A CONTENT-BASED AUTOMATIC FILTER RECOMMENDATION SYSTEM FOR PHOTOGRAPHY AND IMAGE EDITING USING ARTIFICIAL INTELLIGENCE

Yifan Luo[1], Yu Sun[2]

[1]Eden High School, 535 Lake Street, St Catharines, Ontario L2N 4H7, Canada
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

This paper presents the development and evaluation of PhotoGleam, a mobile application designed to enhance image quality through AI-generated filters.The app addresses the common issue of aesthetically unappealing colors in mobile phone photography by leveraging advanced AI models to create custom filters tailored to each image. The paper details the implementation of the AI-driven image processing engine, the backend Flask server, and the Flutter frontend user interface. Through two key experiments, the effectiveness of AI-generated filters was assessed in terms of both image quality and user engagement. Results showed that users consistently preferred AI-generated filters over standard ones, and the introduction of these filters led to increased time spent on the app and more images edited. While there are areas for improvement, such as server scalability and AI model reliance, PhotoGleam demonstrates significant potential as a valuable tool for enhancing mobile photography.

KEYWORDS

AI-generated filters, Mobile photography, Image enhancement, Photo editing app, Custom filters

1. INTRODUCTION

In today's society, more and more people across the world are using mobile phones on a daily basis. Mobile phones' functions have long expanded from just communication, and one important aspect of mobile phones is their cameras. Users only need to press one single button to capture an image. This, compared to the heavyweight and complicated design of digital single-lens reflex cameras (DSLRs) and mirrorless cameras, has naturally driven users towards phone cameras. The convenience and intuitive nature of phone cameras has given everyone, regardless of their experience in photography, the ability to capture images [1].

The most common issue regarding phone cameras is their image quality. Many question the final image quality of a phone camera, stating that the images are less sharp and less aesthetically pleasing. While phone cameras might indeed produce poor-quality images in the past, this is not the case today. Today, phone cameras have developed significantly from the past and are more than capable of taking stunning images.

To understand why phone cameras today are mostly on par with some professional cameras, we must understand how an image is taken. An image is created as light falls on a photosensitive surface. This surface is an image sensor such as a complementary metal-oxide-semiconductor (CMOS) or a charge-coupled device (CCD) [2]. Other components of a camera include the lens, the shutter, the aperture, and the memory; however, these components are not nearly as important as the image sensor. The major difference between the image sensor of a mobile phone and a professional camera is its size, as opposed to the pixel count as many people tend to believe. In the past, the sensor sizes of mobile phones are small, with the iPhone 6 having ⅓" [3]. Nowadays, however, phone manufacturers, such as Xiaomi and Huawei can fit 1" image sensors on mobile phones [4]. This is the same size as the camera sensor on the SONY RX100 VII camera, selling at $1299 [5]. The difference in image quality between phones and cameras is not significant enough to be noticed on small screens such as those on mobile phones.

So what are the causes of the difference regarding the image of mobile phones and professional cameras? Why do images taken with professional equipment always seem "better" than phone images? The secret lies in the colour rather than the equipment. Professional photographers often spend large amounts of time processing an image after taking in image editing software such as Lightroom Classic or Photoshop. This ensures that the color of the image is pleasing as this will be the first thing people notice when viewing an image. Mobile phone users do not do the same. They rely on the basic processing by the image signal processor (ISP) in the smartphone which does not produce remarkable results regarding the color, making their images look grey and unappealing [6]. Therefore, the problem with images taken by mobile phones does not lie in the image quality, but rather in the unappealing colors caused by the lack of image editing and processing by the user.

To solve the issues mentioned within this paper, we propose the PhotoGleam mobile application as a solution to the lack of convenient and intuitive image processing tools for mobile phone users. The PhotoGleam application uses artificial intelligence (AI) to generate a list of possible filters, custom-curated for individual photos itself.

There have been many studies exploring methodologies to enhance image quality and produce aesthetically pleasing colors. In [7],Orhei and Vasiu (2022) attempted traditional sharpening filters. In [8], Moran et al. (2020) tried spatially localized enhancements using deep neural networks. And in [9], Pan et al. (2021)presented a novel method involvingGAN-based improvements for mobile photography. However, each approach has its limitations: Orhei and Vasiu's method [7] relies on outdated filter-based techniques, Moran et al.'s DeepLPF [8] requires significant computational resources, and Pan et al.'s MIEGAN [9] is constrained by its high demand for processing power, making them less practical for real-time use on mobile devices.PhotoGleamaddresses these shortcomings by integrating AI-driven custom filters that are optimized for mobile platforms, providing real-time, high-quality image enhancements that are both efficient and user-friendly.

The user can take a new photo within the application, or upload a photo from the phone's image gallery. Once users decide upon a filter they like, PhotoGleam will simply apply the filter to the image and make the filtered image available for download. If the user does not like any of the filter recommendations, then the user can request more filters until a desired result is reached. The proposed application generates image filters completely automatically without asking for any additional inputs from the user. All the user needs to do is to press the shutter. Then, a list of fully edited and processed images will be available for the user, making the proposed application an effective solution when compared to conventional image editing software.

Additionally, when compared to other image editing software within social media applications such as Snapchat which only have a select list of filters, the filters in PhotoGleam are entirely custom generated by the AI model for that specific image. Not only does this mean that the images will be unique, but it also implies that the potential filter generation variations are limitless.

To evaluate the effectiveness of AI-generated filters in the completed app, two experiments were conducted. The first experiment focused on testing filter accuracy and user satisfaction by having 15 participants compare standard filters with AI-generated ones. The results showed a clear preference for AI-generated filters, with higher ratings in categories like visual appeal, color balance, and perceived quality. The second experiment measured user engagement by tracking the time spent on the app and the number of images edited. Participants using AI-generated filters spent significantly more time editing images and edited more images overall, indicating that these filters not only improve image quality but also enhance the overall user experience.

Both experiments provided valuable insights into the impact of AI on mobile photography, confirming that AI-generated filters can offer significant improvements over standard options, leading to greater user satisfaction and engagement.
This paper is arranged as follows: Section 2 reviews the challenges encountered in building filter recommendation system. Section 3 includes the technical aspects and structure of the implemented app. Section 4 explains and analyzes the two experiments conducted. Section 5 lists and discussesrelated works to my filter system. Section 6 gives a final evaluation of the project.

2. CHALLENGES

During the process of researching, designing, and implementing the automatic filter recommendation system, the following challenges were identified.

2.1. Image Recognition

Image recognition is an essential part of PhotoGleam, and there were many different ideas and concepts for the implementation of this section. We decided to train an image recognition model at first; however, these models were not as effective as we hoped. We believe the cause of this issue is the small sample size of the images we used to train our model. This is because, contrary to popular beliefs, the sample size is the most crucial factor in training machine learning models. In fact, the sample size is much more important than the model used for training [10]. A large dataset with more than 10000 images is now gathered and will be used to train a stronger model, which will be implemented once ready. In the meantime, we propose the use of Gemini for image recognition since it is a pre-trained model that is not only strong but also easy to use.

2.2. Backend Server

The implementation of the backend server also proved to be challenging. PhotoGleam utilizes a Flask web framework written in Python. The backend server was originally planned to be deployed and hosted on Render. However, the server's deployment on Render failed due to many reasons. To mitigate this, we decided to use Amazon Web Services (AWS) to deploy the server instead, which proved to be successful. Despite the successful deployment, many problems still remain. It is important to ensure that the server can process and return AI-generated filters in a timely manner, without causing major delays or timeouts that would disrupt user experience. To resolve this issue, we could attempt to optimize the server's performance by implementing load balancing and caching mechanisms. Scalability is also a major issue. As the server gains popularity and user demands increase, it is crucial that the server's architecture is scalable.

2.3. Flutter Frontend UI

The Flutter frontend user interface (UI) design of PhotoGleam is essential. In the frontend, it is important to balance maintaining a smooth and intuitive user experience with integrating complex image processing functionalities. We believe a great design should be simple, clear, and elegant, there should be no manual or tutorial needed for PhotoGleam. Because of this, PhotoGleam aims to have a UI that is both intuitive and visually appealing so that the user can navigate through it with ease and pleasure. Existing issues on the frontend include lag in UI responsiveness due to the heavy processing load and the complexity of providing real-time filter previews. Optimizations such as efficient state management and leveraging asynchronous processing can be considered to alleviate these issues.

3. Solution

The PhotoGleam application consists of three major components: the Flutter-based frontend UI, the Flask backend server, and the AI image processing engine. The application flow begins when the user captures or uploads an image from the mobile phone, after which the image is sent to the Flask server immediately. In the Flask server, an AI image processing engine is utilized to generate and apply the custom filter. More specifically, an API call to Google's Gemini AI is performed, in which Gemini analyzes the image, considering various factors such as lighting, color composition, and subject matter [11]. Then, Gemini returns a list of image adjustments back to the Flask server, where these suggestions are applied utilizing Pillow, a Python Imaging Library (PIL) [12]. Finally, the edited images are returned to the application, where the user can preview them by scrolling left or right, and download the desired edited image to their device's local storage. If the user is still unsatisfied, then the image can be uploaded again and a new list of filters will be generated again with the press of a button. Figure 1 gives the flowchart of PhotoGleam, highlighting the interactions between each major component.

The frontend application is built using Flutter for cross-platform compatibility, while the backend relies on Flask to handle requests, process images, and communicate with Gemini. The Flask server is hosted using AWS, more specifically an Amazon EC2 instance. Dart is the language used to program the Flutter frontend, and Python is the language used to program and server backend. API calls and prompt engineering were also essential in using Gemini to analyze images and give filter recommendations.

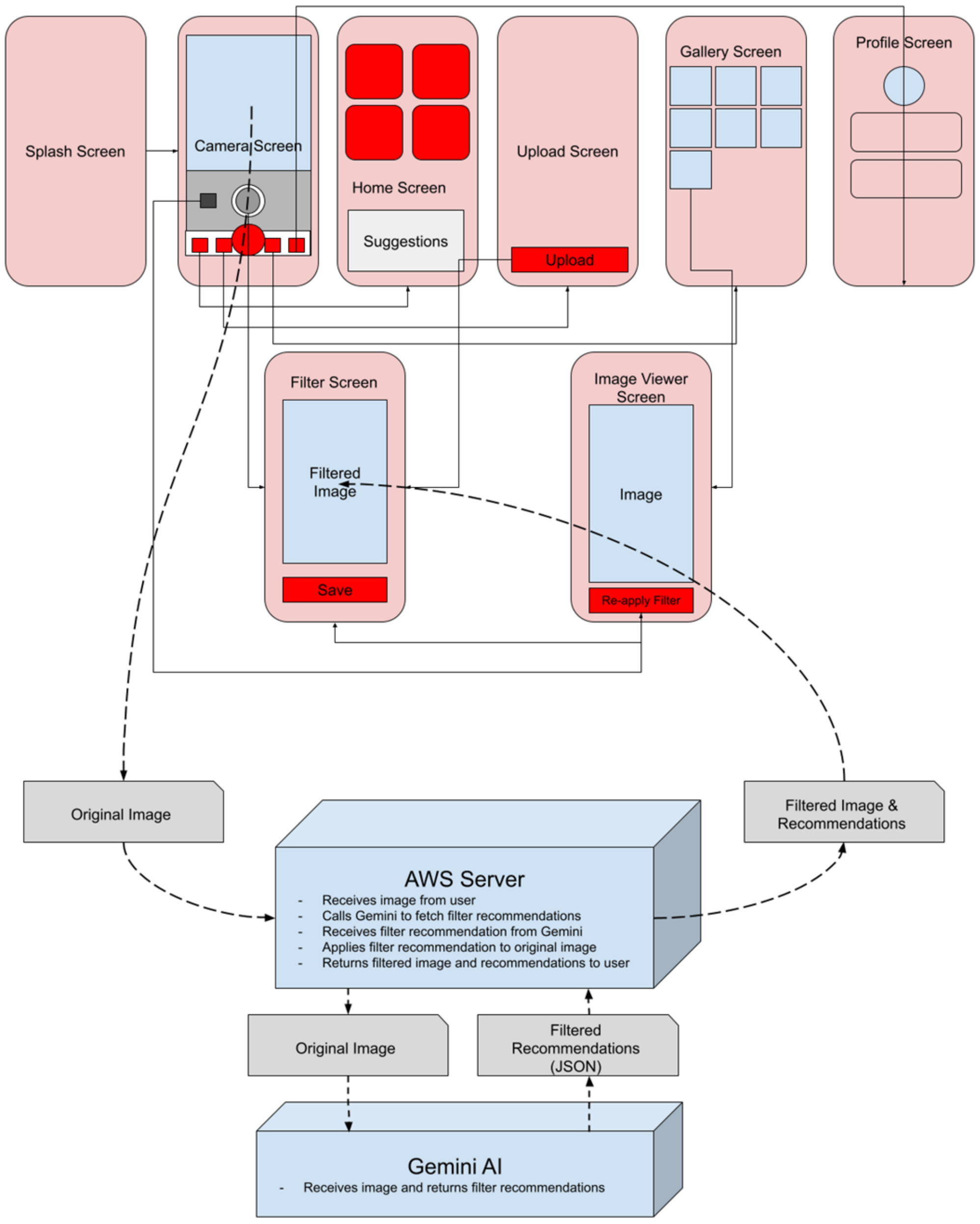
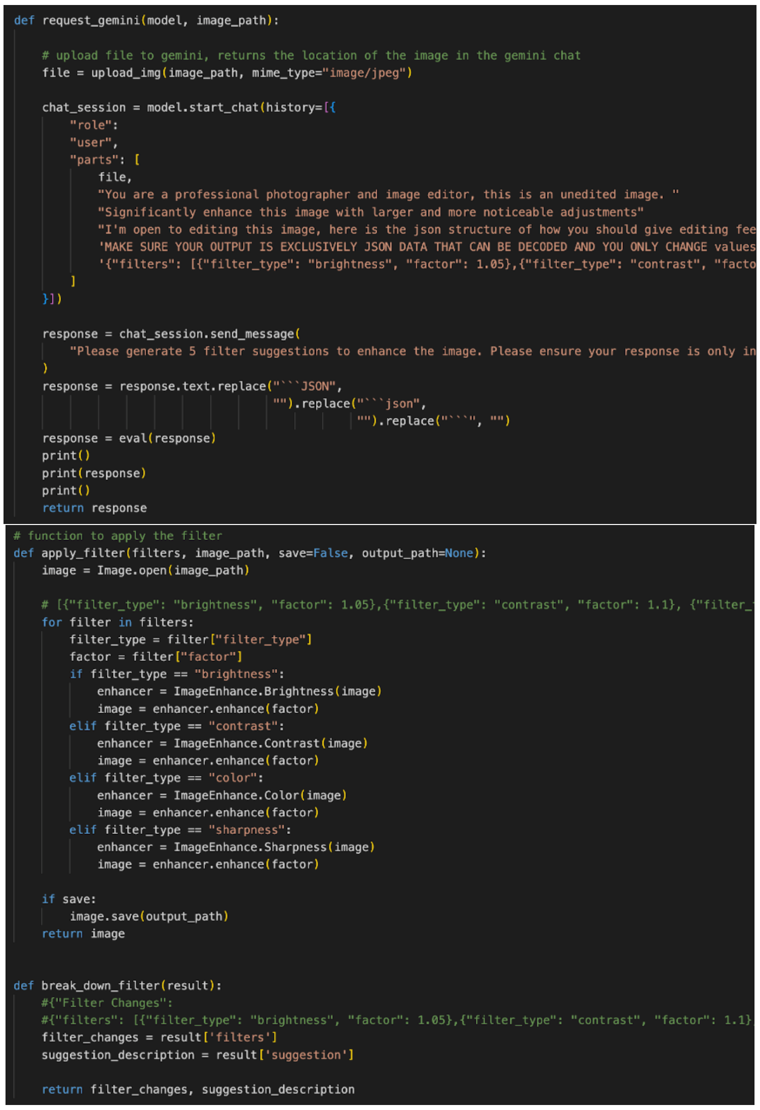

Splash Screen
Camera Screen
Home Screen
Suggestions
Upload Screen
Upload
Gallery Screen
Profile Screen
Filter Screen
Filtered Image
Save
Image Viewer Screen
Image
Re-apply Filter
Original Image
Filtered Image & Recommendations
AWS Server
- Receives image from user
- Calls Gemini to fetch filter recommendations
- Receives filter recommendation from Gemini
- Applies filter recommendation to original image
- Returns filtered image and recommendations to user
Original Image
Filtered Recommendations (JSON)
Gemini AI
- Receives image and returns filter recommendations

Figure 1. Overview of the solution

3.1. AI Image Processing Engine

The AI-driven image processing engine is a core component of PhotoGleam that generates and applies custom filters to images uploaded by users. This component leverages Google's Gemini AI to analyze the image for suggestions and utilizes the Pillow PIL to apply the suggested enhancements. This makes it possible to deliver unique and tailored filters to the user.



````
def request_gemini(model, image_path):

    # upload file to gemini, returns the location of the image in the gemini chat
    file = upload_img(image_path, mime_type="image/jpeg")

    chat_session = model.start_chat(history=[{
        "role":
        "user",
        "parts": [
            file,
            "You are a professional photographer and image editor, this is an unedited image. "
            "Significantly enhance this image with larger and more noticeable adjustments"
            "I'm open to editing this image, here is the json structure of how you should give editing fee
            'MAKE SURE YOUR OUTPUT IS EXCLUSIVELY JSON DATA THAT CAN BE DECODED AND YOU ONLY CHANGE values
            '{"filters": [{"filter_type": "brightness", "factor": 1.05},{"filter_type": "contrast", "facto
        ]
    }])

    response = chat_session.send_message(
        "Please generate 5 filter suggestions to enhance the image. Please ensure your response is only in
    )
    response = response.text.replace("```JSON",
                                     "").replace("```json",
                                                 "").replace("```", "")
    response = eval(response)
    print()
    print(response)
    print()
    return response
````

````
# function to apply the filter
def apply_filter(filters, image_path, save=False, output_path=None):
    image = Image.open(image_path)

    # [{"filter_type": "brightness", "factor": 1.05},{"filter_type": "contrast", "factor": 1.1}, {"filter_
    for filter in filters:
        filter_type = filter["filter_type"]
        factor = filter["factor"]
        if filter_type == "brightness":
            enhancer = ImageEnhance.Brightness(image)
            image = enhancer.enhance(factor)
        elif filter_type == "contrast":
            enhancer = ImageEnhance.Contrast(image)
            image = enhancer.enhance(factor)
        elif filter_type == "color":
            enhancer = ImageEnhance.Color(image)
            image = enhancer.enhance(factor)
        elif filter_type == "sharpness":
            enhancer = ImageEnhance.Sharpness(image)
            image = enhancer.enhance(factor)

    if save:
        image.save(output_path)
    return image


def break_down_filter(result):
    #{"Filter Changes":
    #{"filters": [{"filter_type": "brightness", "factor": 1.05},{"filter_type": "contrast", "factor": 1.1}
    filter_changes = result['filters']
    suggestion_description = result['suggestion']

    return filter_changes, suggestion_description
````

Figure 2. Screenshot of AI image processing engine code

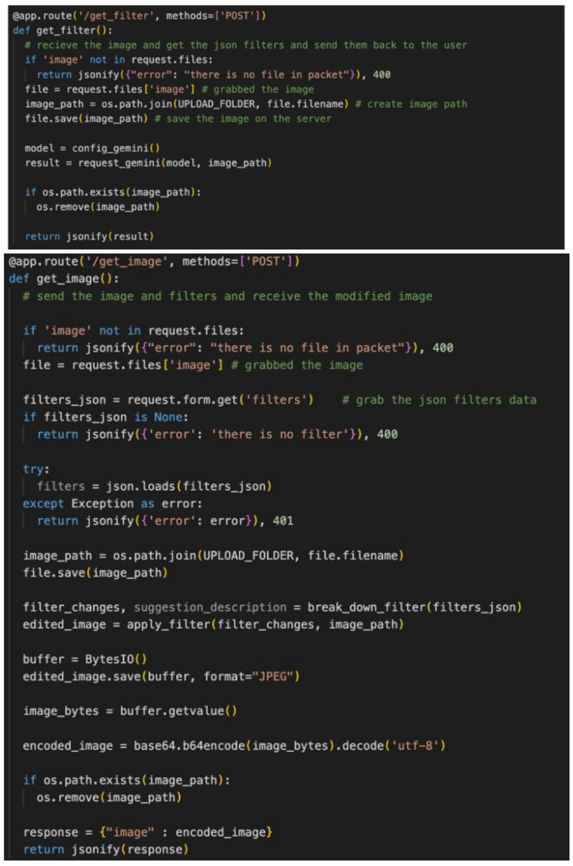
The provided code samples in Figure 2 illustrate the implementation of the AI-driven image processing engine. The image processing engine contains two major parts: Google's Gemini AI for filter recommendation and generation, and Pillow PIL to apply the suggested filters by Gemini. This modular design ensures that the image processing engine is both flexible and powerful, capable of delivering high-quality image enhancements efficiently.

The first code snippet illustrates the API call to Gemini for filter recommendation, which is the first major part of the image processing engine. In the code sample, the request_gemini function takes in two parameters, the image, and the AI model, and sends an API request to Gemini. In the API request, a pre-determined prompt is included alongside the image asking for filter recommendations in terms of brightness, contrast, color, and sharpness. The return format is specified to be in JSON.
The second code snippet illustrates the breakdown and application of the response from Gemini by Pillow. After the JSON data is received back from Gemini, the break_down_filter function is called taking in the returned data as the parameter [13]. This function breaks down the response into filters and suggestions. Then, the apply_filter function takes in the filters broken down in the

break_down_filter function and the original image as parameters, and applies the filters onto the original image utilizing Pillow. The recommendations in brightness, contrast, color, and sharpness are reflected in the filtered image and returned.

3.2. Flask Backend Server

The backend Flask server manages interactions between the Flutter frontend UI and the AI-driven image processing engine. It handles image uploads, communicates with the AI model to request filter suggestions, utilizes Pillow to apply these filter suggestions to the original image, and returns the filtered images back to the user. The Flask server ensures the smooth and secure processing of user images.



```python
@app.route('/get_filter', methods=['POST'])
def get_filter():
  # recieve the image and get the json filters and send them back to the user
  if 'image' not in request.files:
    return jsonify({"error": "there is no file in packet"}), 400
  file = request.files['image'] # grabbed the image
  image_path = os.path.join(UPLOAD_FOLDER, file.filename) # create image path
  file.save(image_path) # save the image on the server

  model = config_gemini()
  result = request_gemini(model, image_path)

  if os.path.exists(image_path):
    os.remove(image_path)

  return jsonify(result)
```

```python
@app.route('/get_image', methods=['POST'])
def get_image():
  # send the image and filters and receive the modified image

  if 'image' not in request.files:
    return jsonify({"error": "there is no file in packet"}), 400
  file = request.files['image'] # grabbed the image

  filters_json = request.form.get('filters')    # grab the json filters data
  if filters_json is None:
    return jsonify({'error': 'there is no filter'}), 400

  try:
    filters = json.loads(filters_json)
  except Exception as error:
    return jsonify({'error': error}), 401

  image_path = os.path.join(UPLOAD_FOLDER, file.filename)
  file.save(image_path)

  filter_changes, suggestion_description = break_down_filter(filters_json)
  edited_image = apply_filter(filter_changes, image_path)

  buffer = BytesIO()
  edited_image.save(buffer, format="JPEG")

  image_bytes = buffer.getvalue()

  encoded_image = base64.b64encode(image_bytes).decode('utf-8')

  if os.path.exists(image_path):
    os.remove(image_path)

  response = {"image" : encoded_image}
  return jsonify(response)
```

Figure 3. Screenshot of Flask backend server code

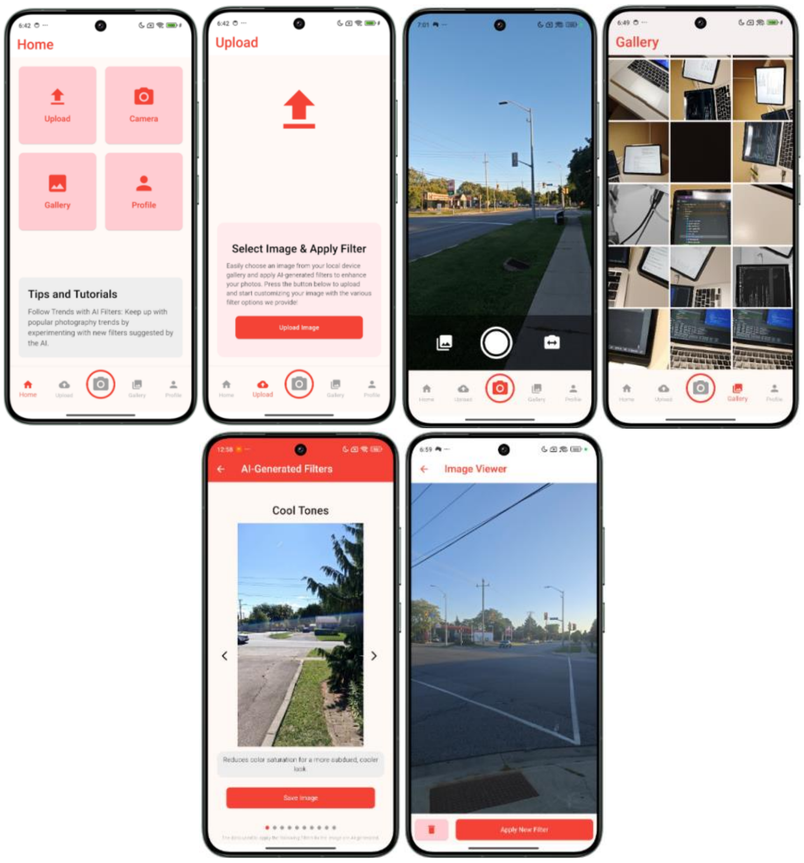
The provided code samples in Figure 3 illustrate the two key components of the Flask server used in PhotoGleam, which acts as a bridge facilitating the communication between the mobile application and the AI-driven image processing engine. The Flask server ensures that image processing requests are handled efficiently, providing a seamless experience for the user.

In the first code snippet, the get_filter route handles HTTP requests for image uploads. Upon receiving an image, the server saves it temporarily in the uploads directory. Then, the config_gemini and request_gemini functions call the image processing engine to initialize Gemini, generate custom filter data, and return them. The returned filters are sent back to the user in JSON format.

In the second code snippet, the get_image route handles HTTP requests for filter applications. After receiving the image and filter suggestions, the server saves the image temporarily and calls the break_down_filter function and the apply_filter function in the image processing engine. These functions apply the filter to the image utilizing Pillow in the image processing engine, and the filtered image is sent back to the user.

3.3. Flutter Frontend UI

The Flutter frontend UI is an essential component that allows users to take and upload images, display custom-generated filters for these images, and download these filtered images to their local devices. This component interacts with the backend server to send the original image, receive filter suggestions, and receive filtered images to display the final result in the mobile application to the user.



Tips and Tutorials

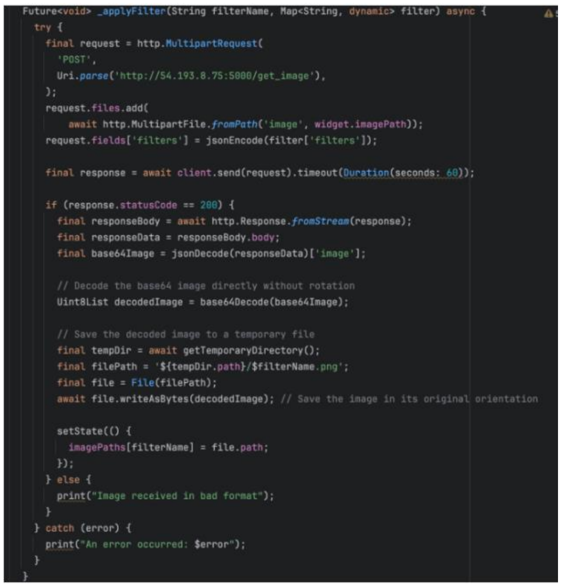
Follow Trends with AI Filters: Keep up with popular photography trends by experimenting with new filters suggested by the AI.

Select Image & Apply Filter

Easily choose an image from your local device gallery and apply AI-generated filters to enhance your photos. Press the button below to upload and start customizing your image with the various filter options we provide!

Reduces color saturation for a more subdued, cooler look

Figure 4. Screenshot of the APP



```
Future<void> _applyFilter(String filterName, Map<String, dynamic> filter) async {
  try {
    final request = http.MultipartRequest(
      'POST',
      Uri.parse('http://54.193.8.75:5000/get_image'),
    );
    request.files.add(
        await http.MultipartFile.fromPath('image', widget.imagePath));
    request.fields['filters'] = jsonEncode(filter['filters']);

    final response = await client.send(request).timeout(Duration(seconds: 60));

    if (response.statusCode == 200) {
      final responseBody = await http.Response.fromStream(response);
      final responseData = responseBody.body;
      final base64Image = jsonDecode(responseData)['image'];

      // Decode the base64 image directly without rotation
      Uint8List decodedImage = base64Decode(base64Image);

      // Save the decoded image to a temporary file
      final tempDir = await getTemporaryDirectory();
      final filePath = '${tempDir.path}/$filterName.png';
      final file = File(filePath);
      await file.writeAsBytes(decodedImage); // Save the image in its original orientation

      setState(() {
        imagePaths[filterName] = file.path;
      });
    } else {
      print("Image received in bad format");
    }
  } catch (error) {
    print("An error occurred: $error");
  }
}
```

Figure 5. Screenshot of Flutter code

The code sample provided in Figure 5 is within the filterScreen page and demonstrates the Flutter UI's functionality for applying filters to images within the PhotoGleam application. When the page is initialized, the _applyFilter() function is triggered, sending the image and selected filters to the backend server via an HTTP POST request.

The server processes the image with the filter recommendations and returns the filtered image, which is then displayed. This code efficiently handles the communication between the Flutter frontend and the Flask backend, ensuring that users can see the results of their filter selections in real-time. The use of http.MultipartRequest allows for easy file upload and JSON data handling, making the interaction between the UI and the server seamless and responsive.

4. EXPERIMENT

4.1. User Satisfaction Experiment

The goal of this experiment is to test the accuracy and effectiveness of the AI-generated filters in enhancing image quality and user satisfaction when compared to standard filter options.

To test the filter accuracy, we will conduct a blind comparison study involving 15 participants. Each participant will be presented with two sets of images: one set with standard filters and another set with AI-generated filters from PhotoGleam. The images will be displayed in a randomized order without any indication of which filter was used. Participants will be asked to rate each image out of 10 based on overall visual appeal, color balance, and perceived quality. The ratings will be collected and analyzed to determine if the AI-generated filters consistently outperform the standard ones in user preference.

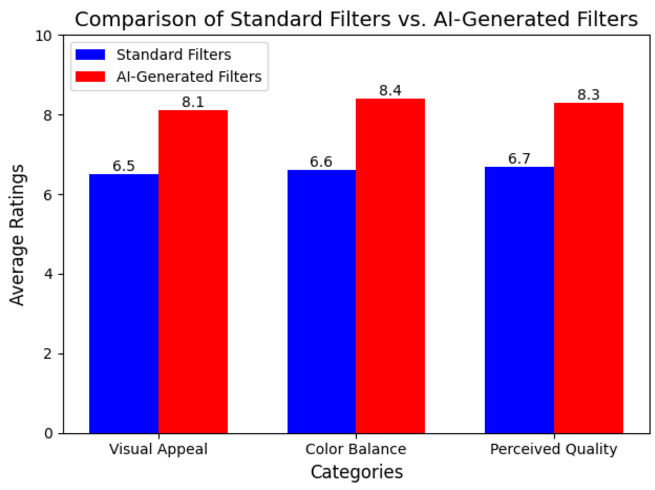


Figure 6. User satisfaction experiment result

The data from the experiment, visualized in Figure 6, indicates that AI-generated filters consistently received higher average ratings compared to standard filters across all categories: Visual Appeal, Color Balance, and Perceived Quality. The AI filters scored an average of 8.1 to 8.4, significantly outperforming the standard filters, which scored between 6.5 and 6.8.

This result suggests that the AI-driven approach effectively enhances image quality in ways that are both noticeable and appreciated by users. The consistency of higher ratings across all participants, despite the small sample size of 15, reinforces the validity of the findings.

Unexpected lower ratings for standard filters may be attributed to their generic nature, which fails to account for the specific characteristics of each image. These results validate the hypothesis that AI-generated filters offer a substantial improvement, making PhotoGleam a valuable tool for enhancing mobile photography. Any minor discrepancies will guide further refinement of the AI model.

4.2. User Engagement Experiment

This experiment aims to evaluate whether the introduction of AI-generated filters in PhotoGleam leads to increased user engagement, measured by the time spent on the app and the number of edited images.

To measure user engagement, we will track the app usage statistics of 15 participants over a two-week period. Participants will use the PhotoGleam app with both standard and AI-generated filters available. The experiment will monitor two key metrics: the average time each participant spends editing images per session and the total number of images edited. A comparison will be made between the usage of AI-generated filters and standard filters. By analyzing these metrics, we can determine if the AI-generated filters not only improve image quality but also enhance user engagement and overall app usage.

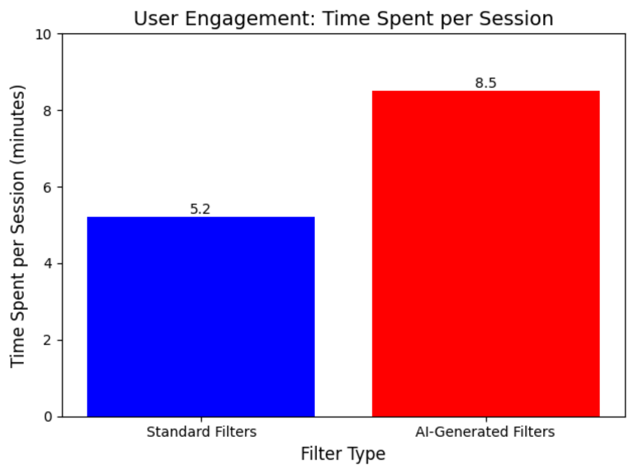


Figure 7. Time spent per session result

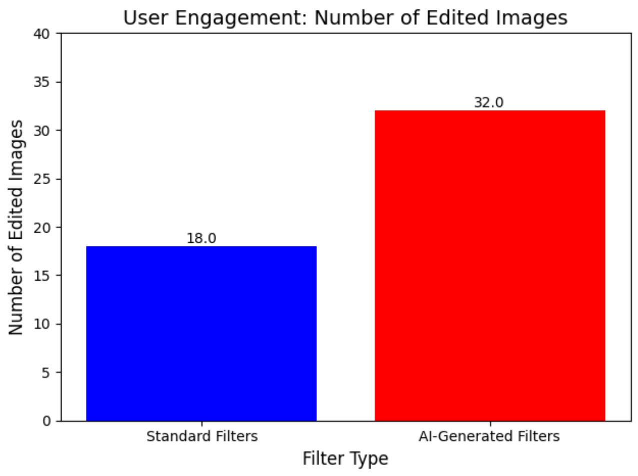


Figure 8. Number of edited images result

The data from the experiment, as visualized in Figure 7 and Figure 8, indicates a significant increase in user engagement when AI-generated filters were used in PhotoGleam. Participants spent an average of 8.5 minutes per session editing images with AI filters, compared to just 5.2 minutes with standard filters. Additionally, the number of images edited nearly doubled, from 18 images using standard filters to 32 images with AI-generated filters.

These results suggest that the introduction of AI-driven filters not only enhances the quality of the images but also significantly boosts user engagement. The increase in time spent and the number of images edited reflect greater user satisfaction and interest in the app's capabilities. This supports the hypothesis that AI-generated filters can make the photo editing experience more appealing, leading to more frequent and prolonged use of the app. These findings will be crucial in positioning PhotoGleam as a preferred tool for mobile photography.

5. Related Work

The study by Orhei and Vasiu (2022) focuses on enhancing image sharpness using dilated filters in digital photography [7]. The authors propose an enhancement of traditional sharpening algorithms, like High Pass Filter (HPF) and Unsharp Masking (UM), by incorporating dilated filters. These modified algorithms are shown to achieve better results both visually and statistically compared to classical methods. However, this approach is limited by its reliance on traditional filter-based techniques, which may not account for complex image characteristics. In contrast, PhotoGleam leverages AI to generate custom filters tailored to each image, providing more flexible and higher-quality enhancements.

Moran et al. (2020) introduced Deep Local Parametric Filters (DeepLPF) for automatic image enhancement [8]. Their approach uses deep neural networks to apply spatially localized filters, offering more refined adjustments compared to global enhancement techniques. While this method excels in providing interpretable and intuitive enhancements, it still requires significant computational resources and might be less accessible on mobile platforms. The PhotoGleam application improves on this by integrating AI-driven custom filters directly into a user-friendly mobile application, making advanced image processing more accessible to the average user.

Pan et al. (2021) developed MIEGAN, a generative adversarial network-based method for enhancing mobile photography images [9]. MIEGAN uses a multi-module cascade network to improve image quality, particularly in low-light conditions. The method successfully balances global and local enhancements, but its reliance on high computational power can limit its use on resource-constrained mobile devices. In contrast, PhotoGleam offers a more efficient solution by using AI to generate filters that require minimal user input, allowing for real-time processing on mobile devices without compromising on image quality.

6. Conclusions

While the PhotoGleam application demonstrates strong potential in enhancing image quality and increasing user engagement through AI-generated filters, there are several limitations that need to be addressed. One major limitation is the dependency on the backend server for filter generation, which may cause delays in processing, especially under high user traffic. To improve this, implementing a more robust and scalable server architecture could mitigate latency issues.

Another limitation is the heavy reliance on Gemini for filter recommendations and suggestions. In the event of Gemini not being available, PhotoGleam would not be operational as no filter suggestions will be returned [14]. The entire function of PhotoGleam depends on Gemini being functional. Additionally, Gemini occasionally has issues returning data in the required JSON format. This issue can be addressed by training a new image recognition model that can recommend custom filters based on the content of the image.

Finally, the user interface could benefit from further refinement to enhance the overall user experience, particularly in making the camera more functional by adding more zoom, focus, and exposure options.

In conclusion, PhotoGleam presents a novel approach to mobile photography by leveraging AI to generate custom filters that enhance image quality and increase user engagement [15]. While there are areas for improvement, the app's current functions, intuitiveness, and effectiveness position it as a valuable tool for both amateur and professional photographers.

References

[1] Yang, Zhi. "A Brief Talk about Camera Phone Photography in the Era of Digital Photography." 2018 2nd International Conference on Management, Education and Social Science (ICMESS 2018). Atlantis Press, 2018.

[2] Jensen, Timothy J. "An Introduction to the Modern DSLR Camera." Budget Astrophotography. Springer, New York, NY, 2015. 1-14.

[3] Morris, Katie. A Beginner's Guide to iPhone 6 and iPhone 6 Plus: Or iPhone 4s, iPhone 5, iPhone 5c, iPhone 5s with iOS 8. Gadchick, 2014.

[4] S. Yoon et al., "World Largest Mobile Image Sensor with All Directional Phase Detection Auto Focus Function," 2021 IEEE Hot Chips 33 Symposium (HCS), Palo Alto, CA, USA, 2021, pp. 1-22, doi: 10.1109/HCS52781.2021.9567122.

[5] Alexander White, "Photographer's Guide to the Sony DSC-RX100 VII: Getting the Most from Sony's SONY DSC RX100 VII".

[6] Fukushima, Tadashi, et al. "Architecture of an image signal processor." Electronics and Communications in Japan (Part I: Communications) 67.2 (1984): 62-70.

[7] Orhei, Ciprian, and Radu Vasiu. "Image sharpening using dilated filters." 2022 IEEE 16th International Symposium on Applied Computational Intelligence and Informatics (SACI). IEEE, 2022.

[8] Moran, Sean, et al. "Deeplpf: Deep local parametric filters for image enhancement." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

[9] Pan, Zhaoqing, et al. "MIEGAN: Mobile image enhancement via a multi-module cascade neural network." IEEE Transactions on Multimedia 24 (2021): 519-533.

[10] Géron, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc.", 2022.

[11] Team, Gemini, et al. "Gemini: a family of highly capable multimodal models." arXiv preprint arXiv:2312.11805 (2023).

[12] Jørgensen, Jakob S., et al. "Core Imaging Library-Part I: a versatile Python framework for tomographic imaging." Philosophical Transactions of the Royal Society A 379.2204 (2021): 20200192.

[13] Bourhis, Pierre, Juan L. Reutter, and Domagoj Vrgoč. "JSON: Data model and query languages." Information Systems 89 (2020): 101478.

[14] Imran, Muhammad, and Norah Almusharraf. "Google Gemini as a next generation AI educational tool: a review of emerging educational technology." Smart Learning Environments 11.1 (2024): 22.

[15] O'Brien, Heather L., and Elaine G. Toms. "What is user engagement? A conceptual framework for defining user engagement with technology." Journal of the American society for Information Science and Technology 59.6 (2008): 938-955.