International Journal of Computing and Digital Systems
2024, VOL. 17, NO. 1, 1–10

http://dx.doi.org/10.12785/ijcds/XXXXXX

Development of a Federated Platform for Accessible AI-Assisted Photo Editing

Oromidayo Owolabi[1] **and Victoria Oguntosin**[1]

[1]*Department of Electrical and Information Engineering, Covenant University, Ota, Ogun State, Nigeria*

Received XX XXXX 2024, Revised XX XXXX 2024, Accepted XX XXXX 2024

Abstract: The rapid advancements in generative artificial intelligence have led to growing demand for accessible and open-source tools that enable users to create and edit images using AI models. However, the high computational requirements and limited variety of models offered by existing applications pose significant barriers to entry. This paper presents a novel federated system of open-source queue servers that connects user clients to GPU clients, allowing users to host, modify, and share AI models freely. The system incorporates a generalisation layer for handling various tasks, a federation protocol for forwarding user requests to appropriate queue servers, and a trust-based priority scheduling scheme for managing bad actors. Experimental results demonstrate the effectiveness of the proposed system in enhancing accessibility and efficiency in generative art. The developed federated network and queue servers have potential applications beyond photo editing, creating new possibilities for collaborative and decentralised AI-assisted content creation. The methodology for this study involves several stages of development to create a fully functioning federated platform for photo editing using AI models. The first stage focuses on the development of a queueing system, which is built using Python, Flask, and WebSockets for communication. The second stage will involve the creation of a client GPU server, which is built using Python, Flask, and SocketIO. In the third stage, a frontend photo editing application is developed using React.js. To ensure that the system can support multiple generative AI models, a Generalization Layer for job execution is created in the fourth stage. The fifth stage involves the development of a federated protocol for server-server communication. Strategies are implemented in the sixth stage to limit bad actors in the system.

Keywords: Federated Platform, AI democratization, Photo Editing

1. INTRODUCTION

In recent years, the rapid advancement of technology has led to significant changes in the way we live and work. One area that has seen particularly rapid development is artificial intelligence (AI), which has become increasingly powerful and capable in a wide range of tasks. The rise of AI systems has led to the development of new technologies that have surpassed traditional software engineering in some sectors, such as computer vision and natural language processing.

One particularly interesting application of AI technology is in the realm of generative art, in which AI algorithms can be used to create original images and designs that are not possible with traditional software. These AI-generated artworks have become increasingly popular and effective, and have even won art competitions. Generative AI systems are able to create unique and original artworks by using complex algorithms to generate variations in the output based on a set of parameters either conditionally or unconditionally. These systems are able to produce a wide range of artistic styles, from photo-realistic images to abstract patterns and designs, by being trained on a vast range of images, such as the LAION-5B dataset [1].

The use of AI in generative art applications often requires a substantial amount of computing power, which can be difficult for some users to access. This has been resolved with the existence of paid web and mobile applications which perform these computations in their servers. However, as these applications abstract away the compute, they also restrict the available models the person can use and their control over them. This has led to a potential demand for a federated platform that will enable users to host, edit, and broadcast these models for the benefit of others while maintaining total transparency. Such a platform would enable users to create and edit images using AI algorithms, without centralized restrictions or the need for powerful computers or specialized hardware.

In order to address this problem and create a federated AI-powered photo editing application, it is necessary to develop open-source queue servers. These queue servers will exist to connect the user clients of the photo editing

E-mail address: author 1, author 2, author 3

application with task-executing GPU clients. By combining the queue servers through a federated network and a photo editing application, we will have a powerful and accessible tool for creating and editing photos.

This study is focused on creating a federated platform for photo editing using AI models. This has the potential to provide a number of benefits, including increased freedom on the part of the users and access to compute resources. By making the platform open, users are able to host and make changes to the AI models they are using without having to pay anyone, which can help to reduce the barriers to entry for using advanced AI technology in photo editing. Additionally, by allowing users to share their models and improvements with others, the platform has the potential to grow and evolve at the pace of AI research, leading to a wide range of new features and capabilities. This can help to make the platform more useful, as well as making it more resilient to changes in the broader AI landscape by not being reliant on certain models or ways of doing things, but instead creating a generalized platform that keeps being useful even with the changes.

AI and generated imagery are on the rise for illustration and photo editing. Currently, the accessibility and the tooling available for these models make them not as widely used. This is due to computing requirements, lack of open-source tooling, and programming experience required to use less widely spread models, among other things. As photography and digital art were great leaps in the evolution of art, generative art could be next great leap but its tooling is still in the early stages.

Currently, the use of advanced AI technology in photo editing is limited by the computing requirements, lack of tooling and lack of awareness of applications by the general population. The companies that manage to bridge this gap do not provide access to a wide range of models due to economic reasons, such as the effort required to train, fine-tune, and market these products to consistently produce high-quality results [2], [3], [4]. This limits the ability of individuals and organizations to take advantage of the latest

AI models and capabilities, and can prevent the wider adoption of these technologies. This research aims to address this problem by creating a federated platform for photo editing using AI models, which will enable anyone to host, use, and make changes to the models freely. This will reduce the barriers to entry for using advanced AI technology in photo editing, and will facilitate the growth and evolution of the platform through the sharing of models and improvements among users.

This study is aimed at creating a federated platform for photo editing using AI models. This aims to increase access to and the proliferation of AI models while enabling privacy, freedom and trust. The objectives of this study are to: Develop a generalizable interface that supports multiple AI models for photo editing; Develop a queueing system

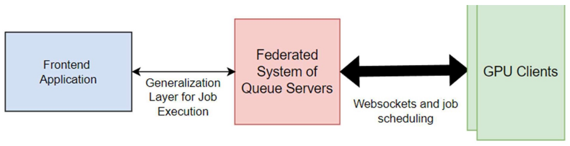

Frontend Application
Generalization Layer for Job Execution
Federated System of Queue Servers
Websockets and job scheduling
GPU Clients

Figure 1. System Block Diagram

and connection model that allows for these models to be seen and used by others; Develop a queue server that implements the connection model, an authentication system and a federated protocol for server-server communication; Develop a frontend application for photo editing with these models, to showcase the ability of the system to run on low-powered devices, while also being flexible to allow any of the models to be used within it; Deploy the queueing servers and frontend application to the cloud.

The federated network and queue servers that is developed is usable in other context than photo editing. An example would be a network of scientists who wish to allow others use their computationally intensive models through the queue servers to foster scientific research. The methodology for this study (Figure 1) involves several stages of development to create a fully functioning federated platform for photo editing using AI models. The first stage focuses on the development of a queueing system, which is built using Python, Flask, and WebSockets for communication. The second stage involves the creation of a client GPU server, which is built using Python, Flask, and SocketIO. In the third stage, a frontend photo editing application is developed using React.js. To ensure that the system can support multiple generative AI models, a Generalization Layer for job execution is created in the fourth stage. The fifth stage involves the development of a federated protocol for server-server communication. Strategies are then implemented in the sixth stage to limit bad actors in the system. After development and testing, the queueing system is deployed on Linode while the frontend application is deployed on Vercel.

2. LITERATURE REVIEW

Generative art, the algorithmic generation of novel images, audio, and other media, has rapidly advanced alongside breakthroughs in artificial intelligence. Specifically, generative adversarial networks (GANs) [5] and diffusion models [6] have demonstrated impressive creative capabilities in recent years, now matching or exceeding human levels for various aesthetic dimensions. Systems like DALL-E 2, Stable Diffusion, and Imagen can synthesise striking high-resolution visuals from text prompts that approach professional quality across manifold styles and subjects.

More recently, there has been an explosion in novel ways for training new models with lower compute requirements, using techniques such as textual inversion [7] and LoRA (Low Rank Adaptation) [8] to further fine-tune these base models. With increased interest in the field, more base models have been made available, from Stable Diffusion XL and Stable Diffusion Turbo to AnalogDiffusion and Pix2Pix. There has also been an increase in modes of interaction with these models, from textual prompting and negative prompting, to ControlNet, outpainting, img2img diffusion, and inpainting methods.

This explosion of creative AI has spawned promising new applications in domains like marketing, game development, and concept art that were previously costly or constrained for independent creators. However, most existing services rely on proprietary platforms with limited control, restrictive content policies, and costly pricing models as they aim to recoup operational resources. Even with recent bursts in the availability of open-source tooling and models for generative art, there is still a gap in accessibility for those who do not have the computing resources to run these models, and also those who wish to experiment with other models and workflows to keep up with this rapid pace without wasting resources and effort.

Diffusion models [6] are a class of generative models that are based on an iterative process of synthesising images by gradually refining them through a series of steps. They are probabilistic models that are designed to learn a data distribution p(x) by iteratively denoising a normally distributed variable. They have achieved high performance in tasks such as image synthesis, conditional image synthesis, super resolution, inpainting and colorization. More recently, latent diffusion models [9] have gained attention in the field of deep learning. Latent diffusion models are a type of diffusion model that incorporate latent variables, which are hidden or unobserved variables that are thought to influence the behaviour of the system being studied. These models improve the efficiency of training and sampling in denoising diffusion models without compromising the quality of the generated samples.

Generative Adversarial Networks (GANs) [5] are a class of generative models that use a two-part neural network to generate synthetic images that are similar to a given set of training data. The GAN architecture consists of the generator and the discriminator, which learn simultaneously. The generator attempts to replicate the underlying distribution of real samples and generates new data samples. The discriminator is typically trained as a binary classifier, with its aim to accurately distinguish between real samples and the generated samples [10]. The training process for a GAN involves optimising the parameters of the generator and discriminator through an adversarial process, with the generator trying to generate realistic-looking images and the discriminator trying to distinguish between real and synthetic images using a minimax optimization process to reach equilibrium. The training process continues until the generator and discriminator reach an equilibrium, at which point the GAN is able to generate synthetic images that are indistinguishable from the real images.

GANs have been widely used for a variety of image generation tasks, including image synthesis, image style transfer, and image super-resolution. They have also been applied to other fields such as speech synthesis [11], where they have achieved promising results. Some examples of applications for GANs in image generation include: Image synthesis, Image style transfer [12], Image super-resolution [13] and Face restoration [14].

Federated software systems consist of independent servers that have a shared communication protocol, effectively creating a union. To achieve this, two layers of communication are implemented. The first is the client-server communication protocol which ensures basic client-server interaction, and the second one, which allows these servers to interface with each other is the server-server protocol. This allows clients to communicate with clients on other servers through message-passing by the server-server protocol. Examples of such protocols are SMTP (Simple Mail Transfer Protocol) and the ActivityPub protocol, which powers decentralised alternatives of social editing apps such as Facebook, YouTube and Twitter, with its alternatives Friendica, Peertube and Mastodon.

A distributed system [15] is a group of independent computing elements that work together as a single, unified system. These systems can include a variety of nodes with different computational capabilities. The key principle is that the nodes can function independently of one another. However, if the nodes do not interact with each other, then there is no point in having them in the same distributed system. Typically, nodes are programmed to work towards a common goal, which is achieved through the exchange of messages between them. A node will respond to incoming messages, process them, and then initiate further communication through message passing. Decentralised systems [16] are distributed systems that do not rely on a central authority or a single point of failure, enabling them to be more resilient and scalable than centralised systems. They are often characterised by their distributed nature, their use of peer-to-peer (P2P) communication, and their reliance on consensus mechanisms to ensure the integrity and security of the system. In this case, each of the computers is a user agent and there are hardly any centralised servers maintaining connections between users. BitTorrent [17], Ethereum [18], and Mastodon [19] are examples of decentralised services that have gained widespread adoption and have had a significant impact. Federated systems have found applications in various domains, including cloud computing environments where load balancing algorithms have been developed to optimize resource allocation [20] [21].

As the number of users of decentralised and federated platforms increase, there are many positive effects, such as more nodes to interact with, and in the case of federated systems, a larger and less centralised network due to the number of active servers. But there are also some negative effects. Due to the lack of centralised control of these systems, they become harder to self-moderate. An increase in the number of users of the platform will lead to an increase in the number of bad actors in the system, making the network less welcoming to users. In this section, methods of moderating these systems so they are beneficial for everyone and perform as expected are introduced.

In decentralised systems, it is important to manage the risk of bad actors, who may attempt to undermine the integrity or security of the system through malicious or selfish behaviour. There are several approaches to managing bad actors in decentralised systems, including:

1) Use of consensus algorithms: Consensus algorithms, such as proof-of-work (PoW) or proof-of-stake (PoS), can be used to secure the network and deter bad actors by requiring them to expend resources or prove their identity in order to participate in the network.
2) Use of reputation or trust systems: Reputation or trust systems can be used to evaluate the trustworthiness or reputation of users or nodes, and to allow or disallow their participation in the network based on their reputation.
3) Use of incentives or rewards: Incentives or rewards can be used to encourage good behaviour and discourage bad behaviour, by providing rewards for positive contributions and penalties for negative contributions.

In federated systems like Mastodon [19], managing bad actors is easier than in the case of fully-decentralised systems. Here are some approaches that can be taken to manage bad actors in a federalized system like Mastodon:

1) Use of moderation teams: Federated systems like Mastodon often have teams of moderators who are responsible for enforcing community guidelines and ensuring that users are behaving in a way that is consistent with the values of the network. These moderators can use tools like account suspension or permanent banishment to manage bad actors.
2) Use of community guidelines: Establishing clear community guidelines can help to deter bad actors by making it clear what types of behaviour are not tolerated on the network. This can include guidelines around hate speech, harassment, and other forms of malicious behaviour. The operation of Mastodon instances follows the principle of content moderation subsidiarity, in which content moderation standards are established and vary between individual instances [22].
3) Use of reporting tools: Federated systems like Mastodon include tools that allow users to report bad actors or inappropriate behaviour. This can help to identify and address problems on the network in a timely manner.
4) Use of incentives or rewards: Incentives or rewards can be used to encourage good behaviour and discourage bad behaviour. For example, Mastodon has a feature called ”boosting” that allows users to amplify the reach of other users’ posts, which can be used as a reward for positive contributions to the network.

When the hosts of a federated system themselves behave as bad actors, it can be more challenging to address. This can be combated by the use of community pressure. This can involve encouraging users to take action, such as by switching to a different host server. When a user chooses to move their account to a different instance, their account data including their blocked, muted, and follower lists and their post history will be migrated, and their followers will automatically follow them to their new account. This eliminates the need to start from scratch when migrating from one federated instance to another. Thus, while federated networks exhibit some degree of clustering, as is characteristic of the Internet, no single instance controls the entire network.

This work directly tackles these issues by creating a federated platform for artificial-intelligence based photo editing with a generalisation layer for models and workflows, alongside a photo-editing application that taps into these capabilities

3. SYSTEM DESIGN

This section is aimed at giving an in-depth analysis of the system and the methodology that was used in the design of a federated platform for AI photo editing. It includes an overview of the high-level architecture of the system, and the design of each of its components which are the queuing server, federated protocol, frontend application and generalization layer. It also introduces the technologies used in the design of the platform.

Lightbox operates on a system of queue servers that interact with each other. The frontend application, used by users, communicates with the queue servers via the HTTP protocol. The GPU clients establish WebSocket connections with their chosen queue server to maintain an open channel. The frontend application serves as a photo editing application, providing users with basic features such as image effects, drawing support, and the ability to load and manipulate images within a canvas. The GPU clients specify the tasks they can perform by utilizing a JSON map, which is then broadcasted to the queue server for user clients to access. The system functions by allowing authenticated users on queue servers to send task execution requests to GPU clients on any queue server. This is made possible through the implementation of a federation protocol and a generalization layer. The System Architecture is shown in Figure 2.

A. Hardware, Software and Functional Requirements

The two main sections of the system for which performance requirements are considered are the queue server and the GPU client, which have separate hardware requirements

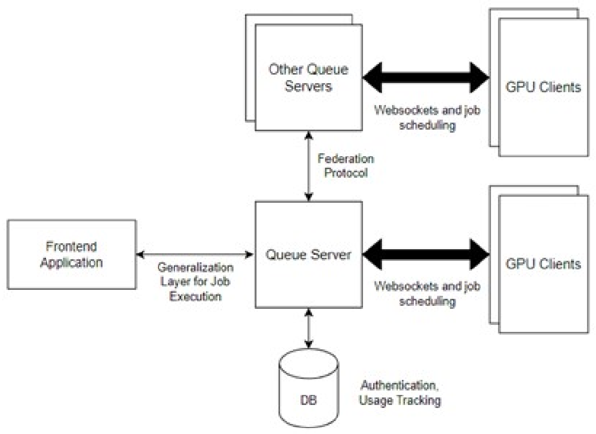

Other Queue Servers
GPU Clients
Websockets and job scheduling
Federation Protocol
Frontend Application
Generalization Layer for Job Execution
Queue Server
GPU Clients
Websockets and job scheduling
DB
Authentication, Usage Tracking

Figure 2. System Design Diagram

as listed in Table I. The frontend client was not considered as it did not have any special operational requirements. It is seen in Table I that the GPU client requires more hardware across the board to effectively run the AI models set up on it. The queue server, GPU client and frontend application have separate software requirements which are outlined in Table II.

The functional requirements of the system are listed:

- The frontend application should be able to perform basic image editing operations on a canvas.
- The frontend application should communicate with the queue servers and GPU clients for task execution using a generalized layer.
- The user should be able to run tasks on the GPU clients through the frontend application.
- The user should be able to make requests from GPU clients connected to different queue servers than the user is registered on.

B. *The Queue Server*

As shown in Figure 3, the queue server uses the WebSocket and HTTP protocols to communicate with GPU and user clients, respectively. It is responsible for scheduling jobs generated by users across GPU clients. Between two queue servers, the only operation supported is federated request passing, which allows job requests to be sent by user clients to GPU clients outside their immediate queue server, allowing the network to be truly federated without centralization of power.

In Table III, the interactions between the user clients, GPU clients and the queue server are laid out. The queue server maintains a mapping of priority queues for each connected GPU client. The priority queue contains all the tasks scheduled for execution for that specific GPU client. The urgency of tasks is sorted by the trust score of the requesting user. Request endpoints wait for a response object to be populated before returning the results to the frontend server.

C. *The GPU Client*

The GPU Client is responsible for executing tasks scheduled by the user clients through the queue server. These task requests and responses are specified in the broadcast information of the client which is a JSON object that specifies all tasks supported by the GPU client alongside their input and output conditions. This allows the system to have multiple GPU clients capable of running different tasks without strict limitations. This specification also makes it easy for the generalisation layer in the frontend to understand what is required as the input of a task and generate the relevant form for that task. The GPU client is a WebSocket client connected to the queue server that receives job requests and channels it to a predefined method in its code. The methods available in our implementation are Text2Img and Img2Img diffusion models with support for inpainting and outpainting, a background removal U-Net, and an image upscaling SRGAN (Super-Resolution Generative Adversarial Network).

D. *The Federation Protocol*

Implementing a federated protocol is crucial for achieving the primary objectives of creating a decentralised AI image editing platform that supports a wide array of models. This protocol enables queue servers to seamlessly share tasks, preventing dependence on a single centralised service or organisation. Multiple GPU clients across servers can be accessed from any server, expanding the pool of available models. The federated protocol involves authenticated message passing between queue servers, ensuring security through an agreed-upon private key. This authentication prevents malicious exploitation of the federated request. To maintain network security, server owners must regularly update their private keys through communication at intervals. Failure to uphold private key homogeneity could lead to network fragmentation. A detailed sequence diagram of federated protocol requests is provided in Figure 4.

E. *The Frontend Application*

The photo editing application created to validate the federated queueing platform was built using React.js and Fabric.js. React.js was used for state management and structuring of the application, while Fabric.js was utilized to control the HTML canvas. The application is divided into the toolbar, canvas, and sidebar sections, as demonstrated in Figure 5.

The toolbar encompasses traditional photo editing application tools such as a brush and color picker, eraser, select tool, effects, and a textbox, which provide basic functionality to the application. The canvas is the primary focus of the program, where users can upload and download images, paint, erase, transform images, delete objects, apply

TABLE I. Hardware Requirements

Components	Queue Server	GPU Client
RAM	2GB	16GB.
Processor Speed	1.8 GHz	2.4GHz
Hard Disk Storage	4GB	40GB of hard disk storage
GPU Requirements	None	Nvidia GeForce RTX 2060 with 6GB of VRAM

TABLE II. Software Requirements

	Frontend Application	Queue Server	GPU Client
Language Environment	Node.js v18	Python 3.10	Python 3.10
Major Library/Framework	React.js	Flask	Pytorch
Supporting Libraries	Fabric.js	Flask_SQLAlchemy	Huggingface, Diffusers
Networking Library	Axios	Flask_socketio	socketio

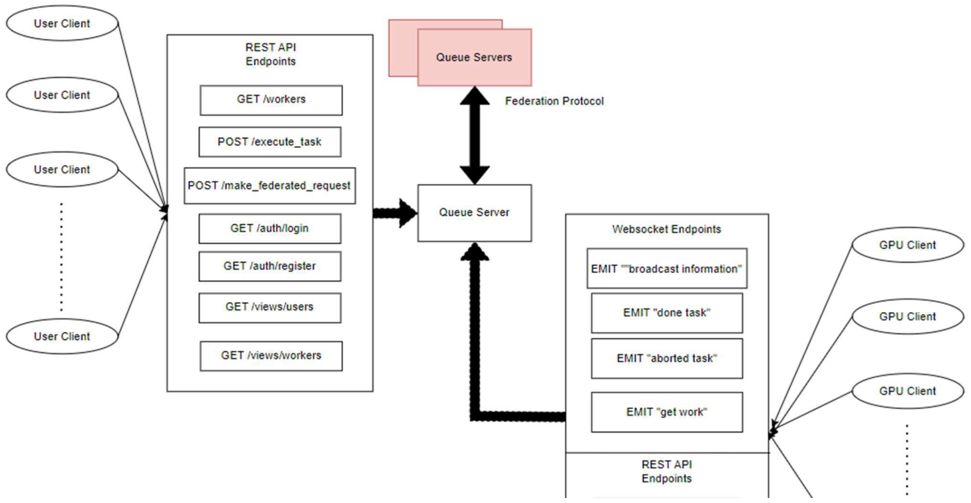


Figure 3. Queue Server Endpoints

TABLE III. Client-Server Interactions

	User Client	GPU Client
Registration and Authentication	User registration and authentication	GPU client registration and authentication.
Task information	Gets the task information of all the GPU clients connected to a queue server	Broadcasts its task information to the queue server.
Job request and execution	Can send job requests to GPU clients in the queue server	Notifies the queue server of its availability to perform jobs
Connection with the queue server	Interacts with the queue server using HTTP requests.	Maintains a WebSocket connection with the queue server.
Interaction with external servers	Can send job requests to GPU clients in another queue server.	Does not interact with external queue servers.

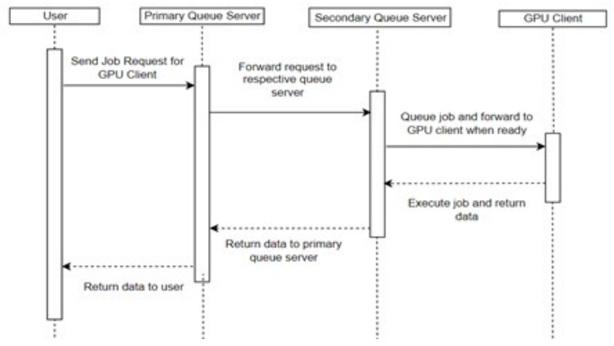

User
Primary Queue Server
Secondary Queue Server
GPU Client
Send Job Request for GPU Client
Forward request to respective queue server
Queue job and forward to GPU client when ready
Execute job and return data
Return data to primary queue server
Return data to user

Figure 4. Federation Protocol Sequence Diagram

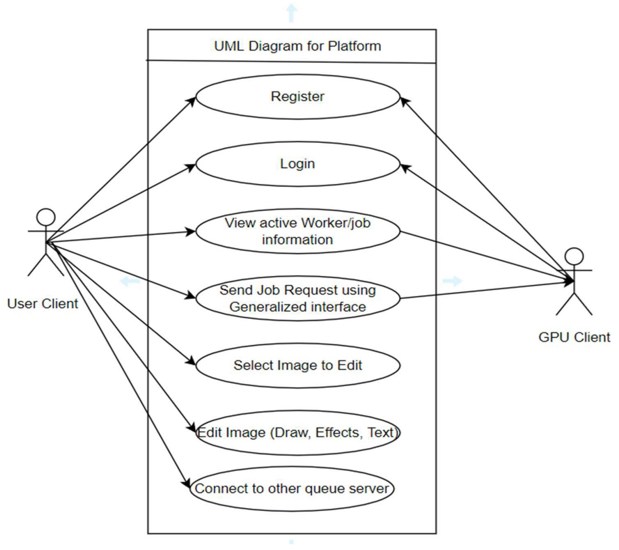

UML Diagram for Platform
Register
Login
View active Worker/job information
Send Job Request using Generalized interface
Select Image to Edit
Edit Image (Draw, Effects, Text)
Connect to other queue server
User Client
GPU Client

Figure 5. Use Case Diagram for Frontend Application

effects, utilize sections as input for AI models, and place images generated by the federated GPU clients.

The sidebar serves as the interface for the generalization layer and the queue server. Users can specify the queue server they wish to connect to, authenticate themselves with their token, enable federated mode, and specify a secondary queue server. They can also refresh worker information of the selected queue server, specify the GPU client and task to run, input information required for the tasks to run, and view and use the returned images from the GPU client and queue server.

The frontend photo editing application serves as the sole interface for users to interact with the entire system. Therefore, it is responsible for creating all the affordances to enable full utilization of the system's capabilities. The frontend application is accountable for several tasks, including basic photo editing operations such as text, effects, manipulating images, and drawing. It is also responsible for authenticating a user with their queue server, viewing GPU broadcast information from a queue server, and task information from a selected task.

Additionally, the frontend photo editing application generates the input form for the chosen task and loads the task's results upon completion. Finally, the application allows users to upload and save images when done with editing. In summary, the photo editing application plays a critical role in enabling users to leverage the full capabilities of the federated queueing platform, allowing them to edit and process images while leveraging the power of distributed computing.

F. *The Generalisation Layer*

The generalisation layer as shown in Figure 6 is responsible for allowing interfacing and execution of various types of ML tasks. It provides support for a wide variety of models and allows us to interact with various GPU clients and their tasks through their broadcast specifications. The generalisation layer works by generating form inputs based on the task input specifications in the broadcast of the GPU client. When the form is completed, the relevant data is propagated to the respective GPU client through the queue server. When the task is completed, the results are displayed depending on the task's output specification.

G. *User Moderation*

Given the federated nature of the system and the fact that the system was designed to execute intensive tasks, it is important to enforce measures which prevent abuse of the system and maintain the quality of the service for all users. These measures will be taken to punish those who take advantage of the system and overuse it and reward those who do not exploit the system. The measures taken are:

1) Priority-based task scheduling: Based on a client's usage of the system, a trust score will be assigned to them. The system can prioritise tasks submitted by trusted entities and assign them a higher priority in the task queue. This ensures that high-priority tasks are completed first and trusted users are rewarded. This helps to prevent bad actors from interfering with important tasks. For entities that are not trustworthy (external queue server requests), a task score below the average value is assigned to them.
2) Account suspension and creation measures: Based on users computed trust scores, if they are low enough, the account can be suspended for a period of time. In order for this measure to have an impact on malicious acts, account creation should be a harder process than simply signing up. Email verification and CAPTCHAs were used to enforce this.

H. *Database Design*

A relational model was chosen to serve as the database of the queue server. It consists of two tables which are the user and worker table and is used to persist information

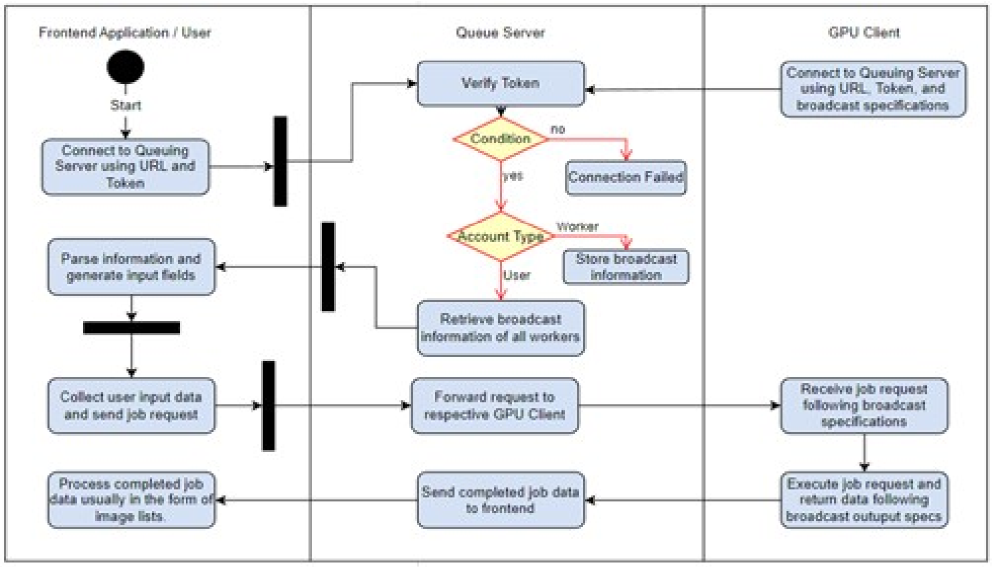

Frontend Application / User
Queue Server
GPU Client
Start
Connect to Queuing Server using URL and Token
Verify Token
Connect to Queuing Server using URL, Token, and broadcast specifications
Condition
no
Connection Failed
yes
Account Type
Worker
Store broadcast information
User
Parse information and generate input fields
Retrieve broadcast information of all workers
Collect user input data and send job request
Forward request to respective GPU Client
Receive job request following broadcast specifications
Execute job request and return data following broadcast outuput specs
Process completed job data usually in the form of image lists.
Send completed job data to frontend

Figure 6. Activity Diagram for Generalisation Layer

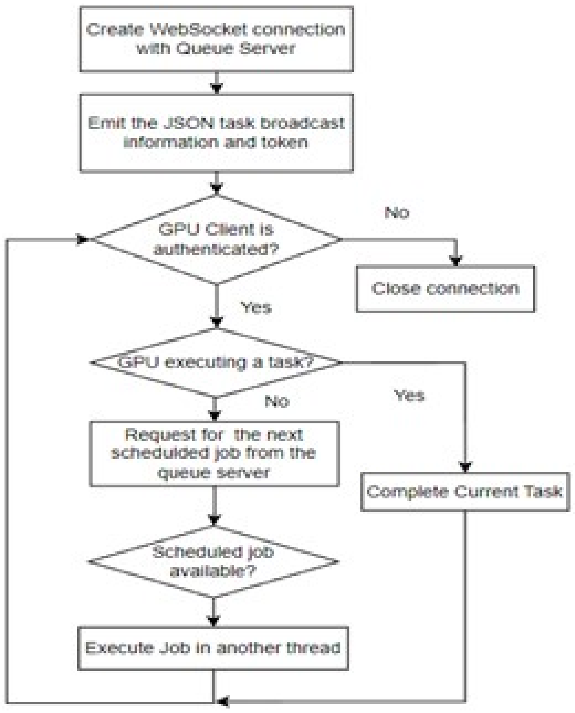

Create WebSocket connection with Queue Server
Emit the JSON task broadcast information and token
GPU Client is authenticated?
No
Close connection
Yes
GPU executing a task?
No
Yes
Request for the next scheduled job from the queue server
Complete Current Task
Scheduled job available?
Execute Job in another thread

Figure 7. GPU Client execution loop

related to authentication, priority and transactions. Transaction information itself is not stored to prevent censorship in the system. The two table layouts are in Table IV.

Table IV contains all the user data that is stored by the queue server. This includes the username, email, password_hash and admin field for authentication and the disabled, trust_score and last_update_time fields for scheduling management. No other user information is stored.

The workers table in Table V contains all the GPU client data that is stored by the queue server. This includes the username, email, password_hash and admin field for authentication and the disabled, and last_broadcast fields for interested user clients to know the last status of the worker. No other worker information is stored.

I. Software Tools

Python: It is a high-level dynamic programming language that is widely used in various fields of software development. Its extensive standard library and third-party libraries make it possible to perform a wide range of tasks such as web development, data analysis, scientific computing, and artificial intelligence.

React.js: It is an open-source JavaScript library that is used to build user interfaces. React.js uses a declarative approach to build UI components, which makes it easy to create complex user interfaces. One of the key features of React.js is its virtual DOM, which is a lightweight representation of the actual DOM. The virtual DOM helps to improve performance by reducing the number of DOM updates, which can be expensive. React.js also has a component-based architecture that allows developers to create reusable UI components that can be easily plugged into other parts of the application.

Flask: It is a lightweight and flexible web framework written in Python that is used to develop web applications. It is ideal for small to medium-sized web applications that require a minimalistic approach. It is built on top of the Werkzeug toolkit and the Jinja2 template engine.

TABLE IV. User Database Table

S/N	Field	Data Type	Description
1	id	Int	Unique identification field of the user
2	username	String	Display name chosen by the user
3	email	String	Unique email of the user.
4	password_hash	String	Hash of the user's password. Used for authentication
5	created	DateTime	Timestamp of account creation.
6	admin	Boolean	Is true if account has administrator privileges
7	disabled	Boolean	Is true if account is disabled
8	trust_score	Int	The trust level of the user client
9	last_update_time	DateTime	The last time the trust score was updated when the user scheduled a task.

TABLE V. Worker Database Table

S/N	Field	Data Type	Description
1	id	Int	Unique identification field of the client
2	username	String	Display name chosen by the client
3	email	String	Unique email of the client.
4	password_hash	String	Hash of the client's password. Used for authentication
5	created	DateTime	Timestamp of account creation.
6	admin	Boolean	Is true if account has administrator privileges
7	disabled	Boolean	Is true if account is disabled
8	last_broadcast	String	The last JSON task broadcast sent out by the GPU client

Werkzeug provides low-level utilities for handling HTTP requests and responses, while Jinja2 is a templating engine that makes it easy to create dynamic HTML pages. Flask also supports various extensions that provide additional functionality, such as Flask-SQLAlchemy for working with databases.

SQLAlchemy: It is a popular SQL toolkit and ORM (Object-Relational Mapping) library for Python. SQLAlchemy makes it easy to work with relational databases by providing a high-level API that abstracts away the complexity of working with databases. It supports various databases such as MySQL, PostgreSQL, SQLite, and Oracle. It provides a powerful ORM that makes it easy to work with databases using Python objects. Its ORM supports various relationships such as one-to-one, one-to-many, and many-to-many.

HuggingFace: It is a leading open-source repository for machine learning models and datasets primarily built using Python. It provides models uploaded by other people over a range of tasks from image Generation, Image Segmentation, Natural Language Processing, Reinforcement Learning, and Object Detection among many others. HuggingFace provides libraries and pipelines ease usage of these hosted models across devices. It also takes advantage of the PyTorch machine learning library for model building and inference.

Fabric.js: It is an open-source JavaScript library that provides a powerful and flexible way of building interactive HTML5 canvas applications. It provides a comprehensive set of APIs for creating and manipulating canvas elements, such as shapes, text, and images. Its modular architecture allows for the creation of reusable components and provides support for event handling and animations.

4. RESULTS AND DISCUSSION

This section presents the work done during the development of the Federated Photo Editing Platform. It covers the implementation of the GPU Client, queue server and frontend application. The WebSocket model, and AI models in use are all analyzed. The frontend application is tested on all the models and simulations are performed to confirm that the system performs under different setups. It also covers the hosting of the final application. The software implementation covers the implementation of the GPU Client, its WebSocket client, its model broadcast and supported models. It also covers the implementation of the queue servers, the federated protocol and the photo editing application. The code is available on GitHub at [23], [24], [25], [26].

A. *GPU Client*

The GPU client was implemented using Python, Flask, and open-source AI models obtained from HuggingFace and GitHub. The methods available in our implementation are Text2Img and Img2Img diffusion models with support for inpainting and outpainting, a background removal U-Net, and an image upscaling SRGAN. Figure 7 details the operation model of the GPU client. The supported text-image generation models were AnalogDiffusion and StableDiffusion v1.4.

1) Text2Img, Img2Img and Inpainting: The supported text-to-image generation models were AnalogDiffusion and StableDiffusion v1.4. The supported image-image models are InstructPix2Pix and StableDiffusion v1.4 img2img. The supported inpainting model was RunwayML/stable-diffusion-inpainting. The general implementation for these in the broadcast is available at modelbroadcast.py in the GPU client code [25].
2) Outpainting: Outpainting is an extension of inpainting which serves to extend an image's boundaries and size. As such, the underlying model was still RunwayML/stable-diffusion-inpainting.
3) Image Upscaling and Background Removal: The image upscaling and background removal models were obtained from GitHub repositories rembg [27] and Real-ESRGAN [28], and were already packaged to require a shell command for execution.

B. Queue Server

The queue server was implemented using Python, Flask and WebSockets. It contains code for the federated protocol, task scheduling, and signup and authentication for GPU and user clients. The registration process has two phases to prevent bots and bad actors on the system. A CAPTCHA must be completed to register. On registration, an email is sent containing the token required for authentication by the queue server. This is also the case for successive logins to confirm the ownership of the email account. As a result, fake email addresses cannot be used and the effort required to create multiple accounts on the queue server is greatly increased. JSON Web Tokens are used for authentication. They are generated on sign-in and have an expiry period. This increases their security as they cease being risks after a while and the user must log in again to get their token. This will prevent mass token stealing and impersonation attacks on the queue server.

We describe the algorithm of how the queue server maintains its internal state, job scheduling and message passing between the user and GPU clients. When a WebSocket connection is created between the GPU client and queue server, the server creates a separate priority queue for all jobs addressed to it. The server also stores the broadcast information of that GPU client in the database and in a worker_broadcast_map to be sent to user clients when needed. User clients can request both local and federated task execution. When this happens, the relevant information for that task is placed in the PriorityQueue of the relevant GPU client being addressed. The GPU client constantly polls for jobs when not busy and will eventually get that task to execute. The user client will await this task completion and get the returned data from the GPU client when the task is complete.

C. The Federated Protocol

The federated protocol essentially functions as a form of request passing between two queue servers. It is implemented in 2 routes on the queue server:

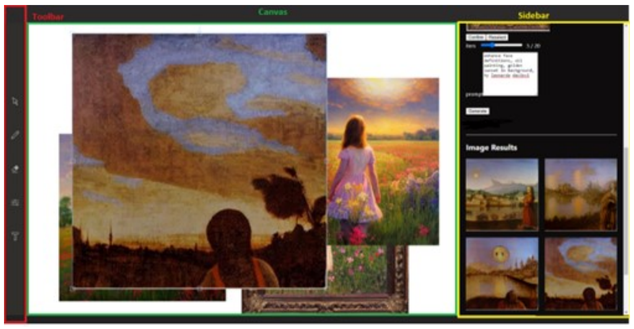


Figure 8. Text-to-Image generation on the Frontend

1) The route to make a federated request takes the task information from the user alongside the requested GPU client and URL of the secondary queue server and forwards the request to that server alongside an agreed token to ensure that the request is from a queue server.
2) The route that receives a federated request verifies the token and executes the task on the requested GPU client. The priority of a federated task is set to be much lower than the baseline to place priority on local members whose activities can be moderated and to prevent exploitation of the protocol by bad actors.

D. The Frontend Application

The photo editing application created to validate the federated queueing platform was made using React.js and Fabric.js. React.js handled all state management and structure of the application while Fabric.js was used to control the HTML canvas. As seen in the Figure 7, the application can be divided into the toolbar, canvas and sidebar sections. The frontend application, through the generalisation layer and the federated network, supports Txt2Img, Img2Img, inpainting and outpainting diffusion models alongside background removal and image upscaling as AI features. For traditional photo editing features, the application supports drawing, erasing, textboxes, dragging images around the canvas space and resizing them, and applying effects such as saturation, brightness, contrast, hue rotation and pixelation changes.

In Figure 8, multiple requests were made to the queue server through the sidebar by using the form generated through the generalisation layer. The results were then displayed in the image grid on the sidebar and placed and scaled on the canvas. In Figure 9, a request was made through the queue server to generate a white house with greenery. Then a request was made to remove the background from the generated image. This result was placed in the image grid and displays just the house with the background removed. This was accomplished with the rembg model [27]. In Figure 10, a generated white house with greenery was first made using Stable Diffusion. Then a request was made to outpaint this image and placed

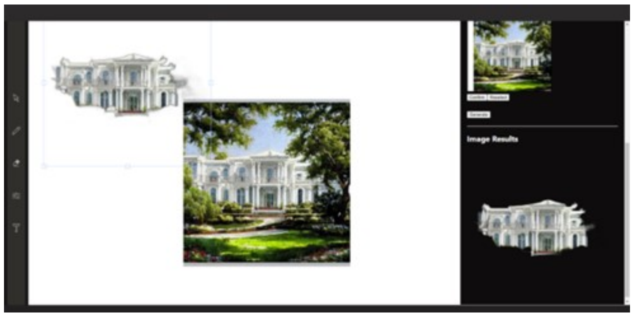


Figure 9. Background Removal on the Frontend

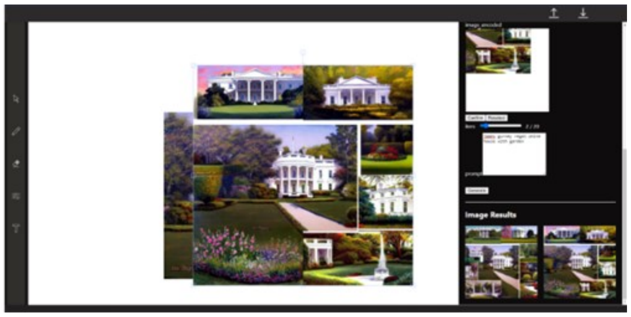


Figure 10. Outpainting on the Frontend

on the canvas. This increased the size of the image by generating more images around it using sections of the original image as input. In Figure 11, Analog Diffusion was used to generate images. Then an Img2Img model was used to generate the images shown in the bottom right panel. One of the generated images had its background removed and was edited using a green brush and an eraser. Figure 12 is the result of the effects configuration panel. Noise was added to the image, the hue was rotated, the image was pixelated, and the image contrast and saturation were increased.

E. Publishing and Deployment

In keeping with the aims of this work, the code for the GPU client [25], frontend application [23] and queue server [24] was released on GitHub under the General Public License v3 (GPL v3). This will allow anyone to

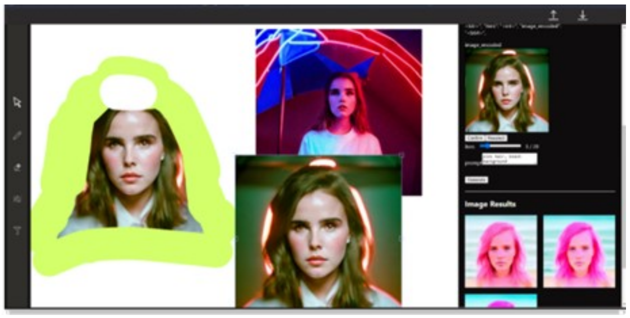


Figure 11. Image-to-Image generation and editing on the Frontend

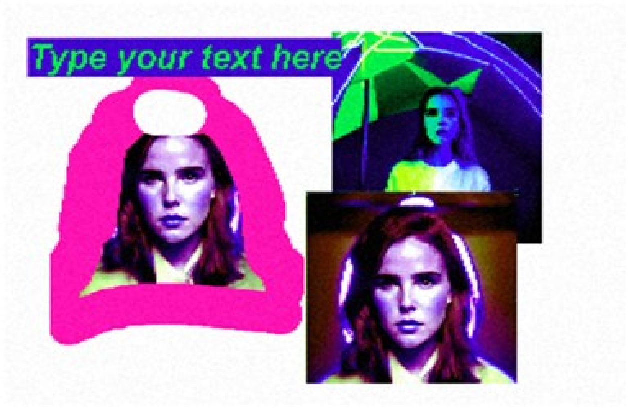
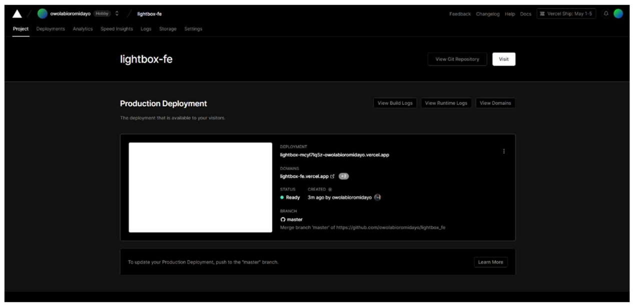


Figure 12. Image Effects editing on the Frontend



lightbox-fe

Production Deployment

The deployment that is available to your visitors.

DEPLOYMENT
lightbox-mcyl7lq5z-owolabioromidayo.vercel.app

DOMAINS
lightbox-fe.vercel.app +2

STATUS Ready

CREATED 3m ago by owolabioromidayo

BRANCH
master
Merge branch 'master' of https://github.com/owolabioromidayo/lightbox_fe

To update your Production Deployment, push to the "master" branch.

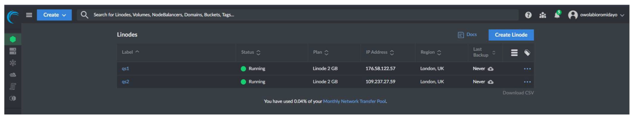
Figure 13. Deployment on Vercel

modify and use the code provided their application is also open. The frontend application was deployed on Vercel (Figure 13). Vercel is a cloud-based platform that supports seamless deployment of Node.js based web applications. Vercel provides features such as automatic SSL certification, asset optimization, and real-time analytics.

The backend application was deployed on Linode (Figure 14). Linode is a cloud computing platform that offers Virtual Private Servers (VPS) to users, allowing them to deploy and manage scalable cloud infrastructure. In order to setup the queue servers on Linode, Linux service scripts had to be setup and linked to bash scripts to start the flask server. This was done to ensure the application restarts if the VPS restarts.

F. Simulations

Simulations were run to evaluate the performance of the queue server over multiple servers using trust scores. These simulations were carried out by multi-threading server, user and GPU instances and randomising trust scores, ids, and


Create
Search for Linodes, Volumes, NodeBalancers, Domains, Buckets, Tags...
owolabioromidayo
Docs
Create Linode
Download CSV

Linodes

Label	Status	Plan	IP Address	Region	Last Backup
qs1	● Running	Linode 2 GB	176.58.122.57	London, UK	Never
qs2	● Running	Linode 2 GB	109.237.27.59	London, UK	Never

You have used 0.04% of your Monthly Network Transfer Pool.

Figure 14. Deployment on Linode

wait time between tasks, all within reasonable bounds, to get a model of the systems performance under assumed normal operations. The number of servers, users and GPU clients were varied to see how the system scales. The other purpose of the simulations is to prove that the queue system functions with multiple users and external queue servers interacting with it.

In the multiple server setup, 4 servers were used, each with an equivalent number of users and GPU clients as in the single server setup i.e. (10 users in the single server setup becomes 10 users for each of the 4 servers). The GPU client count scaled with the total user count by 0.1 i.e. (1 GPU client for every 10 users). For the multiple server setup, user clients choose randomly between making federated requests to other GPU clients or making task requests to their own GPU clients, thus more correctly modelling the final nature of the system. For both setups, users were initialised with a random trust score and all their activities were logged by the server. Table VI shows the simulation parameters.

The time variables for the simulations were scaled on a basis of time scaling roughly 10 times to get more logs. This means 7 minutes of simulation time was made to simulate 1 hour of real time usage. The other variables such as the GPU TASK TIME and USER SLEEP TIME can also be multiplied by 10 to get the real time value. As the simulations were performed by creating hundreds of virtual threads in Python (reaching a maximum of 444 concurrent threads), a language which does not have support for true parallelism, as the servers were just normal threads equally competing for processing time, the performance results of the queue servers in both single and multiple setups are expected to be better than that of the simulations.

From figures 15a and 15b, it can be seen that the task response time drops drastically as the trust score is increased. It is important to note that the values of the bars are added upon each other and each colour region does not contain hidden parts. As the trust score is reduced per count, with a reset interval, it acts as a rate limiter and ensures that those who have not been overusing the system get significantly better quality of service. This also ensures that malicious actors who may have spawned their own queue servers do not worsen the system for other users. The graphs also show that the response time increases with the number of users in the system.

G. Discussion

The implementation of our federated platform for AI-assisted photo editing reveals both promising results and areas for improvement. By distributing computational tasks across a network of GPU clients, we've successfully lowered the barrier to entry for AI-powered image manipulation. Users without high-end hardware can now access a variety of AI models for tasks like image generation, inpainting, and style transfer. The generalisation layer proved effective in integrating diverse AI models, showcasing the system's flexibility. Additionally, our trust-based priority scheduling showed potential in balancing task execution times and managing system load, though further refinement is needed for larger-scale deployments. However, challenges remain in ensuring consistent availability of GPU clients and maintaining performance across a growing network. The system's reliance on volunteer GPU clients may lead to inconsistent access to certain models, potentially impacting user experience. Future work should focus on implementing more robust load balancing algorithms and expanding the range of supported AI models. Exploring applications beyond photo editing could also reveal new opportunities for this federated approach. As we continue to develop this platform, addressing these challenges is important in realising the full potential of democratised access to AI-powered creative tools.

5. CONCLUSION

In conclusion, this paper introduces a federated platform that makes AI-assisted photo editing accessible to a wider audience by distributing tasks across a network of queue servers and GPU clients. The system allows users to register on queue servers, which broadcast available tasks and their input requirements to a generalisation layer in the application. This enables users to submit jobs that are efficiently scheduled and executed on GPU clients, empowering devices without powerful graphics processing capabilities to leverage advanced AI models for creative applications. Simulations were also performed to affirm that the trust-score based scheduling system is effective in single and multiple-server setups. Future improvements could focus on enhancing fault tolerance, expanding input type support, and integrating with existing creative software. This work contributes to the democratisation of generative AI technologies and fosters a collaborative ecosystem for creative experimentation and innovation. The modular design of the platform makes it adaptable to various domains, presenting exciting possibilities for future research and development, such as scientific computing [29].

6. Acknowledgment

The authors acknowledge Covenant University Centre for Research, Innovation and Discovery (CUCRID) in Covenant University for the actualization of this research work for publication.

References

[1] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman *et al.*, "Laion-5b: An open large-scale dataset for training next generation image-text models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 25 278–25 294, 2022.

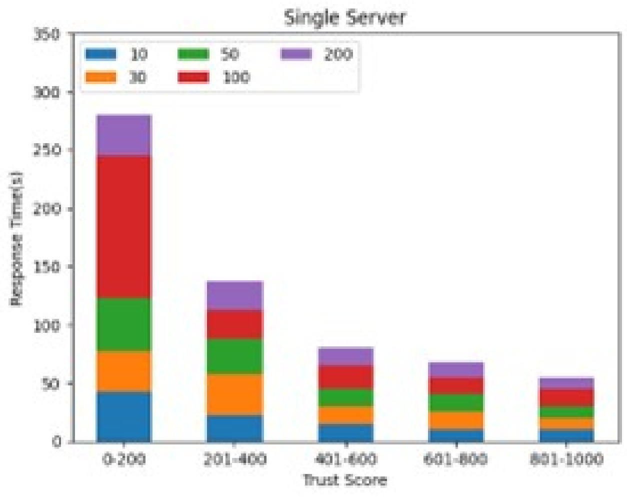
[2] "Dreamstudio," https://beta.dreamstudio.ai/dream, 2023, accessed: September 4, 2024.

[3] "Alpaca - humans: Ai models for image generation," https://www.getalpaca.io/, 2023, accessed: September 4, 2024.

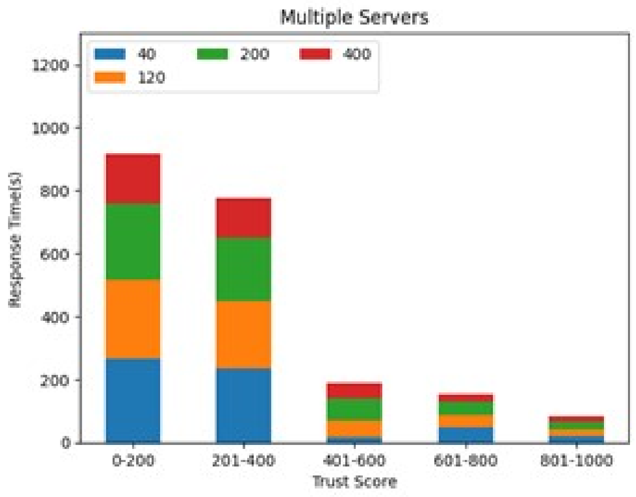
[4] Runway, "Runway - everything you need to make anything you want," https://runwayml.com/, 2023, accessed: September 4, 2024.

TABLE VI. Simulation Parameters

PARAMETER	VALUE	DESCRIPTION
USER_COUNT	—	Number of users in simulation.
GPU_SCALING_RATIO	0.1USER_COUNT	Number of GPU clients
FEDERATED_TRUST_SCORE	400	Trust score for a federated request
GPU_TASK_TIME	5 seconds	Time taken to complete a task
USER_SLEEP_TIME	10-16 seconds	Sleep time between requests
SIM_TIME	7 minutes	Time taken for a simulation



(a)



(b)

Figure 15. Simulation results for (a) Single Server Setup; (b) Multiple Server Setup;

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[6] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*. PMLR, 2015, pp. 2256–2265.

[7] R. Gal, Y. Alaluf, Y. Atzmon, O. Patashnik, A. H. Bermano, G. Chechik, and D. Cohen-Or, "An image is worth one word: Personalizing text-to-image generation using textual inversion," *arXiv preprint arXiv:2208.01618*, 2022.

[8] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[9] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[10] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng, and F.-Y. Wang, "Generative adversarial networks: introduction and outlook," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 4, pp. 588–598, 2017.

[11] J. Kong, J. Kim, and J. Bae, "Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis," *Advances in neural information processing systems*, vol. 33, pp. 17 022–17 033, 2020.

[12] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[13] X. Wang, L. Xie, C. Dong, and Y. Shan, "Realesrgan: Training real-world blind super-resolution with pure synthetic data supplementary material," *Computer Vision Foundation open access*, vol. 1, no. 2, p. 2, 2022.

[14] X. Wang, Y. Li, H. Zhang, and Y. Shan, "Towards real-world blind face restoration with generative facial prior," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 9168–9178.

[15] M. Van Steen and A. S. Tanenbaum, "A brief introduction to distributed systems," *Computing*, vol. 98, pp. 967–1009, 2016.

[16] C. V. B. Murthy, M. L. Shri, S. Kadry, and S. Lim, "Blockchain based cloud computing: Architecture and research challenges," *IEEE Access*, vol. 8, pp. 205 190–205 205, 2020.

[17] J. Gong and N. J. Navimipour, "An in-depth and systematic literature review on the blockchain-based approaches for cloud computing," *Cluster Computing*, pp. 1–18, 2021.

[18] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[19] C. R. Shaw, "Decentralized social networks: Pros and cons of the mastodon platform," 2020.

[20] N. V. Patrick, S. Misra, E. Adetiba, and A. Agrawal, "An incremental load balancing algorithm in federated cloud environment," in *Data, Engineering and Applications: Select Proceedings of IDEA 2021*. Springer, 2022, pp. 395–408.

[21] V. P. Nzanzu, E. Adetiba, J. A. Badejo, M. J. Molo, M. B. Akanle, K. D. Mughole, V. Akande, O. Oshin, V. Oguntosin, C. Takenga *et al.*, "Fedargos-v1: A monitoring architecture for federated cloud computing infrastructures," *IEEE Access*, vol. 10, pp. 133 557–133 573, 2022.

[22] A. Z. Rozenshtein, "Moderating the fediverse: Content moderation on distributed social media," *J. Free Speech L.*, vol. 3, p. 217, 2023.

[23] O. Owolabi, "Lightbox frontend code," https://github.com/owolabioromidayo/lightbox_fe, 2023, accessed: September 4, 2024.

[24] O. O, "Lightbox queue server," https://github.com/owolabioromidayo/lightbox_queue_server, 2023, accessed: September 4, 2024.

[25] ——, "Lightbox gpu client," https://github.com/owolabioromidayo/lightbox_gpu_client, 2023, accessed: September 4, 2024.

[26] O. Owolabi, "Lightbox simulation code," https://github.com/owolabioromidayo/lightbox_simulations, 2023, accessed: September 4, 2024.

[27] X. Wang, L. Xie, C. Dong, and Y. Shan, "Real-esrgan: Training real-world blind super-resolution with pure synthetic data," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 1905–1914.

[28] K. Higuchi, T. Mizuhashi, F. Matulic, and T. Igarashi, "Interactive generation of image variations for copy-paste data augmentation," in *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–7.

[29] E. Adetiba, M. Akanle, V. Akande, J. Badejo, V. P. Nzanzu, M. J. Molo, V. Oguntosin, O. Oshin, and E. Adebiyi, "Fedgen testbed: A federated genomics private cloud infrastructure for precision medicine and artificial intelligence research," in *International Conference on Informatics and Intelligent Applications*. Springer, 2021, pp. 78–91.