service, and returns a standardised `ImageResponse`. Routers are registered in `main.py` with URL prefixes (`/api/inpainting/`, `/api/style/`, `/api/restoration/`).

- **Schema Layer** (`schemas/image.py`): Pydantic models define the API contracts. `InpaintingRequest`, `StyleTransferRequest`, and `RestorationRequest` validate incoming data with type checking and range constraints (e.g., `guidance_scale` between 1.0 and 20.0, `num_inference_steps` between 10 and 100). The shared `ImageResponse` model provides a consistent response format with fields for `success`, `image` (base64), `error`, `model_used`, and `processing_time`.

- **Service Layer** (`services/`): The `DiffusionService` class manages diffusion model pipelines for inpainting and style transfer, while the `RestorationService` handles CodeFormer, GFPGAN, and Real-ESRGAN models. Services are instantiated as **singletons** using factory functions (`get_diffusion_service()`, `get_restoration_service()`) ensuring that loaded models are cached in GPU memory across multiple requests. This avoids redundant model loading and reduces response latency for subsequent requests.

5.2.3 Model Management Strategy

A key architectural decision is the **lazy loading with caching** strategy for ML models. Models are not loaded at server startup; instead, they are loaded on demand when the first request for a

given model is received. Once loaded, the model pipeline is cached in a dictionary keyed by model identifier. This approach offers several benefits:

- **Memory efficiency**: Only models that are actively used consume GPU VRAM.
- **Fast subsequent requests**: Cached models skip the loading phase (which can take 30–120 seconds) on repeat use.
- **Flexible model switching**: Users can select different models per request without preloading all options.
- **Quantisation on demand**: Models can be loaded with different precision levels (FP16, 8-bit, 4-bit NF4) based on the user's hardware constraints.

When GPU VRAM is insufficient for a new model, the system clears previously cached models before attempting to load the requested model.

5.2.4 Communication Protocol

The frontend and backend communicate via **RESTful JSON APIs**. Images are transmitted as base64-encoded strings within JSON payloads, which simplifies the API design by avoiding multipart form data. The data flow for a typical editing operation proceeds as follows:

1. The user uploads an image file in the browser. The frontend converts it to a base64 data URL using the `FileReader` API.
2. The user configures parameters (prompt, model, generation settings) through the UI controls.
3. The frontend constructs a JSON payload containing the base64 image, parameters, and (for inpainting) the mask, and sends it as a POST request to the corresponding API endpoint.
4. The backend validates the request using Pydantic schemas, decodes the base64 images to PIL Image objects, and invokes the appropriate service method.
5. The service loads the model if needed, performs GPU inference, and encodes the result image back to base64.
6. The backend returns an `ImageResponse` JSON object containing the result image, model used, and processing time.
7. The frontend converts the base64 response to a data URL and displays the result alongside the original image for comparison.

5.2.5 Deployment Architecture

DiffusionDesk is designed for flexible deployment across different environments:

- **Local development**: The frontend runs on `localhost:5173` (Vite dev server) and the backend on `localhost:8000` (Uvicorn). CORS middleware configured with `allow_origins=["*"]` permits cross-origin requests between the two ports.
- **Google Colab deployment**: The backend runs inside a Colab notebook with GPU acceleration. Ngrok exposes the FastAPI server via a public HTTPS URL. The frontend is configured to point to the ngrok URL through the `VITE_API_URL` environment variable and can be hosted on a static platform or run locally.
- **Production deployment**: The frontend is built to static files using `npm run build` and deployed to Vercel or Netlify. The backend is deployed on a GPU-equipped server (e.g., NTU HPC cluster with V100/A100 GPUs).

5.3 User Interface Wireframe

This section presents the wireframe designs for the key pages of the DiffusionDesk web application. These wireframes provide a visual representation of the interface's layout and functionality, serving as a blueprint for the frontend implementation. The wireframes were designed to prioritise usability, with clear separation of controls, input areas, and result displays.

Home Page

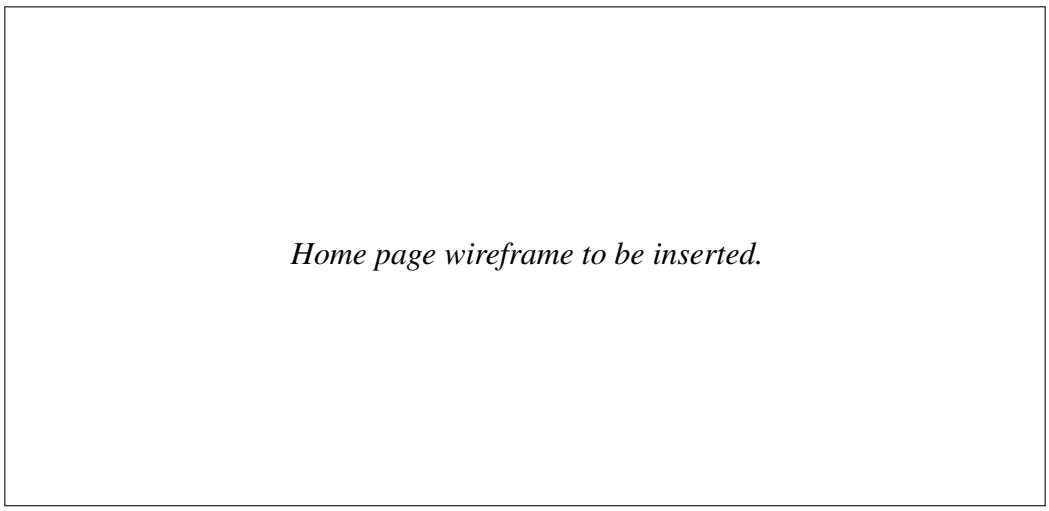

Figure 4: Home Page (Wireframe)

The Home page serves as the landing page for DiffusionDesk. It provides an overview of the application's three core features — Inpainting, Style Transfer, and Restoration — presented as feature cards with brief descriptions. The top navigation bar displays the application name

and tab buttons for navigating to each feature. This page is designed to orient new users and provide quick access to any feature.

Inpainting Page

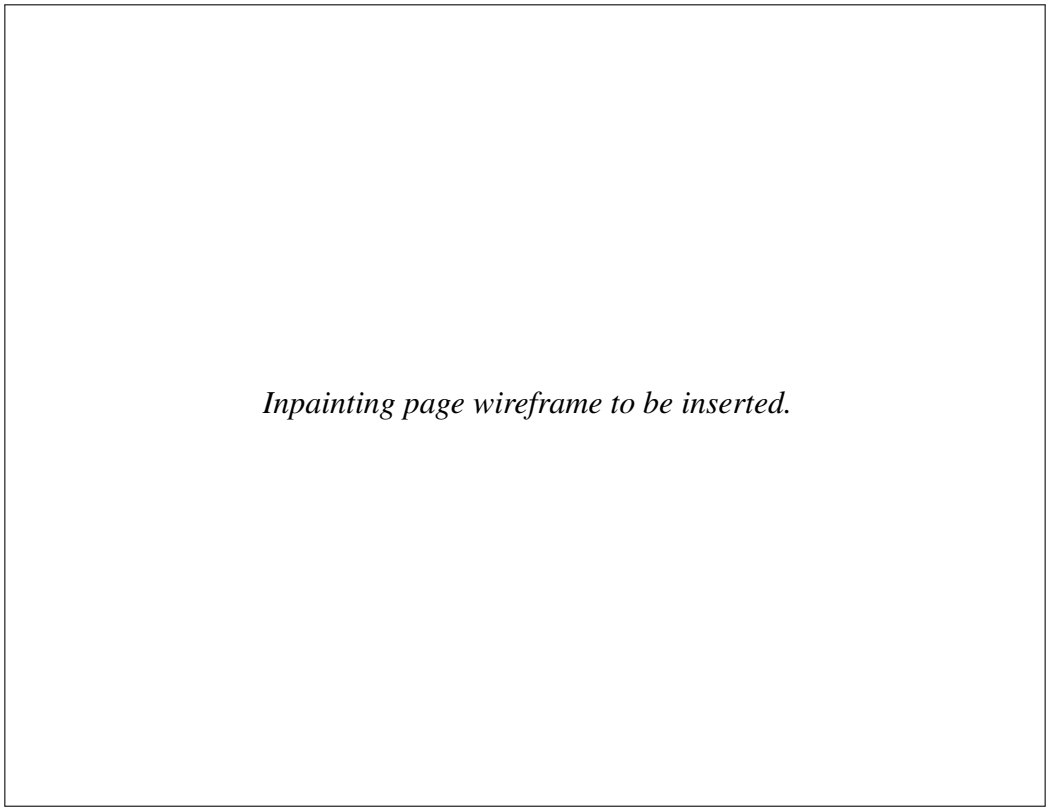

Figure 5: Inpainting Page (Wireframe)

The Inpainting page is the most complex interface in DiffusionDesk. The left side features the image canvas with a mask overlay, where users draw over the region to be inpainted using a configurable brush tool. A brush size slider and "Clear Mask" button are positioned below the canvas. The right side contains the control panel: model selection dropdown, text prompt input, negative prompt input, and a collapsible "Advanced Settings" section with sliders for guidance scale, inference steps, strength, seed input, and quantisation level selection. The "Generate" button triggers the inpainting operation, and the result is displayed below in a side-by-side comparison with the original image. A "Download" button allows saving the result.

The "Generate" button is disabled until three conditions are met: an image is uploaded, a mask is drawn, and a prompt is entered. Inline hints inform the user of any missing requirements.

Style Transfer Page

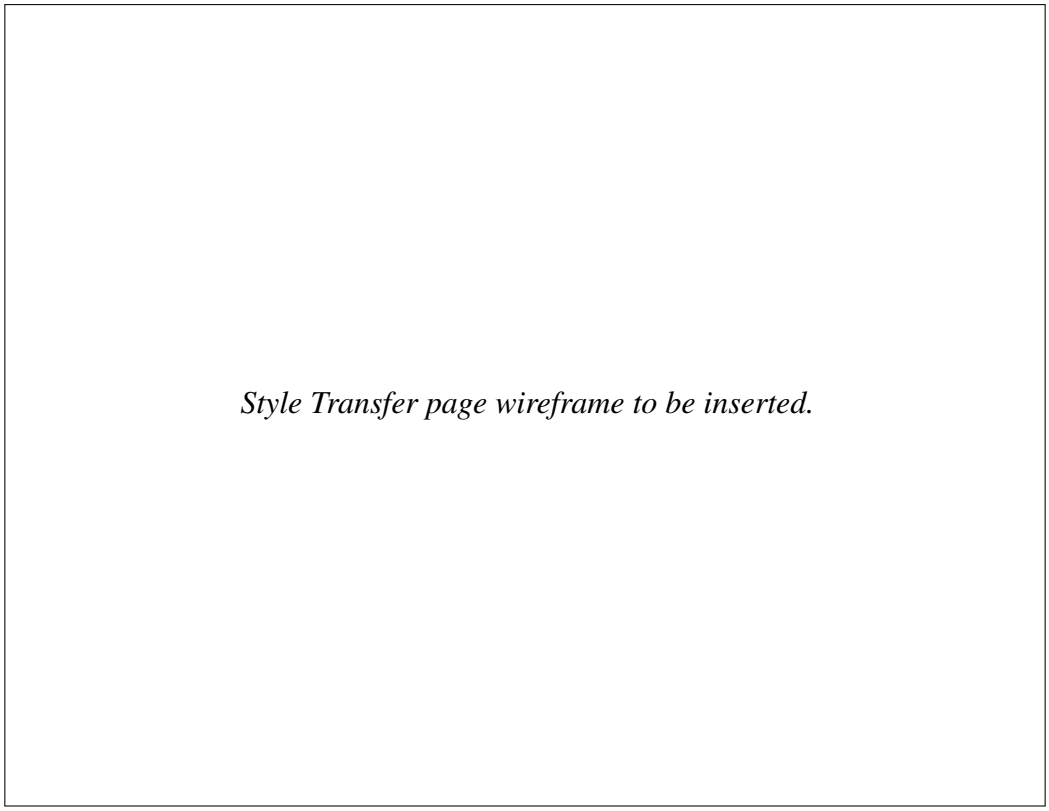

Figure 6: Style Transfer Page (Wireframe)

The Style Transfer page provides a streamlined interface for applying artistic styles to images. The left side displays the uploaded image preview with an upload control. The right side features a grid of style preset buttons (Anime, Oil Painting, Watercolour, Pixel Art, Cinematic, Pencil Sketch, Cyberpunk, Pop Art), each with a visual icon and label. Users can alternatively enable a custom prompt checkbox to enter a free-text style description. A prominent strength slider (0.0–1.0, default 0.7) controls the intensity of the style application — lower values preserve more of the original content, while higher values produce more stylised results. The model selection dropdown and collapsible advanced settings (guidance scale, inference steps, seed) are also available. Results are displayed in the same side-by-side comparison format as the Inpainting page.

Restoration Page

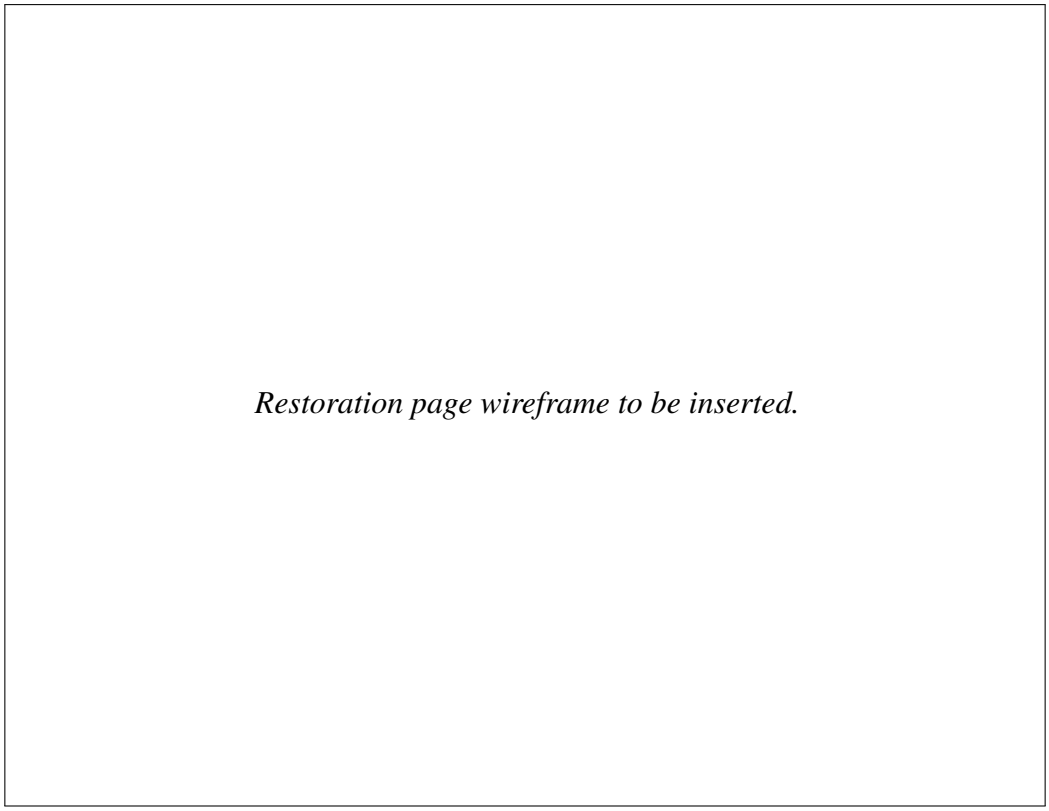

Figure 7: Restoration Page (Wireframe)

The Restoration page offers a checkbox-based interface for selecting restoration operations. The primary options include: Face Enhancement with a model selector (CodeFormer or GFPGAN) and a fidelity slider (for CodeFormer, controlling the balance between quality enhancement and original fidelity); Upscaling with radio buttons for scale factor (None, $2\times$, $4\times$) using Real-ESRGAN; and optional Scratch Removal and Colourisation toggles. The "Restore" button is disabled unless at least one restoration option is enabled. This page does not require a text prompt, making it simpler than the Inpainting and Style Transfer pages. Results are displayed in the same side-by-side comparison format with a download option.

6 Implementation

6.1 Backend Development

6.1.1 Project Structure

6.1.2 API Design

6.1.3 Inpainting Service

6.1.4 Style Transfer Service

6.1.5 Restoration Service

6.1.6 Model Management and VRAM Optimization

6.2 Frontend Development

6.2.1 Project Structure

6.2.2 User Interface Design

6.2.3 Canvas and Mask Drawing

6.2.4 API Integration

7 Project Difficulties and Learning Outcomes

7.1 Project Difficulties

7.2 Learning Outcomes

8 Future Implementation

Bibliography

Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image style transfer using convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Grand View Research (2023). AI image generator market size, share & trends analysis report, 2024–2030. Accessed: 2026-02-02.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ho, J. and Salimans, T. (2022). Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*.

Ma, X., Zhou, X., Huang, H., Jia, G., Wang, Y., Chen, X., and Chen, C. (2023). Uncertainty-aware image inpainting with adaptive feedback network. *Expert Systems with Applications*, 235:121148.

Peebles, W. and Xie, S. (2023). Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Song, J., Meng, C., and Ermon, S. (2021). Denoising diffusion implicit models. In *International Conference on Learning Representations (ICLR)*.

Wang, X., Li, Y., Zhang, H., and Shan, Y. (2021a). GFP-GAN: Towards real-world blind face restoration with generative facial prior. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wang, X., Xie, L., Dong, C., and Shan, Y. (2021b). Real-ESRGAN: Training real-world blind super-resolution with pure synthetic data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*.

Zhou, S., Chan, K. C., Li, C., and Loy, C. C. (2022). Towards robust blind face restoration with codebook lookup transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*.