IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14412

AI-Powered Cloud Removal from Satellite Imagery

Manjula Devi K[1], Roohitha NS[2], C.Chithra Devi M.Sc., (Ph.D),[3]

B.Sc AI & ML, Dept. of Computer Science, Rathinam College of Arts and Science[1,2]

Assistant Professor, Dept. of Computer Science, Rathinam College of Arts and Science[3]

Abstract: Satellite images have kind of become part of everyday decision-making now. Farmers look at them to check crop health, governments use them during floods, and urban planners rely on them to see how cities are expanding.They're powerful tools, no doubt. But let's be real — they're not perfect. One issue that keeps getting in the way? Clouds. Even a thin layer can block important surface details, and once that happens, the image isn't nearly as useful as it should be. You're left guessing what's underneath, which defeats the whole purpose That's what pushed me toward this project. The goal wasn't to reinvent satellite imaging or anything dramatic. It was more practical than that: create a web-based system that can reduce cloud interference using AI. The idea is straightforward. A user uploads a satellite image through a simple web interface, and the system takes care of the rest.

The image is sent to the backend, where an AI-based process analyzes the cloudy regions and tries to reduce their impact without disturbing the actual land features I wanted the system to feel usable, not complicated. No heavy software installation, no confusing steps. Just upload, process, compare, and download.

The before-and-after comparison is actually one of my favorite parts because you can immediately see what changed. It makes the improvement feel real, not theoretical.At the end of the day, the purpose is pretty clear: make satellite images easier to work with. If analysts, researchers, or planners can see the ground more clearly, they can make better decisions. Sometimes, solving something as simple as cloud obstruction can quietly improve a lot of larger processes behind the scenes.

Keywords: satellite image processing, Cloud Removal, Web-based Application, Cloud Detection, Image enhancement, AI - Based Image Processing, React Web Application

I. INTRODUCTION

When I started working on the front end, I decided to stick with a component-based structure to keep everything organized from the start.

Each part of the interface has its own responsibility There is a simple image uploader and a section where users can compare the original image with the processed result.and a processing overlay that pops up so users know something is actually happening in the background

When someone uploads an image, it's converted into Base64 before being sent to the backend.it's really just a simple way to package the image properly so it can be transferred without any issues. However, on the user side, none of that complexity is visible. They just upload the file, wait a few seconds, and see the outcome. That smoothness was important to me — I didn't want people to wonder whether the app froze or crashed. The real cloud removal doesn't happen in the browser

I made that decision pretty early on. Instead of handling everything on the front end, I chose to move the processing to the backend using a Supabase Edge Function called removeclouds.That's where all the heavy lifting takes place. The function scans the image, spots the areas hidden by clouds, and then reduces their effect so the details underneath start to show more clearly Following enhancement, the reconstructed image is transmitted back to the frontend for display. Keeping that workload on the backend keeps the frontend responsive. It also means users don't need powerful systems just to use the app, which I think is a big plus.

Once the image is processed, the app displays the original and enhanced versions side by side. Honestly, that's probably the most satisfying part for me. You can immediately spot the difference — the cloudy patches fade, and more detail

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14412

becomes visible. There's also a download option, because what's the point if you can't actually use the improved image afterward? And if something goes wrong during processing, the system doesn't just stay silent. It throws a proper error message so the user knows what happened. At the end of the day, this project isn't trying to revolution- ize satellite imaging or solve every limitation out there.It sticks to one clear issue — clouds blocking the view — and solves it in a way that feels simple and easy to use.And honestly, watching a cloudy satellite image turn clear in just a few seconds

II. PROBLEM STATEMENT

Satellite images are used everywhere today. From checking crop health and tracking floods to planning cities and studying environmental change, they've quietly become part of how decisions get made. Governments rely on them. Researchers rely on them. Even private companies use them regularly. They're powerful tools, no doubt about that. But even with all the technology we have now, there's still one stubborn issue that keeps getting in the way — clouds.

clouds can be surprisingly disruptive. A single thick patch can hide the exact location someone needs to examine. Sometimes it's just a small area that's covered. Other times, it blocks almost everything important. And in situations like disaster monitoring or crop analysis, timing really matters. Waiting days or weeks for another satellite image isn't always realistic. And to make it worse, the next image might still have cloud cover. That means lost time, incomplete data, and a lot of frustration.

There are tools out there that attempt to fix this problem, but many of them aren't exactly beginner-friendly. Some require heavy software installations. Others demand strong hardware or technical knowledge that not everyone has. For students, small organizations, or anyone without high-end systems, these solutions can feel out of reach. Even trying to run heavy image processing on a regular laptop can slow everything down or make the system lag

That's when it becomes clear that things don't have to be this complicated. There should be a more straightforward solution — something people can use without spending hours setting it up or learning complex tools. Ideally, it should just work through a web platform where a user can upload a satellite image, let the system process it, and receive a clearer result within seconds. No heavy installations. No unnecessary complexity. Just a straightforward solution to a very real and very common problem.

III. LITERATURE REVIEW

A. Traditional Cloud Removal Techniques

Satellite images are extremely helpful for things like checking crop conditions, studying forests, or even assessing flood damage. But clouds? They really get in the way. In tropical and monsoon-heavy regions, it's actually common to get images where large portions are completely covered. That makes the data frustrating to use because you're basically missing important pieces of the ground information.

To deal with this, earlier methods kept things pretty simple. Cloud masking just identifies cloudy pixels and removes them from analysis quick and practical, but it doesn't restore what's hidden underneath. Another approach, called multi-temporal compositing, combines images taken at different times and replaces cloudy parts with clearer ones from another date. It works when clean images are available, but if clouds stick around for days or weeks, there's not much to merge. So while these traditional methods are useful, they don't fully solve the problem.

B. Image Interpolation and Reconstruction Methods

When cloud masking alone doesn't cut it, interpolation steps in, trying to fill the missing parts by looking at the pixels around them—kind of like piecing together a puzzle from the edges the thing is, these methods don't really "get" the image. They just work with averages or patterns, so the filled-in spots can end up smooth but kind of blurry. For quick visuals, that's fine—but if you need real accuracy, like for mapping rivers or cities, it usually falls short.

The thing is, these methods don't really "get" the image. They just work with averages or patterns, so the filled-in spots can end up smooth but kind of blurry. For quick visuals, that's fine—but if you need real accuracy, like for mapping rivers or cities, it usually falls short.

C. Deep Learning-Based Image Restoration Models

Even trying to run these deep learning models—like CNNs or GANs—on a regular laptop can slow everything down or make the system struggle to keep up. They do amazing stuff for cloud removal, but all that processing and data just isn't something a standard personal device can handle smoothly.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14412

Then there are GANs—Generative Adversarial Networks—which take things a step further. They've got this extra "discriminator" that basically judges if the reconstructed image looks real, which makes the results sharper and more natural. The catch? Training them takes a ton of data and serious computing power, so you definitely can't just run them on a regular laptop and expect instant results.

D. Web-Based AI Deployment

Earlier, working with satellite images usually meant relying on bulky desktop software or research models that rarely functioned outside lab setups. But that's changed quite a bit over time. With cloud computing becoming more accessible, AI models for satellite image processing are now being deployed through web platforms.

Rather than installing heavy software, users can just open a browser, upload an image, and see the results within seconds. Behind the scenes, the frontend manages the upload and display, while the backend runs the model often using serverless functions and sends the processed image back.In the earlier days, satellite image processing usually meant working with bulky desktop software or research systems that rarely moved beyond the lab

E. Research Gap and Motivation

Even though cloud removal methods have improved a lot, many of them still require high computational power and large datasets.A lot of these models also stay limited to research environments, without simple platforms where users can actually apply them easily. So in practice, there's still a gap between strong research results and real-world usability.

This project tries to close that gap by bringing AI-based cloud removal into a simple web-based system. It's not just about improving reconstruction quality, but also about making the whole solution practical and easy to use for environmental monitoring

IV. METHODOLOGY

This project takes a pretty common frustration clouds blocking satellite images and makes it easier to deal with. By pairing deep learning with a simple web setup, it's not just about clearer images, but about making the whole thing actually usable.

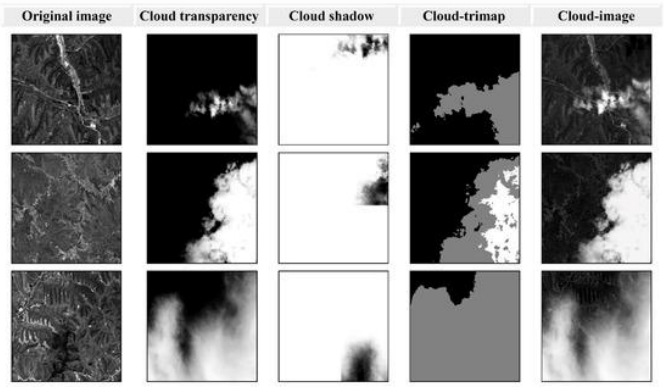

Figure 1: Image processing

A. Image Preprocessing

Before the model can do anything useful, the images need a little prep work. Since they come in different sizes and formats, the first step is resizing them so everything matches what the model expects.The pixel values are then normalized to keep training stable otherwise things can get messy pretty quickly. Some light noise reduction is also done to remove unwanted distortions, and the images are converted into a consistent color format. It's not the most exciting part of the project. this step makes a big difference. Cleaner input usually means better cloud detection and more accurate reconstruction

B. Data Collection and Preparation

The images for this project come from publicly available remote sensing datasets. Since not every satellite image looks the same, the dataset includes a good mix — clear skies, partial clouds, and heavily clouded scenes. That variety really helps the model learn properly instead of getting too comfortable with just one pattern.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14412

This project takes a pretty common problem — clouds hiding useful satellite data — and makes it manageable. With deep learning doing the heavy lifting and a simple web interface handling access, it's not just about cleaner images, but about making them genuinely usable.

C. Model Development

For the core of the system, a deep learning model is built to spot and separate cloud-covered areas from the rest of the satellite image. A convolutional neural network is used since it's pretty good at picking up spatial patterns things like brightness differences, soft cloud textures, and subtle edge variations that usually give clouds away.

The model is trained on the prepared dataset so it can generate accurate cloud masks highlighting the affected regions. During training, adjustments are made to improve stability and make sure it doesn't struggle when lighting changes or when the landscape looks completely different.

D. Cloud Removal and Image Reconstruction

After the clouds are detected, the system moves on to restoring what's hidden underneath. The system uses AI to intelligently fill those gaps, relying on nearby details so everything blends in smoothly. The idea is to keep the image looking smooth and consistent — whether it's trees, rivers, land textures, or buildings — without obvious edits or weird marks If done right, you shouldn't really notice where the clouds were in the first place.

E. System Integration and Real-Time Processing

Once the model was ready, the next step was making it actually usable. So the model was built into a simple web system where the frontend and backend handle their own roles. Users simply upload their image and view the processed result — it's clean, simple, and easy to use Most of the actual work happens in the backend. It stores the uploaded images, gets them ready for processing, runs the model, and sends back the enhanced version.

V. SYSTEM ARCHITECTURE

The system is basically a client-side web app built with React, and honestly, everything happens right inside the browser. There's no constant page reloading or anything like that it just updates the content smoothly, kind of like how modern apps are expected to behave now .The idea is pretty simple keep things clean and fast. Breaking the interface into small reusable components actually makes it easier to manage and tweak later on. There's no heavy backend work happening right now,
so the focus is mainly on getting a smooth and responsive front end in place.

A. User Interaction and Interface Handling

Users can simply open the app in their browser. There's nothing extra to install or set up, which makes it pretty straightforward to use . It all begins with index.html, which loads the basics and gets things going. After that, any click or input is handled on the fly, and the screen updates instantly.

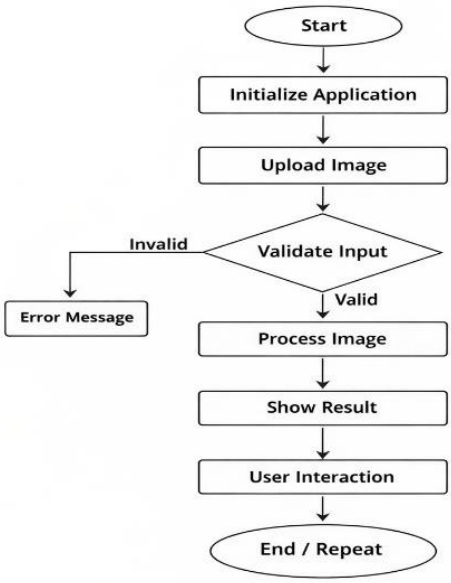

Figure 1 : Work flow for frontend

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14412

B. Application Initialization and Control

The app basically starts from main.jsx. That's where everything loads up, and App.jsx gets rendered onto the page.You can think of it as the point where the HTML and React sides finally come together. After that, App.jsx handles the layout and pulls everything into place so the interface looks complete. Users just open the app in their browser, and it works. No installs, no setup—nothing to worry about.

C. Component-Based Rendering

The system is broken down into smaller, reusable components, all organized inside the components/ folder. Each component takes care of a specific part of the interface, which keeps things from getting too cluttered in one place. Instead of dealing with a huge block of code, everything is split into manageable pieces, making it easier to understand and work with. This setup also makes updates a lot less painful. If something needs to be changed, you can just work on that particular component without affecting the rest of the app. Over time, all these small pieces come together to form the complete interface, giving the application its final look and feel.

D. Dynamic Content Management

React makes life easier, but it's not magic. You don't have to manually update everything state tracks changes, and the interface updates quietly in the background. Hooks and lifecycle features just do their job, keeping things in sync without you even thinking about it. Hooks and lifecycle features just do their thing, so the app stays in sync without you having to think about it. No need to refresh or reload—everything just updates as the user interacts. Because of this, everything feels fast and effortless, almost like it's just happening on its own. Updates happen almost instantly, so the app feels more interactive. You might not notice it right away, but once you use it, the difference is clear compared to old-style page reloads.

E. Configuration and Execution

Getting the app up and running is actually pretty easy, though it's not some kind of magic. Most of the setup lives in the package.json file—it's basically the app's personal checklist, keeping track of all the dependencies and scripts it needs to work. Anything specific to your own setup, like API keys or local paths, goes into the .env file. That way, you can switch machines or environments without touching the main code at all, which honestly makes life a lot easier. That way, you can switch machines or environments without messing with the main code at all. The thing about that is you can switch setups without touching the main code at all, which saves a lot of headaches.

it runs on a local Node.js server. And honestly, that part is kind of fun—you tweak something in the code, hit save, and boom, it's live in the browser. Everything updates quietly in the background, so you don't have to babysit it or worry about breaking things. It just feels... smooth. Really satisfying, actually, to watch your changes take effect instantly

F. Workflow of the System

The workflow of the system is pretty straightforward once you break it down. It starts with the app loading, setting everything up behind the scenes so it's ready to go. From there, whenever a user interacts with it—clicking a button, entering data, or navigating through pages—the system picks up on those actions and processes them step by step. It's not something you really notice while using the app, but there's a lot happening quietly in the background to keep things responsive.

As those interactions come in, the relevant components update automatically, and the interface adjusts itself without needing a full reload. That's what makes everything feel smooth and dynamic. The workflow basically maps out how the app reacts to each action and keeps everything in sync, so the experience feels seamless rather than clunky or delayed.

VI. RESULT AND DISCUSSION

The system holds up pretty well when it comes to handling user interactions and updating things in real time. Once the app loads, everything kind of settles into place in the background, and from there it just feels smooth to use. Clicking buttons, typing inputs, switching between sections—it all happens without that annoying delay you sometimes expect. What really stands out is how the interface updates. It doesn't reload the whole page or anything dramatic; it just updates the parts that need to change, and that alone makes a big difference. During testing, even small code tweaks showed up instantly in the browser, which honestly made the whole development process feel a lot less frustrating and way more interactive.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14412

Figure 3: Before and After Output

If you look a bit deeper, the workflow is doing a lot of heavy lifting. From the outside, everything feels simple, but there's actually a clear flow happening behind the scenes. A user action comes in, the system processes it, updates the right components, and then refreshes the interface. It all happens so fast you barely notice it, but that's what keeps everything consistent. The local Node.js server really helps here too—it shows changes instantly, so testing and debugging feel more like a live process instead of stopping and checking every time. It's easy to overlook at first, but once you get used to it, you really don't want to go back. Of course, it's not completely perfect. As the app grows and gets more complex, things can get a bit tricky. Managing multiple components and frequent updates isn't always easy, and if you're not careful, you might run into unnecessary re-renders or small slowdowns. It's not a huge problem, but it's definitely something to keep in mind. That's where some thoughtful design really matters—keeping components modular, managing state properly, and just being a bit mindful about how updates are handled. When those things are done right, the system still runs pretty smoothly without much trouble.

Overall, the system does what it's supposed to do—and it does it well. It feels responsive, consistent, and easy to work with from both the user's and developer's side. From a user's point of view, everything just works without needing to think about what's going on underneath. And for developers, it's a comfortable setup—you can build, test, and tweak things without constantly running into issues. It's not flashy, but it's reliable, and honestly, that's what matters most.

VII. CONCLUSION

The system works pretty well and feels smooth to use. The workflow is easy to follow, and everything—from uploading an image to seeing the final result—just fits together in a way that makes sense. It never really feels confusing or overloaded, which honestly makes a big difference when you're using it.The app is also quite responsive. Most actions happen almost instantly, so you barely notice any lag. That quick feedback makes it feel more interactive, like the system is keeping up with you instead of slowing you down. Another thing I really liked is how easy it is to work with. The structure is clean, so you don't have to spend time figuring out where things are or how they work. Whether you're just using it or building on top of it As the system grows, performance might need a bit more attention, especially when things get heavier. Still, it's a solid setup—reliable, simple, and something you can build on without too much trouble.

VIII. FUTURE WORK

A. Improve Processing Accuracy

There's definitely scope to make the processing more advanced. Right now it works, but improving the underlying model or logic would give more accurate and reliable results, especially for complex images

B. Performance Optimization

As the system grows, performance might need a bit more attention. Handling larger data and reducing unnecessary updates can help keep everything running smoothly without slowdowns.

C. UI Enhancements

The UI is already simple, but it can be improved further. Adding features like zoom, sliders, or better visual comparisons would make the results easier to understand and more interactive.

D. Scalability and Deployment

Currently, the system runs locally, but deploying it as a full web application would make it more useful. Supporting multiple users and real-time access would take it closer to a real-world solution.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14412

E. Backend Integration

Connecting the system to a proper backend would make a big difference. Right now, most of the work is on the frontend, but adding server-side processing and a database would make it more complete. It would also help in storing images, managing data, and handling more complex operations more efficiently.

REFERENCES

[1]. Meta Platforms, Inc., "React: A JavaScript library for building user interfaces," 2025. [Online]. Available: https://react.dev/
[2]. Open JS Foundation, "Node.js: JavaScript runtime built on Chrome's V8 engine," 2025. [Online]. Available: https://nodejs.org/
[3]. Express.js, "Fast, unopinionated, minimalist web framework for Node.js," 2025. [Online]. Available: https://expressjs.com/
[4]. G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
[5]. M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX OSDI*, 2016, pp. 265–283.
[6]. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018.
[7]. A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep CNNs," *Communications of the ACM*, 2017.
[8]. K. He et al., "Deep residual learning for image recognition," in *Proc. CVPR*, 2016.
[9]. I. Goodfellow et al., "Generative adversarial nets," in *Proc. NIPS*, 2014.
[10]. J. Redmon et al., "You Only Look Once: Unified, real-time object detection," in *Proc. CVPR*, 2016.
[11]. S. Ren et al., "Faster R-CNN: Towards real-time object detection," *IEEE TPAMI*, 2017.
[12]. X. Zhu et al., "Deep learning in remote sensing: A review," *IEEE Geoscience and Remote Sensing Magazine*, 2017.
[13]. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 2015.
[14]. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2015.
[15]. T. O'Shea and J. Hoydis, "An introduction to deep learning for wireless communications," *IEEE Transactions*, 2017.
[16]. Q. Zhang et al., "Cloud detection in satellite imagery using deep learning," *Remote Sensing*, 2019.
[17]. Z. Li et al., "A survey on cloud removal techniques in satellite images," *IEEE Access*, 2020.
[18]. H. Shen et al., "Cloud removal for remote sensing images using GANs," *IEEE JSTARS*, 2019.
[19]. J. Long et al., "Fully convolutional networks for semantic segmentation," in *Proc. CVPR*, 2015.
[20]. O. Ronneberger et al., "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. MICCAI*, 2015.
[21]. Supabase Inc., "Supabase: Open-source backend as a service," 2025. [Online]. Available: https://supabase.com/
[22]. Mozilla Foundation, "JavaScript Guide," 2025. [Online]. Available: https://developer.mozilla.org/
[23]. WHATWG, "HTML Living Standard," 2025. [Online]. Available: https://html.spec.whatwg.org/
[24]. W3C, "CSS Specifications," 2025. [Online]. Available: https://www.w3.org/Style/CSS/
[25]. NPM Inc., "Node Package Manager (npm) Documentation," 2025. [Online]. Available: https://www.npmjs.com/