FlashFit: A Mobile and Smart-Mirror System for Real-Time Virtual Clothing Fitting Using Mediapipe and Cross-Platform Technologies

Xinyuan Qi[1], Garret Washburn[2]

[1]Aquinas International Academy, 3200 Guasti Rd, Suite 100, Ontario, CA 91761
qixinyuan1230@gmail.com
[2]California State Polytechnic University, Pomona, CA, 91768
softcom.lab.cpp@gmail.com

ABSTRACT

As long as online cloth shopping has existed, the issue of seeing how a clothing item looks on a potential customer has always followed. The method proposed in this paper is the FlashFit mobile application and smart-mirror [1]. FlashFit is a system which allows the user to upload images of their own clothing items or ones they've found online and have dynamically resized and fitted onto their image via a recording from the mobile app or in real time with the smart-mirror. The major technologies utilized to create the FlashFit system include Mediapipe for human landmark recognition, the Flutter framework for cross-platform mobile app creation, and a raspberry pi for the smart-mirror [2]. During the development process, the major challenges we faced included the resizing of the clothing item given the position of the user in the camera view as well as the creation of png images from user uploaded clothing item images. Additionally, we employed multiple different experiments within this paper to ensure the FlashFit systems consistency, in which the app performed very well. In result, the FlashFit system is a modern solution to the online clothing market's inability to have fitting rooms and is reliable compared to other solutions currently available.

KEYWORDS

Virtual Fitting Room, Human Landmark Recognition, Smart Mirror, Online Clothing Retail

1. INTRODUCTION

For as long as online cloth shopping has existed, the issue of knowing what the clothing item will look like on the customer has always been a problem. Providing some sort of insight into how clothing items or accessories would fit or look like on the customer has been a problem most online clothing retailers have been investing in finding a solution to for quite a while. This problem affects a lot of online shoppers, as 43% of U.S. consumers bought clothing online over 12 months in 2024 going into 2025. So while this issue may not be one of a life-saving gravity, it still affects a lot of online businesses and consumers and is therefore worthy of a proper solution. The primary victim of this issue, while the argument could be made that both the business and the consumer are, we believe, are the customers. The customers pay good money that they've worked hard to earn and deserve an opportunity to see how the item will look on them, even when shopping online.

Additionally, a look at other solutions to the problem outlined in this paper are reviewed. These solutions include the Tryitout AI online web service, the Learning-Based Animation of Clothing for Virtual Try-on system by Seddi Inc., and lastly a Mixed Reality Virtual Clothes Try-On System proposed in 2013 by a group of researchers [3]. These creators all posed functional solutions to the online clothing market for virtual try-on, however, not all were as practical as others or the FlashFit system. Tryitout AI is not an inherently free service, and creates generative AI images of

the clothing item would look like on the user given an image of said user. While this undoubtedly gives cool results, it is not necessarily accurate as the AI is instructed to make the clothing item fit and look good on the user. The next two solutions proposed in their respective papers, require 3D modeling tools to create meshes of a user as well as the clothing item [4]. While this definitely a great solution and undoubtedly gives hyper-realistic results, it is not exactly practical for an at-home user who is trying to test fit clothing items they have access to. The FlashFit system aims to provide easy and practical access to virtual try-on for users who want to test outfits with their own clothing items they take pictures of or find online.

To solve this problem, this paper proposes the FlashFit system. The FlashFit system includes a mobile application and optional additional mirror, both packaged with a machine learning powered process for taking clothing items and the user's camera feed to virtually layer the clothing item onto the user. All of this is so that they may see what the clothing item or accessory looks like on them before they make their purchase. Additionally, the FlashFit system is not inherently tied to any specific store, and is capable of processing an clothing item image a user uploads, including their own or images they are able to find online. The FlashFit system is a valid solution to this problem, as it is entirely free to use for the consumer and provides an accurate demonstration of what the clothing item will look like on them. Additionally, compared to other solutions available currently, the FlashFit system has an easy learning curve and does not require an advanced amount of processing power or special, often expensive, equipment. The FlashFit system is a stay at home clothing and accessory try-on solution that anyone can download and use and expect accurate result with any input they can muster.

Within this paper, two main experiments are recorded showcasing the quickness of the general tasks the FlashFit system is responsible for. These two experiments cover the system's processing time versus the input video length as well as the loading time of a processed video versus the video length. In both experiments, we have found the FlashFit system to remain proficient in both tasks. For the System's processing time, we have found the rule of thumb for processing time to be, on average, around 10 times the input video length. This does take a while, however, when put into consideration the task at hand as well as the nature of the testing server, it is very clear to see how much this will improve. For the second experiment, the time of loading a video was very positive. The results conclude that the average loading time remained consistently around 1.5 seconds for each video. We imagine this is the case because of the employed use of Firebase for file storage, as Firebase is a great cloud solution for a database structure.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Navigating the ever-changing Flutter frameworks

During the development process of the mobile application, the most challenging aspect was navigating the ever-changing Flutter framework. The Flutter framework is very advanced and quite bleeding edge. This results in a complicated framework that feels as though it is constantly changing, and code that we would write a week had to be modified as it no longer meets today's definitions. Additionally, navigating and incorporating all of the functionalities that the mobile needed to have, such as camera and back-end service connection, was quite challenging. However, in the end we proved to have created a very stable and thought through mobile application.

2.2. The memory management issues

Another major challenge we faced during the creation of FlashFit was the memory management issues posed by the back-end server. The hosting provider we used for the back-end server was Render.com, and while they do provide a stable service and connection, their memory space is quite limited. The current server we are using only has 2gb of memory, which for image processing is quite limited. This results in us having a slower processing time for our users. However, this problem could easily be solved by implementing a server that has more than the current amount of memory. Just adding 2gb more of memory would likely result in twice the speed of processing.

2.3. The creation of the core FlashFit system

By far, the most challenging problem we encountered during development had to be the creation of the core FlashFit system, the Mediapipe-powered clothing overlay [5]. This system employs Mediapipe to identify the major landmarks on the user in the camera space as well as the clothing item, dynamically adjusts and resizes the clothing item to fit the user, and then layers the item on top of the user. Establishing this system was its own task, but fine tuning and adjusting it was quite a challenge as well in itself. Additionally, ensuring the user and clothing's landmarks were identified were also a challenge we had to solve, which ensured proper functionality throughout.

3. SOLUTION

The three major components that are linked together to comprise the FlashFit project are the FlashFit mobile application, the Render.com hosted back-end server, and lastly Mediapipe for the landmark identification and clothing overlay. The FlashFit project begins when a user downloads the mobile app or visits the website online, and records a video of themselves modeling whatever clothes they would like to try on. After they have finished recording themselves, the user is able to submit their video for processing. When they submit, the video is sent to the Render.com back-end server, where it is processed by our algorithm. Our algorithm includes Mediapipe for identifying the image landmarks of the user's major body, after which, the clothing items are overlaid onto the corresponding landmarks for the relevant body parts (i.e. shoulders and arms for a sweater) [10]. After the video has completed processing and the final video has been saved locally, the video is sent to the Firebase database to store. Upon the user navigating to the history page, the user will be able to see and download all processed videos to review the fit for their clothing items. The Flutter framework was utilized to maintain the FlashFit mobile application, as well as the Flask server framework for creating and running the flow of the back-end server [6].

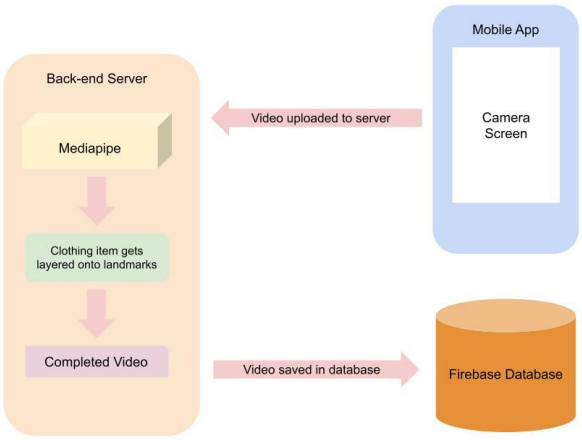


Figure 1. Overview of the solution

The FlashFit mobile application serves the purpose of allowing the user to interact with the back-end server, and even further, the machine learning algorithm designed to create the final video with the clothing item overlaid [7]. Additionally, the mobile app also serves as a place for the user to easily record their video as well as select what clothing items to overlay.

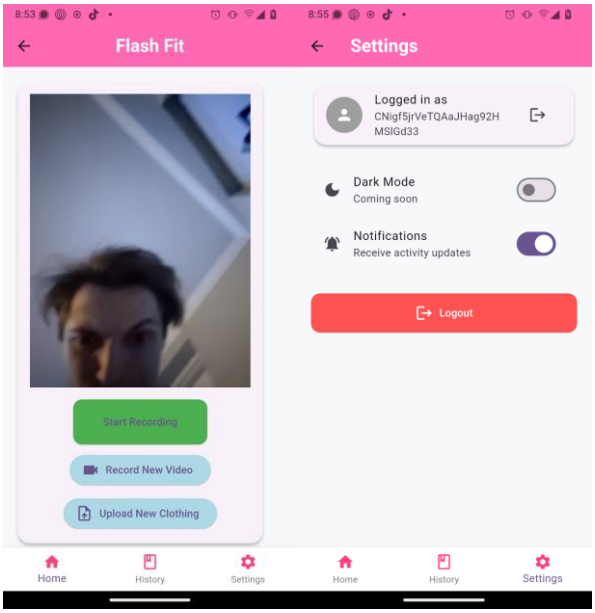
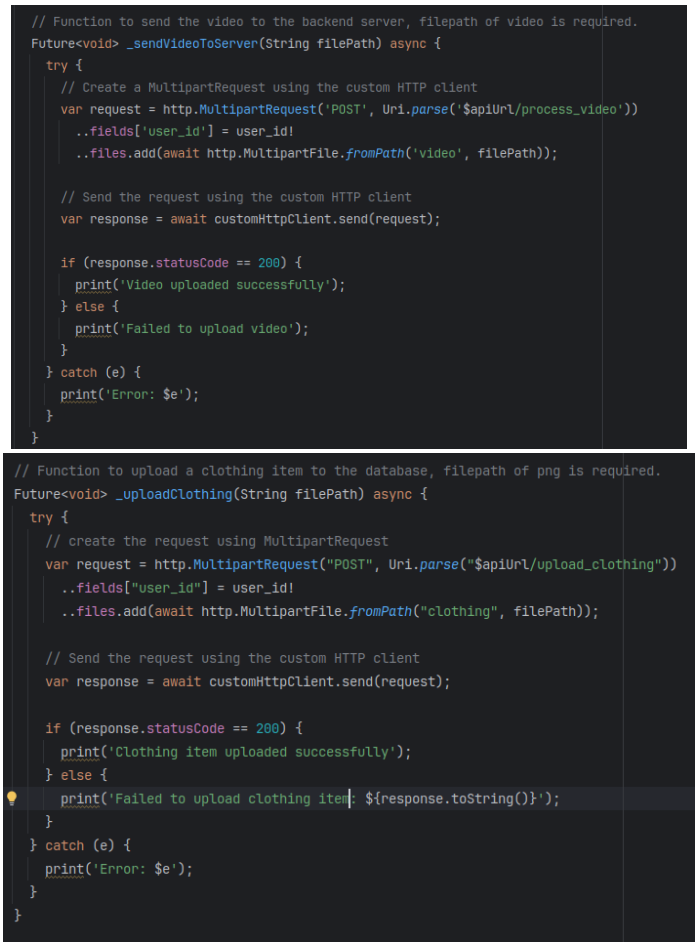


Figure 2. Screenshot of flash fit and settings



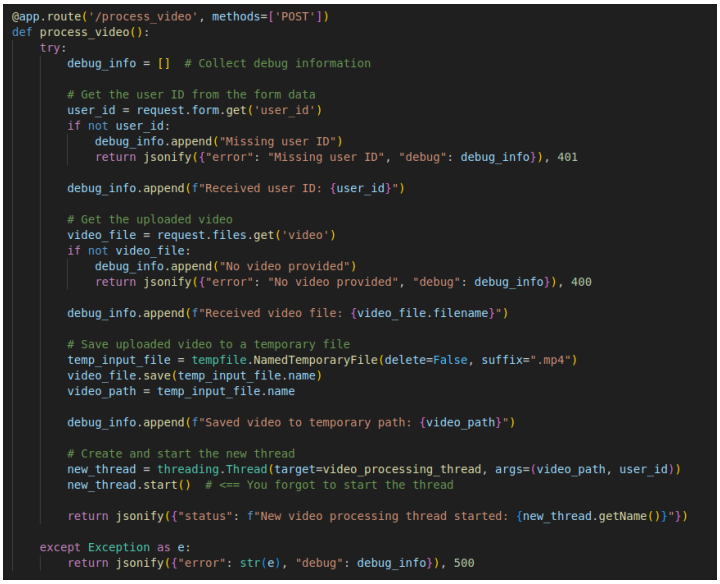
```dart
// Function to send the video to the backend server, filepath of video is required.
Future<void> _sendVideoToServer(String filePath) async {
  try {
    // Create a MultipartRequest using the custom HTTP client
    var request = http.MultipartRequest('POST', Uri.parse('$apiUrl/process_video'))
      ..fields['user_id'] = user_id!
      ..files.add(await http.MultipartFile.fromPath('video', filePath));

    // Send the request using the custom HTTP client
    var response = await customHttpClient.send(request);

    if (response.statusCode == 200) {
      print('Video uploaded successfully');
    } else {
      print('Failed to upload video');
    }
  } catch (e) {
    print('Error: $e');
  }
}
```

```dart
// Function to upload a clothing item to the database, filepath of png is required.
Future<void> _uploadClothing(String filePath) async {
  try {
    // create the request using MultipartRequest
    var request = http.MultipartRequest("POST", Uri.parse("$apiUrl/upload_clothing"))
      ..fields["user_id"] = user_id!
      ..files.add(await http.MultipartFile.fromPath("clothing", filePath));

    // Send the request using the custom HTTP client
    var response = await customHttpClient.send(request);

    if (response.statusCode == 200) {
      print('Clothing item uploaded successfully');
    } else {
      print('Failed to upload clothing item: ${response.toString()}');
    }
  } catch (e) {
    print('Error: $e');
  }
}
```

Figure 3. Screenshot of code 1

The code seen in the above screenshots are the _sendVideoToServer and _uploadClothing functions. These functions serve the purpose, as their names suggest, the sending of both the user recorded videos and clothing item images to the back-end server for processing and handling. As seen in each function, the functions start with a try statement, in which if any errors are caught, the function catches them and will print the error. Within the try statements, a multipart request is made, with the contents of the user_id as well as the video or clothing item file itself. After the multipart requests are made, they are sent using a custom http client object. The purpose of this custom http client is due to our back-end server having an ssl certificate, but not a domain or verified ssl. After the responses are collected, the status code is checked to ensure that the video or clothing item was posted. If there were no issues and the status code is 200, then the mobile app logs that the request was successfully sent.

The next major component within the FlashFit project is the hosted Render.com back-end server [8]. The back-end server serves as the application’ s backbone, as it is the central for all communication between the mobile app users, the mediapipe machine learning algorithm, and the Firebase database.

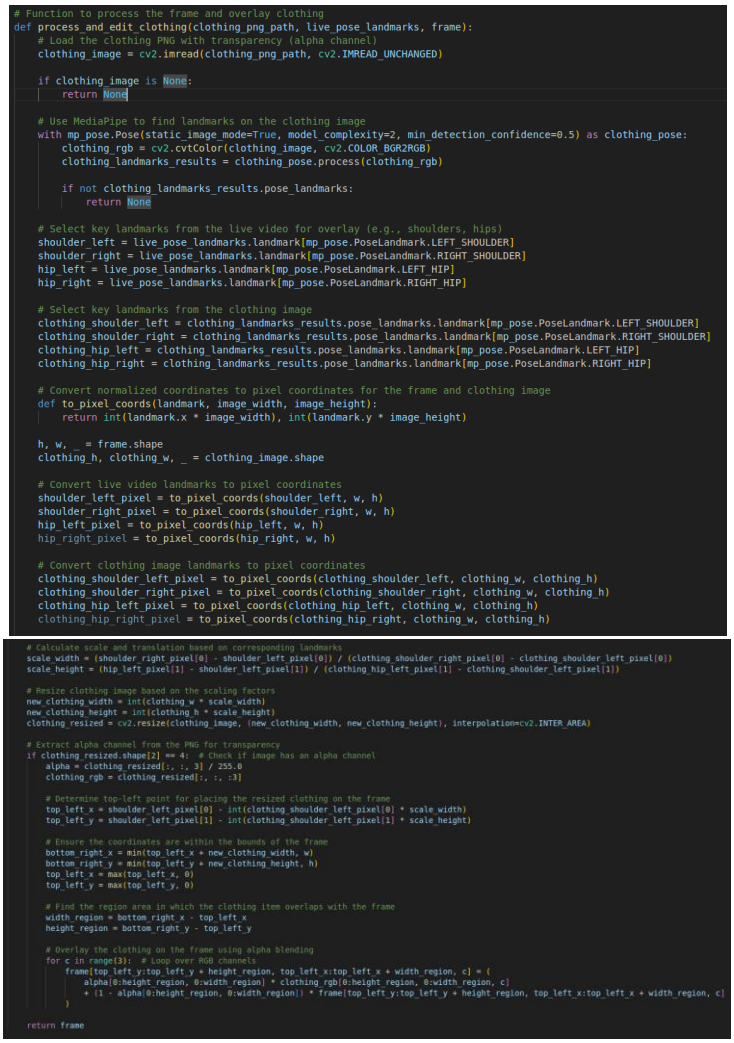


```python
@app.route('/process_video', methods=['POST'])
def process_video():
    try:
        debug_info = []  # Collect debug information

        # Get the user ID from the form data
        user_id = request.form.get('user_id')
        if not user_id:
            debug_info.append("Missing user ID")
            return jsonify({"error": "Missing user ID", "debug": debug_info}), 401

        debug_info.append(f"Received user ID: {user_id}")

        # Get the uploaded video
        video_file = request.files.get('video')
        if not video_file:
            debug_info.append("No video provided")
            return jsonify({"error": "No video provided", "debug": debug_info}), 400

        debug_info.append(f"Received video file: {video_file.filename}")

        # Save uploaded video to a temporary file
        temp_input_file = tempfile.NamedTemporaryFile(delete=False, suffix=".mp4")
        video_file.save(temp_input_file.name)
        video_path = temp_input_file.name

        debug_info.append(f"Saved video to temporary path: {video_path}")

        # Create and start the new thread
        new_thread = threading.Thread(target=video_processing_thread, args=(video_path, user_id))
        new_thread.start()  # <== You forgot to start the thread

        return jsonify({"status": f"New video processing thread started: {new_thread.getName()}"})

    except Exception as e:
        return jsonify({"error": str(e), "debug": debug_info}), 500
```

Figure 4. Screenshot of code 2

The above screenshot displays the '/process_video' route within the Flask back-end server. This route serves the purpose of being able to receive the user's video and start the video processing thread. Upon receiving the request, this server route will grab the user_id as well as the video file from the request. It also ensures their presence and will return an appropriate error if one is missing. After obtaining the user_id and video file, the route creates a temporary file with the video_file's content and will start the new thread with the video_processing_thread. Upon the starting of the thread, the user is returned a response indicating that their video processing has been started. Additionally, the entire content of the route is wrapped in a try statement, as to prevent any unwanted exceptions halting the server and to provide detailed error reports.

The Mediapipe powered clothing overlay is the last major component within the FlashFit application. This algorithm takes care of the processing of the video and overlaying of the clothing item onto the user's body.



```python
# Function to process the frame and overlay clothing
def process_and_edit_clothing(clothing_png_path, live_pose_landmarks, frame):
    # Load the clothing PNG with transparency (alpha channel)
    clothing_image = cv2.imread(clothing_png_path, cv2.IMREAD_UNCHANGED)

    if clothing_image is None:
        return None

    # Use MediaPipe to find landmarks on the clothing image
    with mp_pose.Pose(static_image_mode=True, model_complexity=2, min_detection_confidence=0.5) as clothing_pose:
        clothing_rgb = cv2.cvtColor(clothing_image, cv2.COLOR_BGR2RGB)
        clothing_landmarks_results = clothing_pose.process(clothing_rgb)

        if not clothing_landmarks_results.pose_landmarks:
            return None

    # Select key landmarks from the live video for overlay (e.g., shoulders, hips)
    shoulder_left = live_pose_landmarks.landmark[mp_pose.PoseLandmark.LEFT_SHOULDER]
    shoulder_right = live_pose_landmarks.landmark[mp_pose.PoseLandmark.RIGHT_SHOULDER]
    hip_left = live_pose_landmarks.landmark[mp_pose.PoseLandmark.LEFT_HIP]
    hip_right = live_pose_landmarks.landmark[mp_pose.PoseLandmark.RIGHT_HIP]

    # Select key landmarks from the clothing image
    clothing_shoulder_left = clothing_landmarks_results.pose_landmarks.landmark[mp_pose.PoseLandmark.LEFT_SHOULDER]
    clothing_shoulder_right = clothing_landmarks_results.pose_landmarks.landmark[mp_pose.PoseLandmark.RIGHT_SHOULDER]
    clothing_hip_left = clothing_landmarks_results.pose_landmarks.landmark[mp_pose.PoseLandmark.LEFT_HIP]
    clothing_hip_right = clothing_landmarks_results.pose_landmarks.landmark[mp_pose.PoseLandmark.RIGHT_HIP]

    # Convert normalized coordinates to pixel coordinates for the frame and clothing image
    def to_pixel_coords(landmark, image_width, image_height):
        return int(landmark.x * image_width), int(landmark.y * image_height)

    h, w, _ = frame.shape
    clothing_h, clothing_w, _ = clothing_image.shape

    # Convert live video landmarks to pixel coordinates
    shoulder_left_pixel = to_pixel_coords(shoulder_left, w, h)
    shoulder_right_pixel = to_pixel_coords(shoulder_right, w, h)
    hip_left_pixel = to_pixel_coords(hip_left, w, h)
    hip_right_pixel = to_pixel_coords(hip_right, w, h)

    # Convert clothing image landmarks to pixel coordinates
    clothing_shoulder_left_pixel = to_pixel_coords(clothing_shoulder_left, clothing_w, clothing_h)
    clothing_shoulder_right_pixel = to_pixel_coords(clothing_shoulder_right, clothing_w, clothing_h)
    clothing_hip_left_pixel = to_pixel_coords(clothing_hip_left, clothing_w, clothing_h)
    clothing_hip_right_pixel = to_pixel_coords(clothing_hip_right, clothing_w, clothing_h)
```

```python
    # Calculate scale and translation based on corresponding landmarks
    scale_width = (shoulder_right_pixel[0] - shoulder_left_pixel[0]) / (clothing_shoulder_right_pixel[0] - clothing_shoulder_left_pixel[0])
    scale_height = (hip_left_pixel[1] - shoulder_left_pixel[1]) / (clothing_hip_left_pixel[1] - clothing_shoulder_left_pixel[1])

    # Resize clothing image based on the scaling factors
    new_clothing_width = int(clothing_w * scale_width)
    new_clothing_height = int(clothing_h * scale_height)
    clothing_resized = cv2.resize(clothing_image, (new_clothing_width, new_clothing_height), interpolation=cv2.INTER_AREA)

    # Extract alpha channel from the PNG for transparency
    if clothing_resized.shape[2] == 4:  # Check if image has an alpha channel
        alpha = clothing_resized[:, :, 3] / 255.0
        clothing_rgb = clothing_resized[:, :, :3]

        # Determine top-left point for placing the resized clothing on the frame
        top_left_x = shoulder_left_pixel[0] - int(clothing_shoulder_left_pixel[0] * scale_width)
        top_left_y = shoulder_left_pixel[1] - int(clothing_shoulder_left_pixel[1] * scale_height)

        # Ensure the coordinates are within the bounds of the frame
        bottom_right_x = min(top_left_x + new_clothing_width, w)
        bottom_right_y = min(top_left_y + new_clothing_height, h)
        top_left_x = max(top_left_x, 0)
        top_left_y = max(top_left_y, 0)

        # Find the region area in which the clothing item overlaps with the frame
        width_region = bottom_right_x - top_left_x
        height_region = bottom_right_y - top_left_y

        # Overlay the clothing on the frame using alpha blending
        for c in range(3):  # Loop over RGB channels
            frame[top_left_y:top_left_y + height_region, top_left_x:top_left_x + width_region, c] = (
                alpha[0:height_region, 0:width_region] * clothing_rgb[0:height_region, 0:width_region, c]
                + (1 - alpha[0:height_region, 0:width_region]) * frame[top_left_y:top_left_y + height_region, top_left_x:top_left_x + width_region, c]
            )

    return frame
```

Figure 5. Screenshot of code 3

In the above screenshot, the process_and_edit_clothing method is defined to handle the landmark identification on the person and clothing item and the overlay of their landmarks. The function starts by opening the clothing image and identifying the landmarks on it. Next, the landmarks of the person in the image are found, and the key landmarks for laying a top are found for both the person and top. These landmarks include the left and right shoulders and hips. Then, the landmarks are converted to pixel values so that the clothing item may be placed accurately on top of the person. Lastly, the image is scaled to the scale of the user in the image, and the clothing item is overlaid onto the person and the frame is returned. Overall, the method is quite elaborate and utilizes a lot of math for mapping the clothing item on top of the person.

4. EXPERIMENT

4.1. Experiment 1

A crucial aspect of the FlashFit project is the consistency and speed of the back-end servers ability to process user uploaded videos. It is important that the back-end server be able to process the videos and serve the resulting processed video in a timely manner so users have quick results.

For this experiment, the design will remain quite simple. The experiment will consist of two experimenters, one responsible for uploading videos of different lengths and monitoring the back-ends processing, and the other responsible for keeping track of time. The first experimenter will upload a video, giving the second experimenter the signal to start the timer. Once the processing is complete, the first experimenter will then give the signal for the second to stop the timer. The experimenters will then log their findings, keeping track of how long each video of different length takes to complete processing. The video lengths will start at 5 seconds, then increase in length by 5 seconds till they reach a 20 second video.

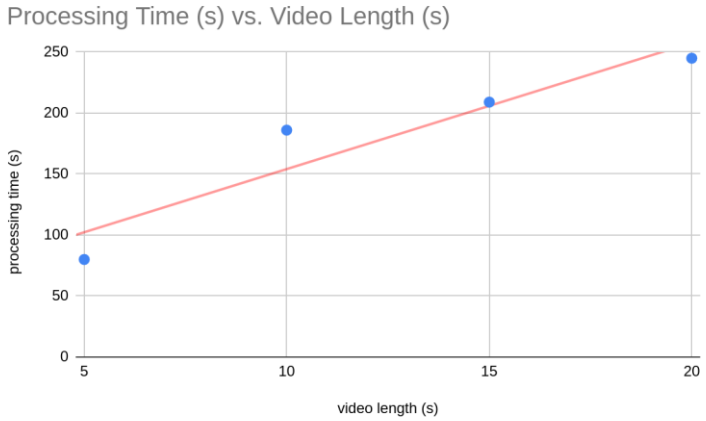


Figure 6. Figure of experiment 1

After performing the experiment, the experimenters were able to find estimates of the time it took to perform the processing given their experiment setup. Given the data collected, it is clear that there is a positive trend that as the length of the user submitted video increases, the processing time seems to increase on average around 10 times the input video length. Evidently, however, the increase in processing time does not appear to be consistent given the outliers in the data. However, these occurrences could be due to a few flaws in the experimental setup. Despite this, though, the experimental setup was designed to reach the place of the user and put the experimenter in the shoes of the end user so the results will be more like what they experience. We believe the results to be sporadic due to a few factors. These factors could be due to the changing position of the user in the different videos including position and scenery as well as the obvious potential internet and server connection issues. However, the experiment does capture what it will actually be like for the user, therefore, the results remain accurate.

4.2. Experiment 2

Another crucial component within the application is the result viewing feature. It is important that the viewing feature within the application works quickly and consistently, as the intention of the entire app is to receive these results.

To ensure the consistency of the result viewing feature, we will perform an experiment to find the average loading time it takes for a video result to be viewed after requesting. The experiment team will consist of two members, the first for clicking open the item and informing the second experimenter when to start and stop the timer. In order to ensure a consistent downloading experience and to identify any trends, we will perform this experiment by downloading and viewing videos of different lengths (the same lengths as performed in the second experiment).

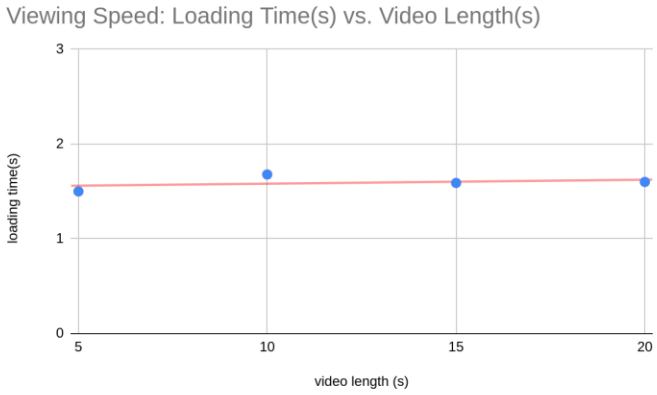


Figure 7. Figure of experiment 2

At the experiment's conclusion, the experimenters found that the result viewing downloading and viewing speed stayed quite consistent amongst various different video sizes ranging from 5 to 20 seconds. The average download time floating around 1.5 seconds, with an actual average of 1.59 seconds. Additionally, taking a look at the data in the above chart, it is clear to see due to the trendline and points that the average download speed remained consistent across the different video lengths. We believe this is due to Firebase Storage database managing the hosting, and therefore supplying consistent and stable download links for the user's device to download from [9]. It is important to note, however, that the download speed does stutter at the 10 second data point. We believe this is mainly due to general lag and completely random, as that is always a consideration we must consider.

5. Related Work

One methodology for solving the issue of the inability to try on clothes at home is Tryitout AI [11]. Tryitout AI is an online tool which allows the user to upload an image of themselves and the clothing item they would like to 'try on.' The Tryitout AI model then takes the images and merges them, creating a new generated image of you wearing the clothing item. While this seems great as a concept, an obvious issue with the system is the model is designed to assume the clothing item fits. The FlashFit system does not utilize image generation and gives the user an up front honest view of what the clothing item would look like on them.

Another solution to the same problem is the Learning-Based Animation of Clothing for Virtual Try-on system proposed by Seddi Inc. [12]. Seddi proposed this system, which utilized complete 3D meshes of clothings items and people, as a solution which creates hyper-realistic renderings of

what a clothing item would look like on a person. The solution is highly advanced and does a great job of creating these renderings, however, there are some unfortunate implications. In order for the system to be used, complete 3D mesh renderings of both the clothing item and user must be created. This requires special, and expensive, tooling that the user does not have access to. The FlashFit system provides a practical, easy to use, and completely free solution for users to experiment with designs using their own clothing items as well as others they find online.

The Mixed Reality Virtual Clothes Try-On System proposed in a paper in 2013 by a group of researchers is a system, similar to the previous method, that creates an approximate 3D model of the user using machine learning and fits the clothing image on top of that mesh [13]. This system is quite useful, as it does not require the user to use advanced 3D modeling tooling to create a mesh of the clothing item or themselves, and instead allows the user to simply upload the clothing item and stand in the camera view. However, It is apparent in the data depicted in the paper that the proportions of the 3D renderings of the user tend to vary in accuracy. It appears as though the rendering model is a bit hasty with generalizations, and is not too proficient at getting accurate proportions of each user. The FlashFit system dynamically resizes the size of the clothing item, without changing proportions, to fit the user wherever they are in the image. This ensures a proper look of what the clothing item would actually look like on the user.

6. CONCLUSIONS

Reflecting back on the current state of the FlashFit project, it is evident that there are a few drawbacks to the current system. The first item on the list of things to fix is the layering of the clothing item onto the user. Occasionally, a clothing item does not line up properly on the user, such as a t-shirt not aligning with a user's shoulders or pants not extending down to the user's ankles. A solution to this problem could be implementing some more in-depth resizing logic utilizing the user's proportions, however, we have also considered implementing a similar meshing system as seen in some of the other methodologies covered previously [14]. Additionally, the mobile app version, not including its functionality with the interactive mirror, does require an extensive amount of time for the user's video to be processed and be able to be viewed by the user. This creates a delay of service, and is not particularly convenient. The solution that we are actively working on to remedy this is to package the cloth layering system in the app, so the process can consistently happen in real time [15].

At the conclusion of the FlashFit project, I am extremely grateful for all of the support that I have received from my family and individuals who have helped me during the development of this project. FlashFit poses as a fantastic solution for individuals to be able to try on clothes from home, and I am satisfied with making it available for free for anyone to use.

REFERENCES

[1] Yusri, Muhammad Mu'izzudeen, et al. "Smart mirror for smart life." 2017 6th ICT International Student Project Conference (ICT-ISPC). IEEE, 2017.

[2] Faust, Sebastian. Using Google´ s Flutter framework for the development of a large-scale reference application. Diss. Hochschulbibliothek der Technischen Hochschule Köln, 2020.

[3] Meng, Yuwei, Pik Yin Mok, and Xiaogang Jin. "Interactive virtual try-on clothing design systems." Computer-Aided Design 42.4 (2010): 310-321.

[4] Steffen, Wolfgang, et al. "Shape: A 3D modeling tool for astrophysics." IEEE Transactions on Visualization and Computer Graphics 17.4 (2010): 454-465.

[5] Lugaresi, Camillo, et al. "Mediapipe: A framework for building perception pipelines." arXiv preprint arXiv:1906.08172 (2019).

[6] Lathkar, Malhar. Building Web Apps with Python and Flask: Learn to Develop and Deploy Responsive RESTful Web Applications Using Flask Framework (English Edition). BPB Publications, 2021.

[7] Mahesh, Batta. "Machine learning algorithms-a review." International Journal of Science and Research (IJSR).[Internet] 9.1 (2020): 381-386.

[8] Dima, M., M-T. Dima, and M. Mihailescu. "Flash-Fit Algorithms for Circles in Particle Physics." Physics of Particles and Nuclei Letters 21.4 (2024): 800-803.

[9] Sudiartha, I. K. G., et al. "Data structure comparison between mysql relational database and firebase database nosql on mobile based tourist tracking application." Journal of Physics: Conference Series. Vol. 1569. No. 3. IOP Publishing, 2020.

[10] Lugaresi, Camillo, et al. "Mediapipe: A framework for perceiving and processing reality." Third workshop on computer vision for AR/VR at IEEE computer vision and pattern recognition (CVPR). Vol. 2019. 2019.

[11] Lloyd, Peter. "You make it and you try it out: Seeds of design discipline futures." Design Studies 65 (2019): 167-181.

[12] Santesteban, Igor, Miguel A. Otaduy, and Dan Casas. "Learning ‐ based animation of clothing for virtual try ‐ on." Computer Graphics Forum. Vol. 38. No. 2. 2019.

[13] Yuan, Miaolong, et al. "A mixed reality virtual clothes try-on system." IEEE Transactions on Multimedia 15.8 (2013): 1958-1968.

[14] Rao, Rajeev R., David Blaauw, and Dennis Sylvester. "Soft error reduction in combinational logic using gate resizing and flipflop selection." Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design. 2006.

[15] Ruckman, J. E. "The application of a layered system to the marketing of outdoor clothing." Journal of fashion marketing and management: An international Journal 9.1 (2005): 122-129.