Virtual Try-on Platform

By

Yeaw Wooi Tong

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2025

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Ms Tong Dong Ling who has given me this bright opportunity to engage in virtual try on project. It is my first step to establish a career in VTO field. A million thanks to you.

Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

COPYRIGHT STATEMENT

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ABSTRACT

The project focuses on developing a comprehensive Virtual Try-On (VTO) application that integrates Deep learning (DL) technologies. The aim is to provide users with an immersive experience to visualize and customize their full-body look, including clothing, makeup. This solution addresses the limitations of existing virtual try-on systems, which are often constrained to specific categories and reliant on in-store systems. The proposed VTO app is designed to offer a complete styling experience, allowing users to experiment with their entire look within one platform. The use of DL enhances effectiveness and realism of the Virtual Try-On (VTO) system. The app also tackles challenges related to hygiene and time consumption by eliminating the need for physical try-ons in stores, making it a timely solution in the post-pandemic digital shopping era. The project employs Android as the development platform to ensure accessibility across a wide range of devices, particularly in emerging markets. By leveraging advanced technologies, the app promises a more convenient, and engaging shopping and styling experience.

Area of Study (Minimum 1 and Maximum 2): Deep Learning

Keywords (Minimum 5 and Maximum 10): Mobile Application, Deep Learning, Virtual Try-On System, Segmentation, AI-Driven Image Synthesis

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

VTO	Virtual Try-On
SDM	Supervised Descent Method
AI	Artificial Intelligence
DL	Deep Learning
SAM 2	Segment Anything 2
AR	Augmented Reality
TPS	thin-plate spline

CHAPTER 1

Introduction

The rapid advancements in digital technology have significantly transformed industries, including the fashion sector. Traditionally, personal styling and fashion decisions required physical interactions, such as trying on clothes, applying makeup, and selecting accessories. These activities, typically done in physical stores, involved consumers trying on multiple outfits to find the right combination for events or social gatherings. This process of repeatedly changing and adjusting outfits to achieve the desired look is time-consuming and often exhausting. Virtual try-on (VTO) systems, which allow users to visualize clothing and accessories on their bodies before making a purchase, have become an integral part of the fashion industry's digital evolution [1].

In response to the COVID-19 pandemic, the retail landscape, especially in non-essential sectors like fashion, witnessed significant changes as consumers shifted from in-store shopping to online platforms. With the pandemic exacerbating hygiene concerns, VTO systems have gained even more importance, providing a safer, contactless shopping experience [2]. The growing demand for online shopping has made VTO systems crucial for consumers who seek convenience and flexibility in their shopping experience.

This project seeks to address these inefficiencies by developing a Virtual Try-On Application that offers full-body customization. The app allows users to virtually simulate their entire look, including clothes, makeup before stepping out or attending an event. Unlike existing applications, which tend to focus on specific aspects of styling (such as makeup or clothing), this app integrates several styling elements, offering users a complete and immersive customization experience. Moreover, the application enhances the online shopping experience by providing a realistic preview of how products will look on the user’s body. This feature reduces the need for physical try-ons, lowering the likelihood of returns. Additionally, some current virtual try-on systems are only available in physical stores, limiting their convenience and accessibility. The Virtual Try-On App solves this issue by allowing users to style themselves from anywhere, providing engaging experience. According to Kavin et al.

(2024), such applications, using tools like OpenCV and Unity3D, reduce the need for physical try-ons, making the shopping process more efficient and user-friendly.

The project will implement Deep learning technologies, enable users to visualize their selected products in real-time. By generating accurate simulations, users can confidently make styling decisions without physically interacting with the items, thus creating a seamless, efficient, and convenient shopping and styling experience. The ability to interact with virtual garments enhances the shopping experience by making it more immersive and realistic [4].

1.1 Problem Statement and Motivation

The current landscape of virtual fashion and styling tools often lacks an all-encompassing solution that allows users to fully visualize and customize their appearance. Most digital fashion tools are either limited to specific categories, such as makeup or clothing, or confined to physical store systems that require users to be on-site. This creates a fragmented and inconvenient experience for consumers who want to coordinate their entire look for an event or occasion.

Additionally, physically trying on clothes and makeup in stores is not only time-consuming but also a source of fatigue for many consumers. The repetitive process of changing outfits to find the perfect combination often leads to dissatisfaction with the shopping experience. The COVID-19 pandemic has further exacerbated this issue, with growing concerns about hygiene when using shared fitting rooms or makeup testers in stores. As Ghodhbani et al. (2022) suggest, the pandemic has increased consumer demand for contact-free shopping solutions, further highlighting the need for virtual try-on systems that offer a comprehensive and user-friendly experience.

The Virtual Try-On App aims to solve these problems by offering a single platform where users can visualize their entire look in one go. By using Deep learning, the app provides accurate simulations of products, allowing users to make informed decisions about their styling choices without the need for physical interactions. This not only enhances convenience but also addresses consumer concerns about hygiene and time efficiency.

The motivation for developing the Virtual Try-On App stems from the desire to improve both the user experience in personal styling and the online shopping process. Traditionally, achieving the perfect look for events or daily wear requires significant

time spent trying on multiple outfits. This is often a frustrating and exhausting process, particularly when consumers are uncertain about how the different elements of their look will come together. The Virtual Try-On App seeks to eliminate this issue by allowing users to experiment with their entire look virtually, reducing the time and effort involved in achieving their desired appearance.

In the digital age, consumers are increasingly turning to online shopping for convenience, but the inability to physically try on items remains a significant barrier. This project aims to bridge the gap between the physical and virtual shopping experience by providing users with a reliable and realistic simulation of how clothes, makeup will fit and look on them. The use of advanced technologies, such as Deep learning, further enhances the user experience by offering accurate visualizations that mimic real-world outcomes.

The app is also driven by the need to offer a more hygienic and time-efficient alternative to traditional in-store experiences. By enabling users to virtually customize their look from the comfort of their own home, the app eliminates the need for physical try-ons, reducing potential health risks and saving time. This shift to a virtual solution offers consumers a greater level of convenience, allowing them to style themselves without the constraints of physical locations.

1.2 Project Objectives

1. **To develop an Android-based Virtual Try-On (VTO) application**:

   The core objective of this project is to build a mobile app that functions on the Android platform, making use of the operating system's flexibility and broad user base. By targeting Android, the app will be accessible to a wide range of devices, from high-end smartphones to budget-friendly options, thus increasing its reach.

2. **To integrate Deep learning technologies :**

   The app leverages deep learning models for clothing segmentation and try-on image generation. These technologies enhance the user experience by accurately combining clothing items with user photos and enabling real-time virtual try-ons.

3. To enable full-body customization:

The app will allow users to try on clothing, makeup in real time, offering a full-body customization experience. This will enable users to visualize their entire look in one seamless experience, eliminating the need for physical try-ons in stores.

1.3 Project Scope and Direction

The scope of this project is to develop an Android-based Virtual Try-On (VTO) application that integrates Deep Learning (DL) technologies to provide users with an seamless virtual styling experience. The app will allow users to virtually try on clothing, makeup. A key feature of the project is its focus on full-body customization, enabling users to visualize a cohesive look by trying on multiple fashion items within a single platform. The app is designed as a mobile application, ensuring users can conveniently use the app from anywhere and at any time, making it a flexible solution for modern lifestyles. By optimizing the app for a broad range of Android devices, the project ensures wide accessibility across various smartphones, catering to a diverse audience. The final product will be a fully functional, scalable, and versatile VTO app that delivers an accessible and accurate virtual styling solution to meet modern consumer needs.

1.4 Contributions

The Virtual Try-On App introduces several important contributions that have the potential to transform the way users interact with fashion and personal styling. One of the most significant contributions of the app is its ability to offer a complete and cohesive full-body customization experience. Unlike existing apps that focus on individual aspects of styling, such as makeup or clothing, this app brings together all the necessary elements such as clothing and makeup into one integrated platform. Users can experiment with their entire look within a single app, allowing for a smoother and more enjoyable styling process.

Another major contribution of the app is the use of deeo learning technologies to deliver a realistic user experience. By leveraging DL, the app enables users to

visualize their chosen products in real time. The incorporation of deep learning allows the app to refine these simulations ensuring a more accurate and customized experience.

Additionally, the app offers a convenient, store-free experience that significantly enhances the online shopping process. Users no longer need to visit physical stores to try on items or rely on store-based virtual try-on systems. Instead, the app allows them to experiment with their look anytime and anywhere, making the process of personal styling more accessible and flexible. The app's ability to provide realistic visual try-on previews, reduces the chances of purchasing clothes that do not suit them, ultimately lowering return rates and improving overall user satisfaction. As noted by Ghodhbani et al. (2022), such advancements can significantly improve the user experience by reducing the need for physical store visits and minimizing the likelihood of returns due to poor fit.

1.5 Report Organization

This report is structured into several chapters to comprehensively present the development of the Virtual Try-On (VTO) mobile application. **Chapter 1** introduces the project by outlining the background, motivation, objectives, scope, and contributions. **Chapter 2** provides a literature review of existing VTO applications and related research, highlighting their strengths, weaknesses, and the research gaps this project aims to address. **Chapter 3** details the proposed methodology, including system design specifications, architecture, tools and technologies, AI model workflows, and performance requirements. **Chapter 4** presents the system design, covering authentication, wardrobe management, clothing segmentation, and try-on image generation. **Chapter 5** discusses system implementation, including hardware and software setup, module integration, and operational workflows. **Chapter 6** focuses on system testing and evaluation, outlining the testing strategies, setup, and results. Finally, **Chapter 7** concludes the report by summarizing the project's outcomes and proposing future directions for enhancement.

CHAPTER 2

Literature Reviews

2.1 Existing VTO Application

The rapid development of **deep learning technologies** has revolutionized the fashion and beauty industries. **Virtual Try-On (VTO) systems** allow users to visualize clothing, makeup, in a virtual environment, offering a more interactive and engaging shopping experience. However, existing VTO solutions have limitations in terms of usability, accessibility, and completeness, which often hinder their adoption and effectiveness.

2.1.1 YouCam Makeup App

The YouCam Makeup app, developed by Perfect Corp., is a leading example of beauty tech that leverages AR and machine learning to provide users with a comprehensive virtual try-on experience for makeup. Launched with the aim of enhancing the digital beauty shopping experience, YouCam allows users to apply virtual makeup, analyze their skin, and receive personalized product recommendations based on their unique features. The app combines elements of social commerce with beauty tech, making it not only a VTO platform but also a tool for AI-powered skincare analysis, livestream tutorials, and personalized beauty consultations.

A standout feature of the app is its ability to map over 200 facial landmarks using patented AgileFace technology, which allows for the precise application of virtual makeup and real-time facial retouching. The skin analysis tool evaluates factors such as wrinkles, dark circles, and moisture levels, generating a "skin age" score for users, which can prompt purchases from partnering beauty brands to "improve" the user's skin. This personalization and integration with beauty brands have led to increased engagement and higher conversion rates during the pandemic, with brand partners' conversion rates increasing by up to two and a half times[5].

Figure 2.1 YouCam Interface

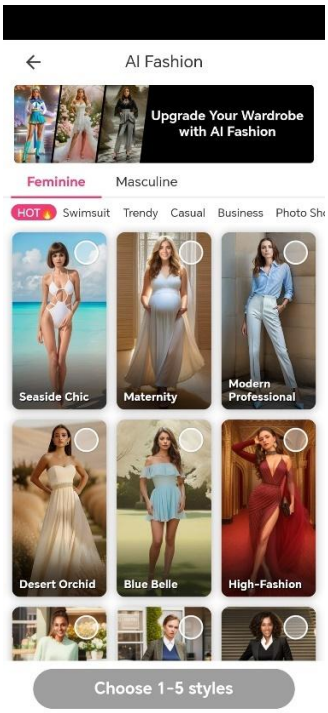


Figure 2.2 AI Fashion

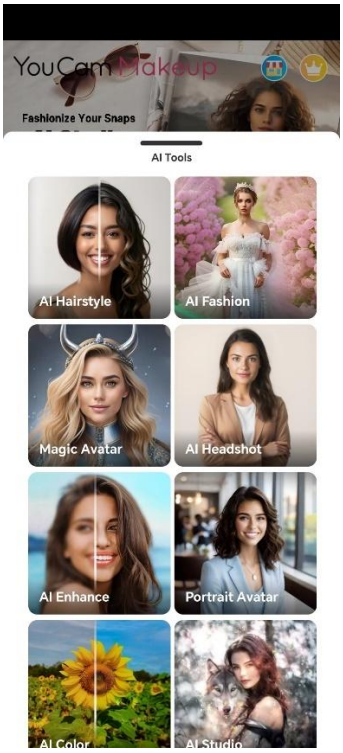


Figure 2.3 AI Tools

Limitations

YouCam Makeup delivers a solid virtual makeup experience through AR and AI, yet its capability is focused on beauty aspects such as makeup application, skin analysis, and facial retouching. It has no feature for clothing try-on or fashion styling. Hence, users looking for a complete styling experience including clothes must rely on other apps. The lack of clothing integration renders it less convenient for full-body virtual try-on or fashion coordination.

2.1.2 Smart Mirrors in Retail

The integration of smart mirrors in retail stores has brought a transformative change to the shopping experience. Smart mirrors, equipped with augmented reality (AR) and artificial intelligence (AI), allow customers to virtually try on outfits without physically changing clothes. As Daiani (2024) explains, smart mirrors offer users contextually relevant environments, such as a beach or wedding setting, enabling them to visualize how an outfit would look in a particular situation. The use of these

contextual features enhances both customer satisfaction and decision-making, as users can better assess the suitability of their attire for specific events.

Smart mirrors have also been shown to improve customer engagement by offering interactive features such as gesture recognition, voice commands, and personalized recommendations based on previous shopping behavior [6]. These technologies create a more immersive and convenient shopping experience, enabling users to make informed purchase decisions without the need to physically try on multiple outfits. For instance, interactive lighting controls allow customers to visualize how an outfit looks in different lighting environments, such as outdoor or indoor settings, enhancing the realism of the virtual try-on experience.

Figure 2.4 Smart Mirrors

Limitations

Smart mirrors provide interactive in-store experiences by enabling users to try clothes virtually with AR and contextual environments. However, they are meant for physical stores and thus not accessible to users shopping online or at home. In addition, they usually come with expensive hardware and upkeep, limiting them to high-end stores. Most smart mirrors also lack makeup try-on capability, so they are unable to offer an end-to-end styling experience that includes both fashion and beauty.

2.1.3 Closet Organization Apps (Pureple, Cladwell, Closet+)

Closet organization apps such as **Pureple**, **Cladwell**, and **Closet+** provide users with the ability to digitally catalog and manage their clothing, helping them better visualize, plan, and organize outfits [7]. While these apps serve as **virtual wardrobe managers**, they focus exclusively on **clothing**.

- **Pureple**: Offers features like cataloging clothes, planning outfits, and a community-driven outfit suggestion system, but focuses solely on clothing. It suffers from an ineffective outfit suggestion algorithm, frequent advertisements, and a poor user interface.

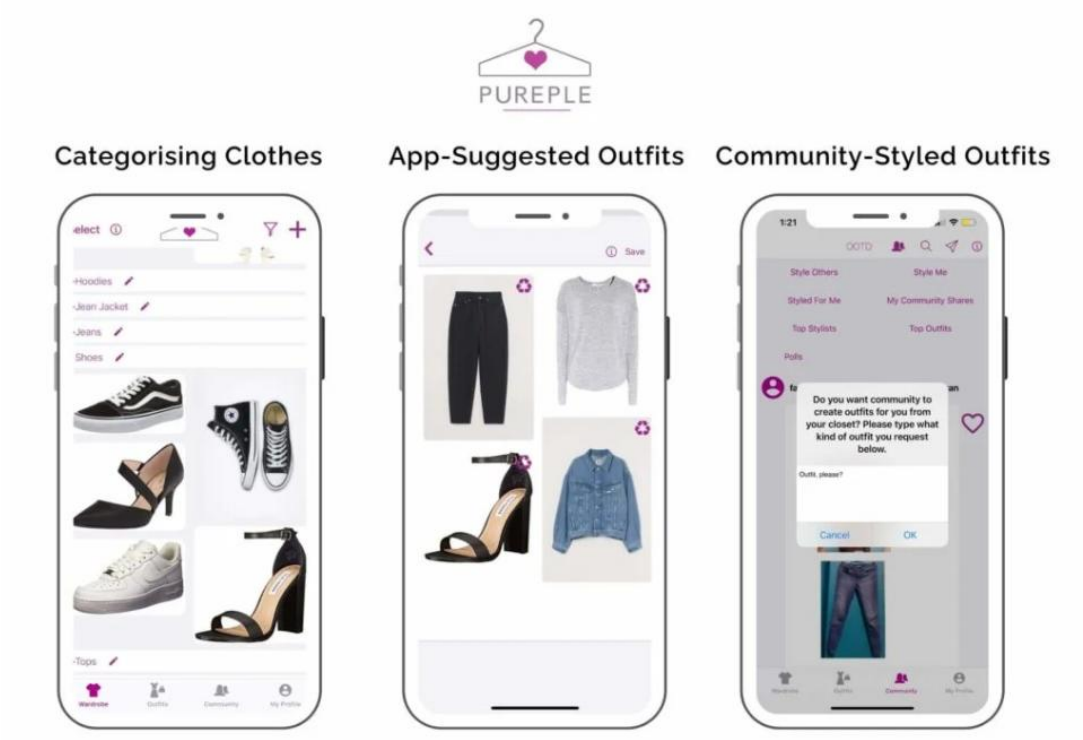


Figure 2.5 Pureple

- **Cladwell**: Provides weather-appropriate outfit suggestions and allows users to track their wardrobe utilization. However, it only focuses on clothing management and do not offer **try-on capabilities.**

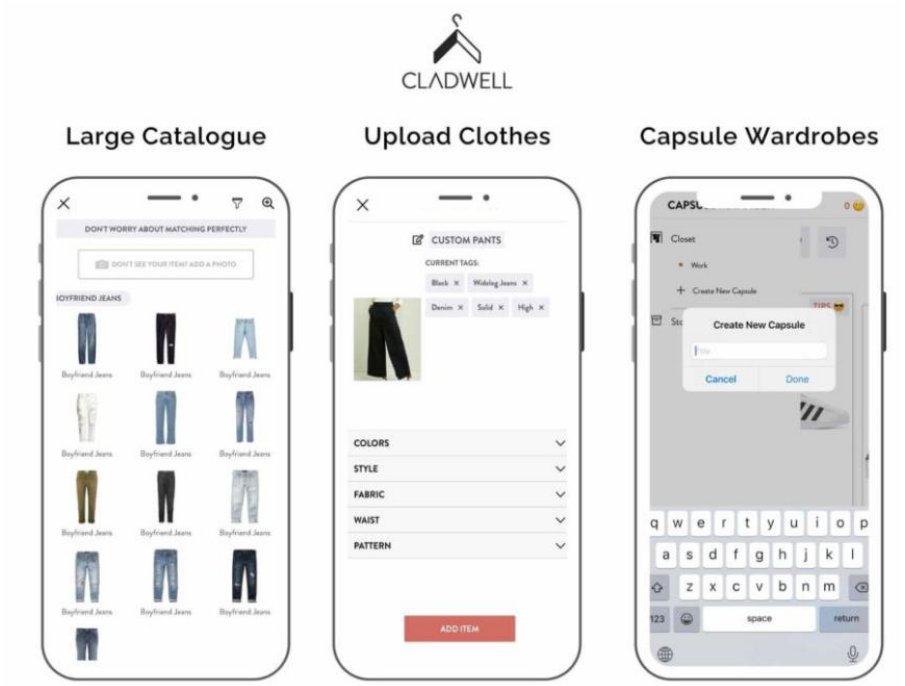


Figure 2.6 Cladwell

- **Closet+**: A simpler app focused on organizing clothes, creating outfits, and planning what to wear for future events. It requires manual uploads of clothing photos and do not offer **try-on capabilities.**

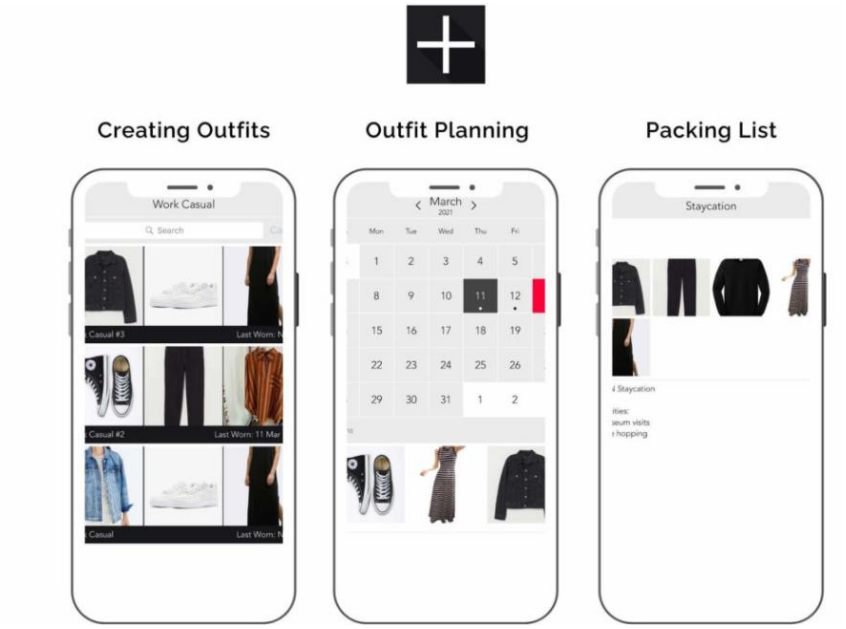


Figure 2.7 Closet+

Limitations

These apps are largely virtual closet organizers. While they help users manage, plan, and sort through their outfits, they don't include virtual try-on simulation. The users cannot view dynamic pictures of themselves wearing the clothes; all they get is static pictures of the clothing, with no capability to view the clothing on them. Further, they also do not permit makeup styling and facial beauty, and most don't include AI-based outfit suggestions. This restricts their use to simple wardrobe management instead of an active, interactive fashion experience.

2.1.4 Style.me: AR-Based Virtual Fashion Try-On

Style.me is an advanced virtual try-on platform that integrates AR and AI to provide users with an immersive experience in trying on clothing items virtually. The app allows users to visualize clothing on realistic 3D avatars generated based on their body measurements. Style.me uses deep learning algorithms and computer vision to enhance the precision of fit and garment simulation, ensuring that users can visualize how clothes will fit and drape on their bodies. A standout feature of Style.me is its focus on offering a comprehensive virtual shopping experience. Unlike other apps that focus solely on a single product category, Style.me integrates various types of clothing, including tops, bottoms, dresses, and outerwear. It offers retailers and brands the ability to showcase their collections in virtual form, enhancing the online shopping experience and increasing engagement[8].

Figure 2.8 Style.me

Limitations

Style.me uses AI and 3D avatars to mimic clothing try-ons with a virtual dressing room experience that feels real. It does this only for individual pieces of clothing that are already in the app through partnerships with fashion brands, however. Users can't upload their own images of clothing they'd like to try on, which limits personalization. Style.me is solely for clothing, too, and doesn't allow makeup or accessories, so it can't offer a complete head-to-toe styling option.

2.2 Previous work on VTO

2.2.1 VITON: An Image-based Virtual Try-on Network

Overview and Strength

VITON (Virtual Try-On Network) is the very first image-based virtual try-on system that was designed to clothe an individual in a target photo with an outfit from a product image without relying on 3D body modeling (Han et al., 2018). This was a departure from the conventional approaches that relied on costly and complex 3D scans or mesh reconstructions. VITON also utilizes a two-stage coarse-to-fine generation pipeline. During the first phase, a coarse image of the person with the target clothing is generated by an encoder-decoder network based on a clothing-agnostic representation such as pose heatmaps, body shape masks, and facial areas in order to retain important visual information. The second stage continues to enhance the output by warping the garment through thin-plate spline (TPS) transformation and blending it with the image using a learned alpha mask, which helps in enhancing texture alignment and visual realism [9].

One of the most potent arguments for VITON is that it is easily available, as it functions solely on 2D RGB data and does not need depth data or 3D input. The clothing-agnostic design allows successful preservation of pose and identity, and renders the try-on result more realistic in appearance. Two-stage architecture enhances visual quality in general, especially in depicting folds of clothes, texture, and garment arrangement. Its input requirement compatibility also makes it extremely scalable and

deployable to actual online e-commerce platforms. VITON also employs perceptual loss and L1 loss to further improve convergence at training and image realism.

Trained on the Zalando dataset that is image-diverse and richly annotated, VITON has the ability to generalize to a diverse range of garments and body types, a feature that adds to its real-world usability. In spite of all its weaknesses, VITON set the baseline blueprint for most later try-on networks (Han et al., 2018). To show the VITON architecture, Figure 2.8 can be used to demonstrate the two-stage pipeline comprising: (1) the coarse try-on image generation via a U-Net architecture from pose and segmentation maps, and (2) the refinement module with TPS warping and alpha composition. Figure 2.9 can show side-by-side comparisons of the original image, warped garment, and VITON's synthesized output, showing both its effectiveness and limitations [9].

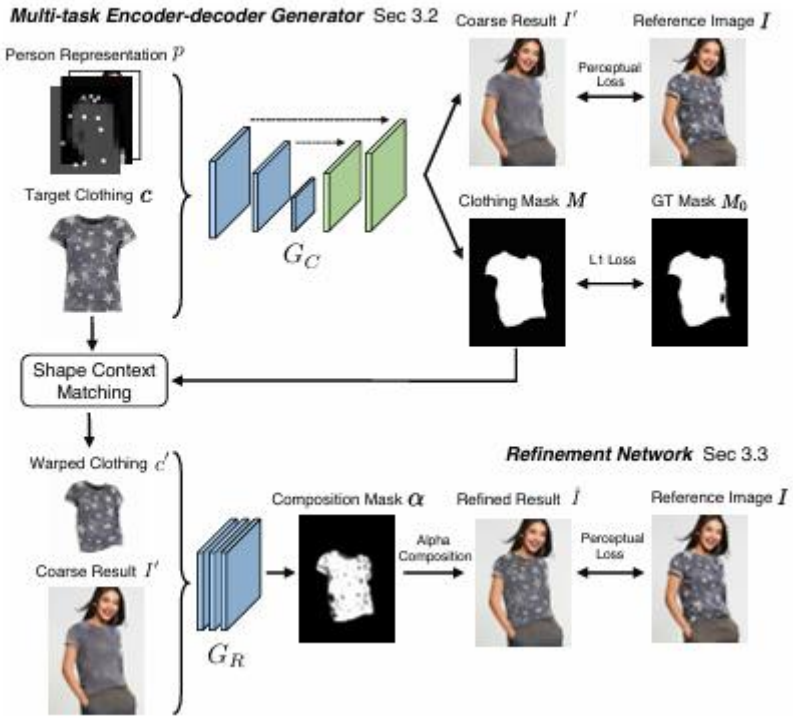


Figure 2.9 two-stage pipeline comprising

Figure 2.10 side-by-side comparisons

Limitations

Despite its pioneering contributions, VITON has several notable limitations. Firstly, the model generates low-resolution outputs restricted to 256×192 pixels, which limits its applicability in high-resolution contexts such as digital advertising or fashion catalog production. Additionally, although VITON includes a refinement stage, it often struggles to maintain high fidelity in garment textures—details like logos, text, or embroidery are sometimes lost or blurred, especially under challenging conditions like occlusion or inconsistent lighting. The model also exhibits sensitivity to body poses and occlusions; it performs poorly when body parts overlap (e.g., crossed arms) or when poses deviate significantly from the norm, leading to unrealistic overlaps or distortions in the synthesized image. Moreover, VITON is primarily tailored for upper-body garments and lacks the generalizability needed to handle full-body outfits or accessories without significant reengineering. Lastly, the TPS-based warping used in VITON, though effective to a degree, lacks the adaptability of newer deep learning-based spatial transformation techniques, which limits its performance on more complex or layered clothing items.

2.2.2 SieveNet: A Unified Framework for Robust Image-Based Virtual Try-On

SieveNet is a remarkable enhancement of image-based virtual try-on (VTO) systems that seeks to enhance the limitations of garment misalignment, artifact generation, and poor pose adaptation typical of earlier models like CP-VTON. It is

designed with three separate modules: the Coarse-to-Fine Warping module, the Conditional Segmentation Mask Prediction module, and the Segmentation-Assisted Texture Translation module. These are combined through the utilization of novel loss functions, such as a perceptual geometric matching loss and a duelling triplet loss, in order to enhance both the photorealism and alignment of the eventual synthesized result.

The overall framework, illustrated in Figure 2.10, starts with a target garment and model input image. This pair is fed into the Coarse-to-Fine Warping module, which conducts a two-stage warping of the garment to match the model pose. The Conditional Segmentation module then produces a garment-conditioned semantic map, which guides the ultimate Segmentation-Assisted Texture Translation module to synthesize a coherent, artifact-free composite image.

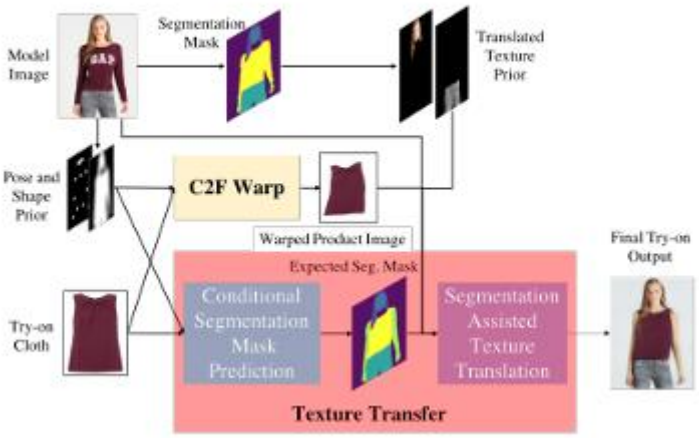


Figure 2.11 Inference Pipeline of the SieveNet framework

The Coarse-to-Fine Warping module applies Thin-Plate Spline (TPS) transformation hierarchically. In step one, a coarse warp roughly aligns the garment with the model's silhouette through a 19-channel person representation. In step two, a perceptual geometric matching loss further aligns the result by matching deep feature distances in a VGG feature space. This two-stage strategy is demonstrated to boost the accuracy of warping significantly, depicted in Figure 2.11, alongside comparing coarse and fine warping results [10].

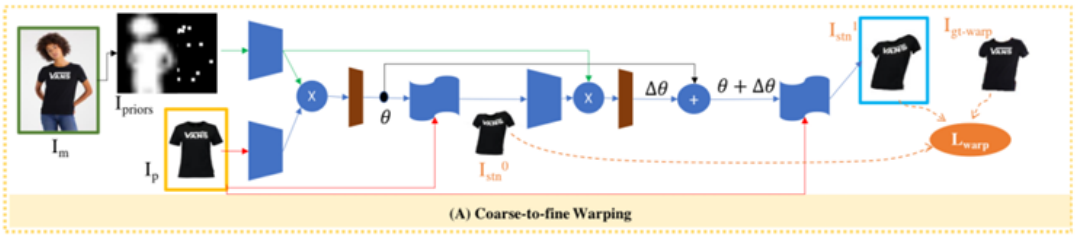


Figure 2.12 Coarse-to-Fine Warping Module

Second, the Conditional Segmentation Mask Prediction module plays a significant role in avoiding classic VTO pitfalls such as bleeding textures or incorrect clothing placement under occlusion or complex poses. A UNet architecture is employed to generate an "expected" segmentation mask to guide the final composition. Examples of its effect, especially on challenging occluded instances, are demonstrated in Figure 2.12 [10].

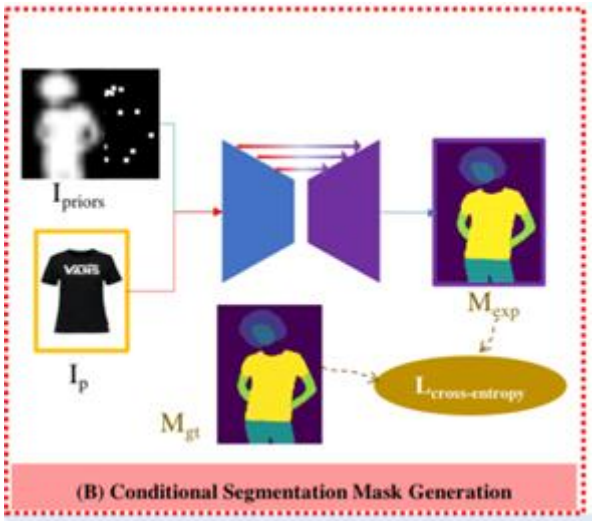


Figure 2.13 Conditional Segmentation Mask Prediction module

The final Segmentation-Assisted Texture Translation module merges the warped clothing and segmentation map to compose the ultimate try-on image. It blends texture from unchanged areas of the original model image and creates the final composition through a learned composition mask. Enhanced by a duelling triplet loss, this module improves image realism by encouraging proximity to ground-truth images and distance from previous-stage outputs. Figure 2.13 shows the heightened fidelity and texture legibility afforded by this strategy [10].

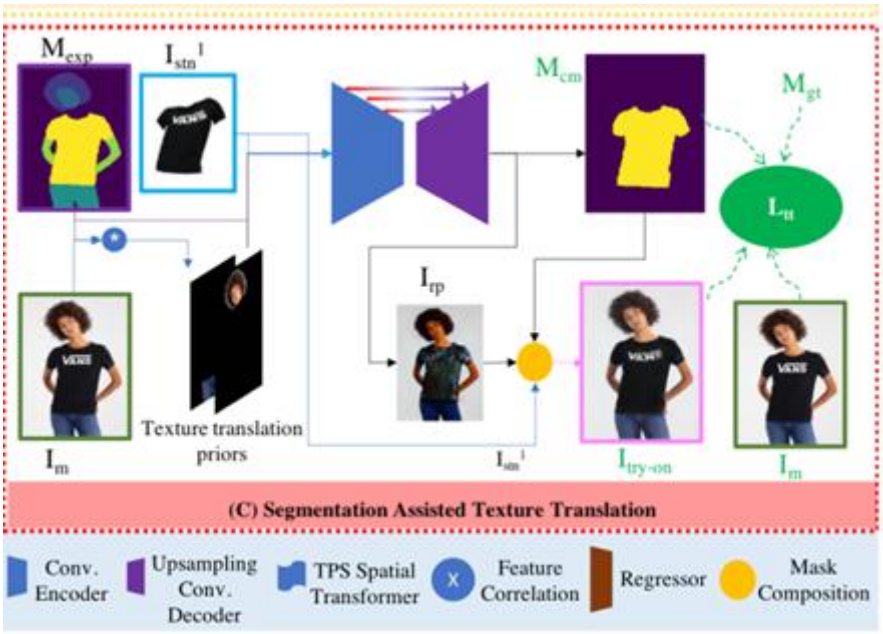


Figure 2.14 Segmentation-Assisted Texture Translation module

Performance metrics reveal SieveNet's superiority over CP-VTON in structural similarity (SSIM), multiscale SSIM, Frechet Inception Distance (FID), and Peak Signal-to-Noise Ratio (PSNR). These are listed in Figure 2.13, which validates the combined strength of every module within the architecture.Qualitatively speaking, SieveNet is able to produce significantly more realistic outputs than CP-VTON [10].

Configuration	SSIM	MS-SSIM	FID	PSNR	IS
GMM + TOM (CP-VTON)	0.698	0.746	20.331	14.544	2.66 ± 0.14
GMM + SATT	0.751	0.787	15.89	16.05	2.84 ± 0.13
C2F + SATT	0.755	0.794	14.79	16.39	2.80 ± 0.08
C2F + SATT-D (SieveNet)	**0.766**	**0.809**	**14.65**	**16.98**	$\mathbf{2.82 \pm 0.09}$

Figure 2.15 Quantitative comparison of Proposed vs CP-VTON. GMM, TOM

The system demonstrates strong warping on diverse body poses and clothing types due to the utilization of a two-stage transformation plan. The conditional segmentation mask is especially effective at maintaining garment outlines and eliminating undesired artifacts. In addition to this, duelling triplet loss encourages texture details and general visual quality, even under occlusion. Side-by-side comparison images in Figure 2.14

illustrate these enhancements, where CP-VTON and SieveNet results are presented comparatively.

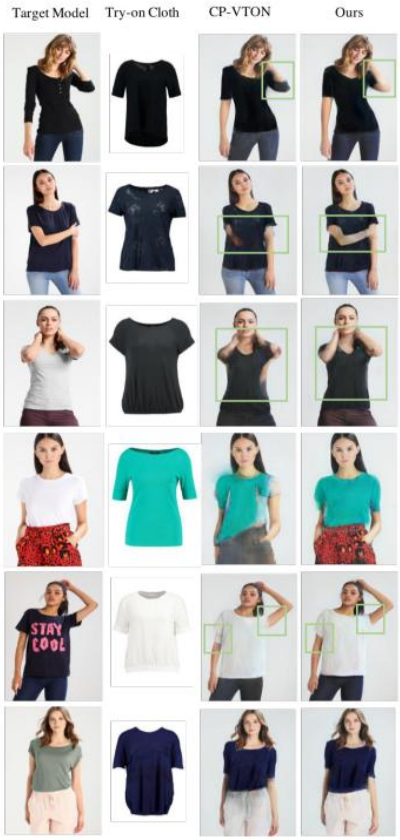


Figure 2.16 Side-by-side comparison images

Overall, SieveNet is a high-quality standard of VTO research that combines architectural novelty with specifically crafted loss functions, thereby yielding a flexible and high-quality solution well-suited for practical fashion application.

Limitations

SieveNet faces several limitations in its current approach. One major challenge is its limited occlusion handling, where it struggles with complex occlusions such as hair or folded limbs. This results in artifacts and errors in the final try-on image, reducing overall quality. Another limitation is its fixed input representation; SieveNet relies on a fixed 19-channel person representation and handcrafted priors, which limits its adaptability across datasets with different formats or image qualities, making it less flexible in diverse scenarios. Additionally, SieveNet operates in a 2D framework

without 3D or depth awareness, which reduces its ability to represent garments realistically, particularly for items that require contour adaptation, such as loose or draping clothing. Without an understanding of spatial depth, SieveNet cannot provide a fully realistic depiction of how clothes fit and move in three-dimensional space.

2.3 Summary of Strengths and Limitations of Existing VTO Solutions

Table 2.1 Summary of Strengths and Limitations

System/App	Strengths	Limitations
YouCam Makeup	- Accurate virtual makeup with AR- Facial landmark detection (AgileFace)- Skin analysis and beauty suggestions	- Only focuses on makeup and skincare - No clothing or full-body styling features
Smart Mirrors	- Real-time AR clothing try-on- Context-aware environments (e.g., lighting)- Gesture and voice interaction	- Confined to physical retail environments - Expensive hardware and maintenance- No makeup or complete look styling
Pureple/ Cladwell/ Closet+	- Effective wardrobe organization- Outfit planning and suggestions- Weather-based recommendations (Cladwell)	- No virtual try-on or image simulation - No makeup or face styling- Mostly static and manual input
Style.me	- High-quality 3D avatars for try-on- Realistic fit and draping using AI- Supports multiple clothing types	- Only supports pre-loaded clothing from brand partners - No user-uploaded clothing - No makeup or accessories integration

VITON	- Image-based VTO without 3D data- Clothing-agnostic person representation- Two-stage refinement with TPS warping- Scalable using 2D RGB data	- Low-resolution outputs (256×192) - Poor texture preservation (logos, embroidery) - Sensitive to occlusion and body pose - Focuses only on upper-body garments - TPS warping lacks deep adaptability
SieveNet	- Coarse-to-fine warping improves alignment- Conditional segmentation helps avoid artifacts- Superior texture translation with triplet loss	- Limited occlusion handling (e.g., folded arms, hair) - Uses fixed handcrafted priors - No 3D/depth awareness - Lacks adaptability to varied datasets or image quality

2.4 Proposed Solution

The proposed comprehensive mobile-based solution addresses past virtual try-on system limitations by introducing an adaptive platform which improves device accessibility through personalization and increased reality levels.

The current VTO systems face two crucial limitations including low-resolution output generation and poor texture retention in clothing appearance. This system resolves such constraints through its implementation of state-of-the-art image synthesis methods which create detailed high-resolution outputs. The algorithm maintains detailed fabric components such as text together with embroidery details and patterns despite changing lighting conditions or partial light exposures. The system employs contemporary deep learning-based spatial transformations instead of using problematic Thin Plate Spline (TPS) warping procedures because TPS functions poorly with

complex poses and layered clothing situations. The methodology provides optimal alignment while retaining visual clarity when dealing with crossed body parts and irregular poses thus lowering the appearance of distortions and artifacts.

The system improves its simulation ability by implementing pseudo-depth understanding capabilities to duplicate 3D knowledge in a two-dimensional display. Such an approach enables the model to better simulate garment draping over the body specifically for loose-fitting garments and multi-layered outfits which results in heightened reality during try-on sessions.

Users can now improve existing garment selection through the proposed solution because it enables the uploading of personal clothing images. Users can integrate their clothing into the virtual wardrobe through segmentation before adding them to the try-on feature. The solution improves user personalization while enlarging the selection of styles available through an app beyond its pre-defined fashion collections.

The VTO mobile application proposal creates a single platform that merges clothing and makeup features to overcome both the single-product focus and the restricted full-body customization. Users benefit from whole-app integration which creates a more convenient and enjoyable process to create their desired look. Users now experience a consistent design flow because they can avoid application switching to test various fashion and beauty products without requiring continuous interface changes.A mobile application development of the VTO system removes users' need to visit physical stores because it fulfills all their styling needs in one platform. Through the application's interface users can test clothing items as well as apply cosmetics and accessories regardless of their present location in a manner that reflects increasing digital shopping trends. The new approach enhances user convenience since it enables remote shopping and styling preferences which are gaining value following the pandemic.

The proposed solution through its mobile platform delivers a comprehensive user-focused approach for addressing existing VTO challenges by delivering enhanced realism and universal compatibility in one convenient mobile platform.

CHAPTER 3

System Methodology/Approach

3.1Methodology

The development of the Virtual Try-On (VTO) mobile application followed an Agile software development methodology. Agile was selected because it supports flexibility, incremental delivery, and iterative refinement of features, which is well suited for combining mobile app development with AI-driven functionality. Through short development cycles (sprints), each feature was designed, implemented, tested, and reviewed before integration into the full system. This ensured that user requirements were continuously addressed while allowing the system to evolve based on testing feedback.

Figure 3.1 Agile Methodology

According to Figure 3.1, the planning phase is the first. During this stage, needs analysis of the application Virtual Try-On was carried out to develop an efficient and convenient system. The idea was to create an application giving the user the ability to upload clothes and personal photos, categorize the clothing items, virtually try on clothes, add makeup effects, and manage what is in the wardrobe. Functional requirements were user authentication, wardrobe management, the usage of the Grounded-SAM2 model to segment clothes, the use of the KolorVTO to generate realistic try-on photos, the use of the Banuba to generate makeup try-on photos and to store the data in Firebase. Non-functional requirements were centered on low-latency response of the AI server, the ability to get a 85 or more image accuracy to ensure try-on realism, and a clean responsive UI that was built in Jetpack Compose.

The second stage is the design phase, which incorporated the system architecture that defines the interplay between the mobile application, AI backend and Firebase cloud services. The architecture was composed of a frontend, which is an Android application based on Jetpack Compose, a backend, which is an application hosted on FastAPI to execute AI models (Grounded-SAM2 to segment an image and KolorVTO to generate an image), and Firebase services to handle user authentication, data, and media storage. SDK to live try-on was also provided in Banuba. This step also included the creation of diagrams like use case diagrams, activity diagrams and system architecture diagram to show the interaction between modules.

The development phase was aimed at the implementation of the modules that were planned. The authentication was developed using Firebase Authentication where users are free to register with email. The management of the wardrobe allowed users to provide pictures of clothes that were forwarded to the backend to be segmented by the Grounded-SAM2 model. The images were segmented and sent back to the app where the user was expected to confirm before classification into Firebase. KolorVTO was also built in to create try-on photos, which gave realistic previews. Banuba SDK was used to allow real-time makeup trial using the camera or uploading a static image. Outfit planning and event management functions were introduced to enable one to save the outfit on a certain date. The modular development process was done so that every individual component was tested to be integrated.

The testing stage consisted in testing the back-end and front-end. In the case of the backend, Grounded-SAM2 segmented the uploaded clothes and the results were compared to determine the level of accuracy. The KolorVTO model has been tested by creating results of try-on and comparing it with the expected behavior in terms of realism. Regarding the frontend, UI/UX items like wardrobe navigation, outfit planner and makeup panel were tested in black-box testing to ensure they acted accordingly. Performance testing was also aimed at testing the response times under varying network conditions as poor connectivity led to more time in processing and lower accuracy of output in other cases.

Deployment stage entailed that the system was packaged into a working prototype used in Android gadgets. AI backend was deployed using Fast API, and it allowed real-time API requests to be made by the app and integrate Firebase services to do the authentication, storage, and data synchronization. The Android devices were launched with the mobile application and tested to make sure the application is compatible with various Android hardware, both high-end and mid-range.

Lastly, the review process assessed the project results with respect to the initial goals. The system managed to deliver clothing segmentation, realistic try-on generation, wardrobe management, event planning and makeup try-on features successfully. The system had strengths such as modular integration of various AI models and scalability on a cloud basis.

3.2 System Design Diagram

3.2.1 System Architecture Diagram

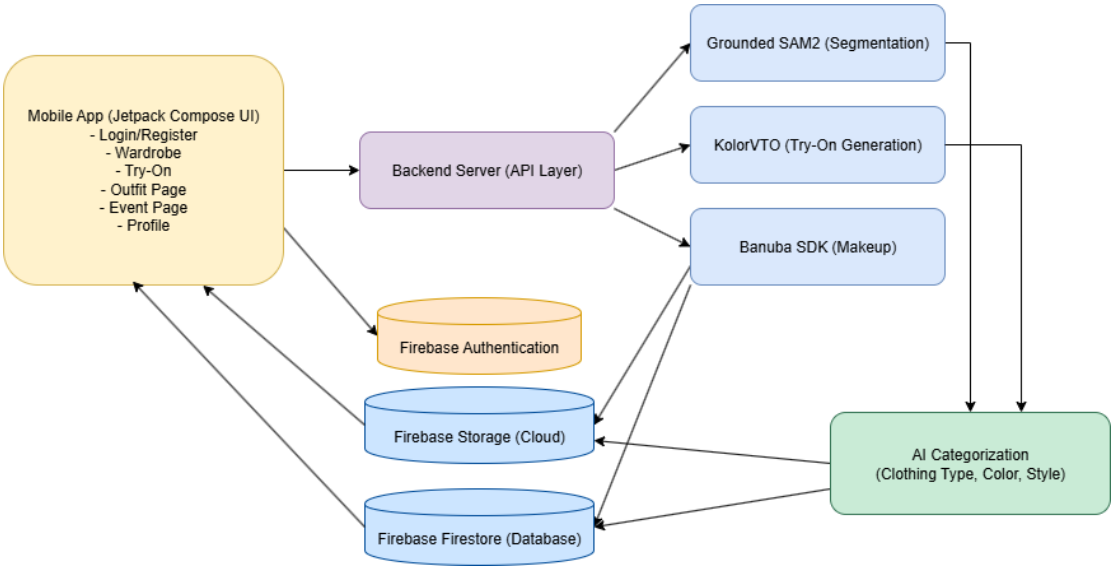


Figure 3.2 System Architecture Diagram

The system architecture of the Virtual Try-On Mobile Application is designed to integrate the mobile app, backend server, AI models, and Firebase cloud services into a seamless workflow. The mobile application, built using Jetpack Compose, serves as the user-facing interface and provides core features such as login and registration, wardrobe management, virtual try-on, outfit page, event planner, and user profile. All requests and user actions initiated from the mobile app are sent to the Backend Server through an API layer.

The Backend Server acts as the central controller, receiving requests from the mobile application and coordinating them with the AI models and Firebase services. It manages three key AI models: **Grounded SAM2**, which performs clothing segmentation to extract apparel from uploaded user images; **KolorVTO**, which generates photorealistic try-on images by overlaying segmented clothes on the user's body; and **Banuba SDK**, which provides virtual makeup simulation including lipstick, foundation, and eyeshadow effects. The outputs from these AI models are not directly stored. Instead, they first pass through the **AI Categorization Module**, which

automatically labels and classifies results with metadata such as clothing type, color, and style.

The AI Categorization Module ensures that both segmented images and try-on results are properly organized before storage. Once categorized, the data is sent to **Firebase Storage** for storing image files such as segmented clothes, generated try-on previews, and makeup results, while **Firebase Firestore** stores structured metadata including clothing categories, user wardrobe details, and event-linked outfit information. Additionally, **Firebase Authentication** handles secure user login and registration. Both Firestore and Storage communicate back to the mobile application, allowing users to view their saved wardrobe items, outfits, and events within the app.

This architecture demonstrates a modular and layered design where the mobile app handles user interaction, the backend server coordinates AI and cloud services, and Firebase ensures secure authentication, scalable data storage, and retrieval. The integration of AI Categorization as an intermediate layer adds intelligence to the system by organizing and labeling outputs, which not only improves wardrobe management but also provides a foundation for future recommendation and personalization features.

3.2.2 Use Case Diagram

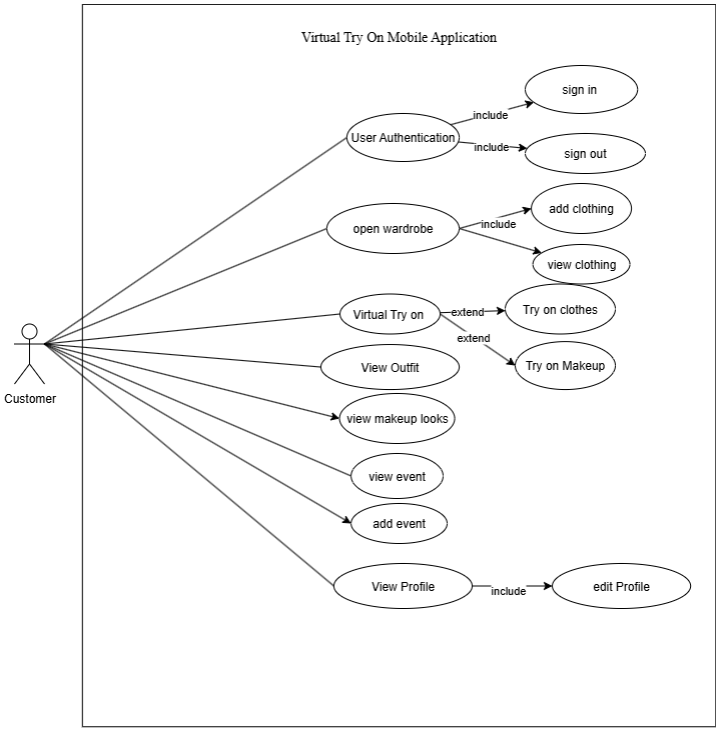


Figure 3.3 Use Case Diagram

The use case diagram of the Virtual Try-On mobile application illustrates the main interactions between the customer and the system. The customer is the primary actor who can access different features after authentication. Through Firebase Authentication, the customer is able to sign in and sign out of the app securely. Once logged in, the customer can manage their wardrobe by uploading clothing images, which are first processed by the segmentaion system to segment and send to AI categorization server classify the items before storing the images and metadata in Firebase services. The customer can also browse and view their wardrobe items. Another major functionality is the Virtual Try-On feature, where the customer can select clothing items from the wardrobe and generate realistic try-on results using the KolorVTO model. This feature also extends to makeup try-on through the Banuba SDK, allowing the customer to visualize full-body styling, with the option to view saved or recommended makeup looks. In addition, the system provides outfit and event planning functions, where the customer can create events, assign outfits to them, and view planned outfits for specific occasions. The customer can also view and edit their personal profile, which includes details such as name, measurements, and style preferences. Overall, the use case diagram demonstrates how the system integrates authentication, wardrobe management, AI-powered categorization, virtual try-on, event planning, and profile management into a cohesive experience for the user.

3.2.3 Activity Diagram

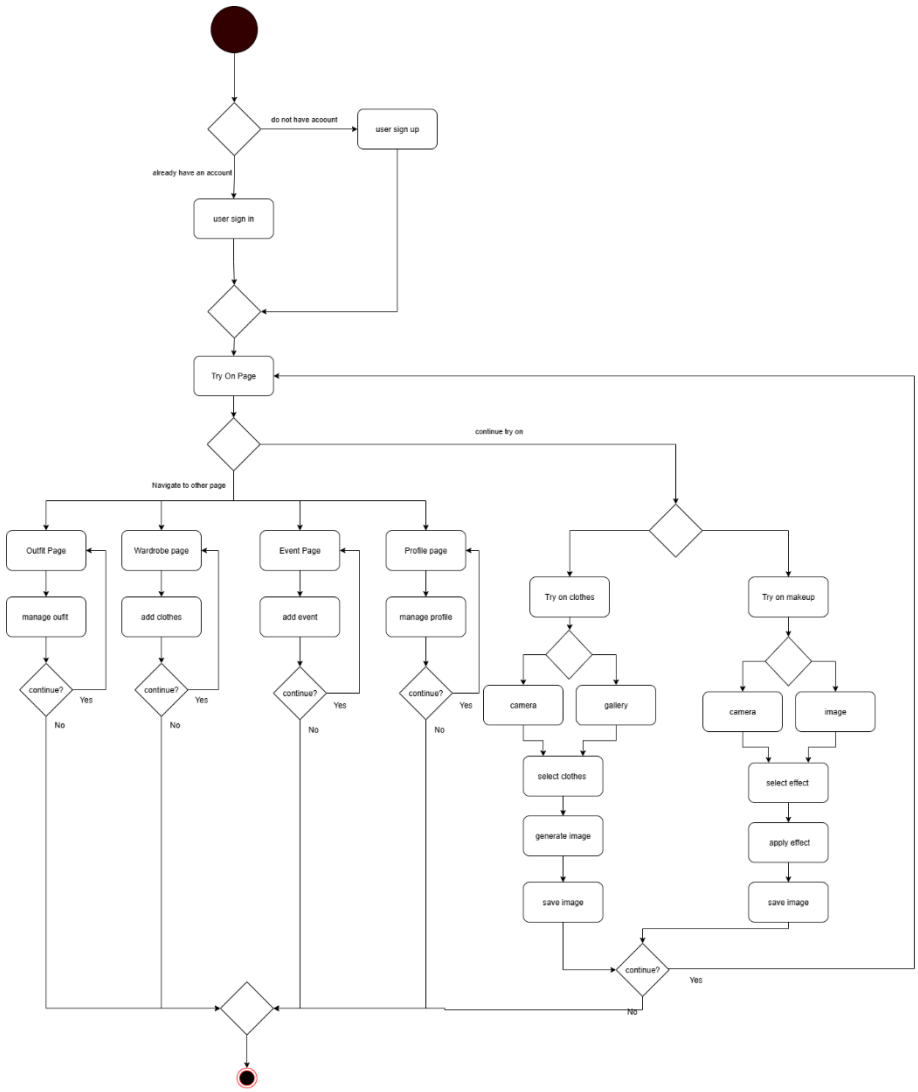


Figure 3.4 Activity Diagram

This system activity diagram illustrates the complete flow of how a user interacts with the Virtual Try-On App. It starts from the moment the user logs into the system and then branches into the different features provided by the app. Once inside, the user can navigate to various sections such as the wardrobe, outfit management, event scheduling, profile management, and the virtual try-on feature.

In the wardrobe, the user is able to upload and manage clothing items that can later be used for styling. The outfit section allows the user to manage the outfit item. The event section supports planning by letting users assign specific outfits to occasions, ensuring that their styling is ready in advance. The profile area allows users to maintain

their personal details and preferences, which influence the overall personalization of the app.

The virtual try-on process is a core part of the diagram, where users can choose to experiment with clothing or makeup. For clothing, the app generates realistic images of the user wearing selected items, while for makeup, the app provides a way to preview different looks. In both cases, the results can be saved for future use.

Overall, the activity diagram shows how the system is designed to give users full flexibility: they can freely move between wardrobe, outfits, events, profiles, and try-on activities, repeating actions as needed until they are satisfied. It emphasizes the interactive and iterative experience the app provides, enabling users to continuously experiment with and refine their personal style.

3.2.4 AI Model Workflow and Performance

SAM2 Model (Clothing Segmentation)

- **Functionality**: The **SAM2** model segments clothing items from uploaded user images. Below is a diagram showing the process from image upload to segmentation and storage.

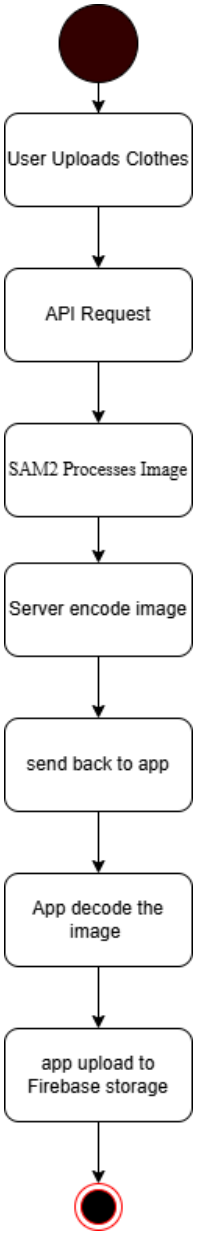


Figure 3.5 Workflow of SAM2

1. **User Uploads Clothing Image**: The user uploads an image containing clothing to the app.
2. **API Request**: The app sends a request to the backend for segmentation.
3. **SAM2 Model Processes Image**: The backend calls the **SAM2** model to segment the clothing from the background.
4. **SAM2 encode image**: Cannot send file, so need to encode the image.
5. **When send it back, the app need to decode**

6. **Result Storage**: The segmented image and metadata are saved to **Firebase Storage** and **Firestore** for future retrieval.

KolorVTO Model (Try-On Image Generation)

- **Functionality**: The **KolorVTO** model generates try-on images by mapping the segmented clothing onto the user's photo. Below is a diagram showing how this works.

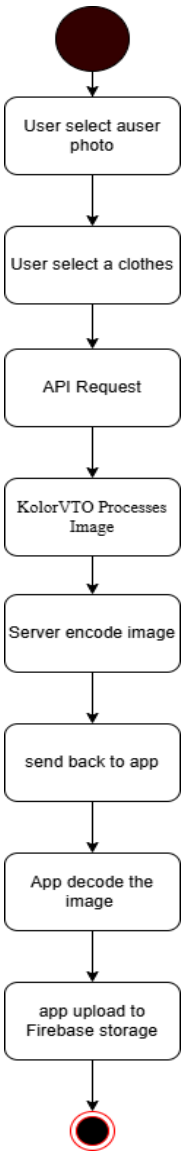


Figure 3.6 Workflow of KolorVTO

1. **User Selects Clothing and Uploads Photo**: The user selects a clothing item and uploads a full-body image.

2. **API Request**: The app sends the full-body image and the segmented clothing image to the backend.
3. **KolorVTO Model Processes Images**: The backend calls the **KolorVTO** model to generate a try-on image.
4. **Try-On Image Returned**: The generated try-on image is sent back to the app and displayed to the user.
5. **Result Storage**: The try-on image and metadata are saved to **Firebase Storage** and **Firestore**.

3.3 Timeline

3.3.1 FYP 1

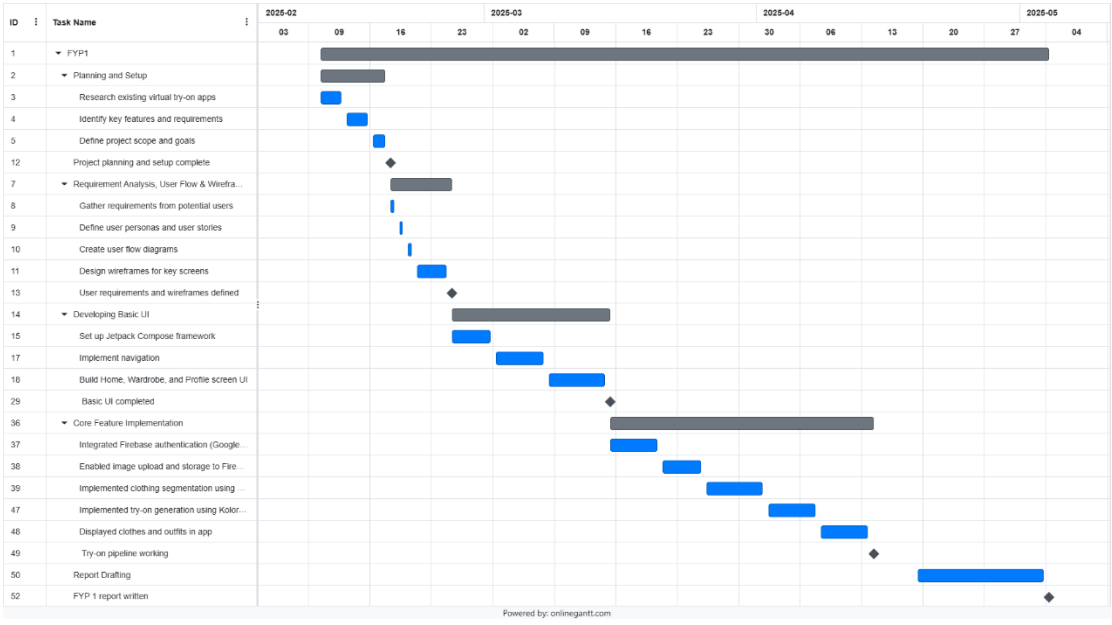


Figure 3.7 FYP1 Timeline

3.3.2 FYP 2

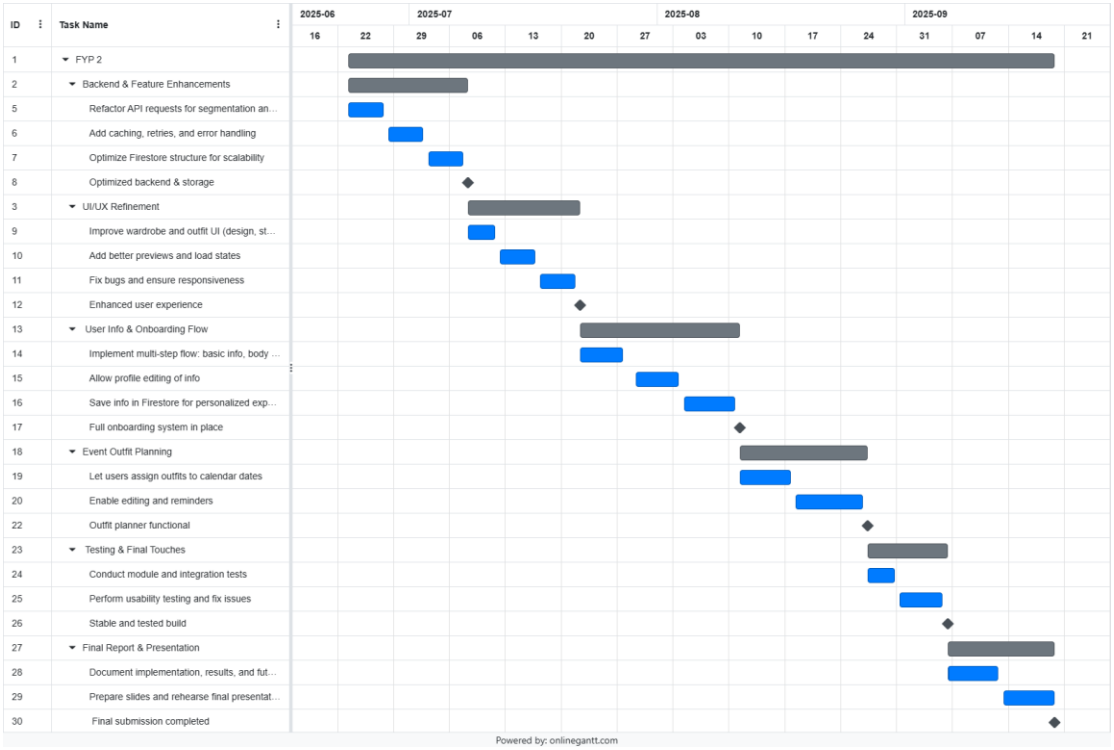


Figure 3.8 FYP2 Timeline

CHAPTER 4

System Design

4.1 System Block Diagram

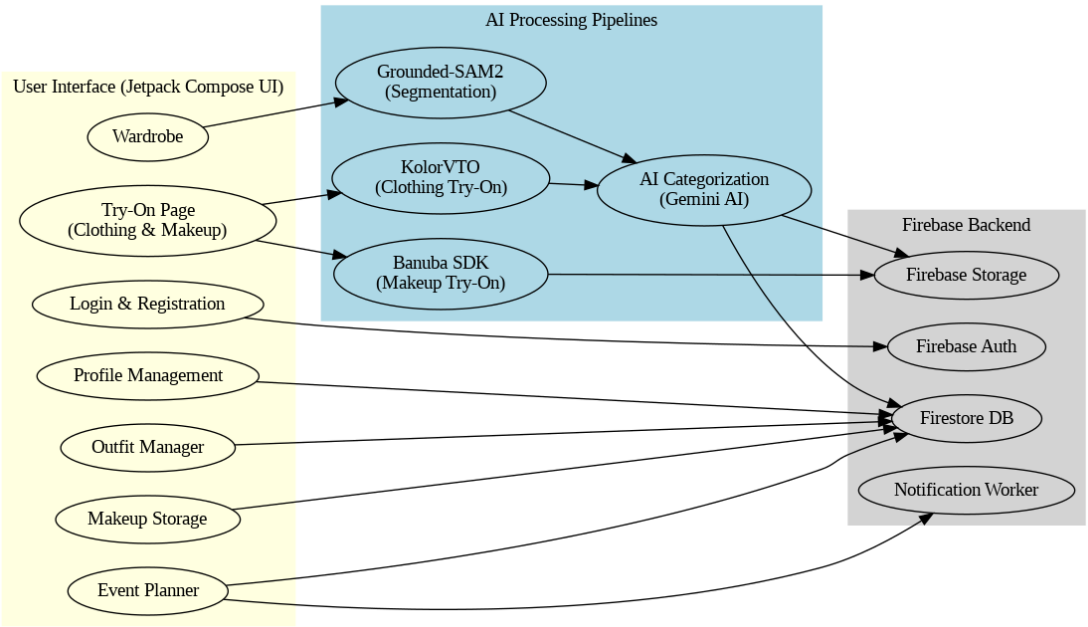


Figure 4.1 System Block Diagram

The block diagram illustrates the modular architecture and end-to-end data flow of the Virtual Try-On mobile application. The process begins with the Authentication Module, which manages secure login and registration using Firebase Authentication. Once authenticated, new users are guided through the Onboarding Flow, where personal details such as full name, date of birth, and style preferences are collected. These details are stored in Firebase Firestore, forming the basis for a personalized experience. Existing users can later access this information through the Profile Management Module, which retrieves data from Firestore, displays it within the app, and allows users to make edits that are automatically updated in real time.

The Wardrobe Module enables users to upload raw images of clothing items. These images are processed through the Segmentation Pipeline, which integrates GroundingDINO for clothing region detection and SAM2 for precise segmentation. Postprocessing ensures clean, user-confirmed outputs, which are then transmitted to the AI Categorization Server. This server performs feature extraction to classify the clothing based on attributes such as type, color, and style. The processed images are

stored in Firebase Storage, while the categorized metadata is stored in Firestore, ensuring synchronization between visual and structured information.

The Try-On Module supports both Clothing Try-On and Makeup Try-On. For clothing, users select items from their wardrobe, and the system retrieves the corresponding segmented images from Firebase Storage. These, along with the user's photo, are sent to the KolorVTO pipeline, which generates photorealistic try-on previews. The generated results are passed to the AI Categorization Server, which extracts additional metadata and stored the data and the result to the Firebase. The results are then displayed on the Outfit Storage Page, where users can view, manage, and reuse their try-on outcomes. For makeup, the Banuba SDK applies real-time AR effects such as lipstick, foundation, or eyeshadow directly to the user's face. The generated outputs are saved to Firebase Storage and displayed in the Makeup Storage Page, where they can be retrieved with corresponding details.

The Event Planner extends this functionality by letting users assign saved outfits to upcoming events. Event details, including linked outfits, are stored in Firestore, while reminders are triggered through the Notification Worker to ensure timely user engagement.

From a data flow perspective, the system follows a structured cycle:

1. User inputs (login, personal info, clothing uploads, makeup selections).
2. Processing (segmentation via GroundingDINO + SAM2, try-on generation via KolorVTO, AR makeup via Banuba, and categorization via the AI server).
3. Storage and retrieval through Firebase (Firestore for structured metadata and scheduling, Storage for images and try-on results).
4. Presentation through a Jetpack Compose-based UI, which displays personalized content across wardrobe, outfit, makeup, and event modules.

The backend integration ensures that Firebase functions as the central hub for both structured and unstructured data, while AI servers handle heavy computational tasks like segmentation, categorization, and try-on generation. This modular and interconnected design guarantees stability, scalability, and a seamless user experience across all components of the system.

4.2 Module Design

Authentication Module

The Authentication Module handles secure user registration and login. It supports email/password, authentication via Firebase Authentication. This module manages user sessions, validates login states, and interacts with Firestore to maintain user account records. Key classes include LoginActivity, RegisterActivity, and corresponding ViewModel classes, which coordinate with Firebase methods such as signInWithEmailAndPassword() and signInWithCredential(). The module provides input validation, error handling, and redirects users upon successful login.

Wardrobe Management Module

The Wardrobe Management Module allows users to upload clothing images from their device or camera. Uploaded images trigger the **Image Segmentation Pipeline**. The module uses WardrobeViewModel and WardrobeScreen (Jetpack Compose UI) to display wardrobe contents. Confirmed segmented images, along with metadata generated by the AI Categorization Server, are stored in Firebase Storage and Firestore to form the user's digital wardrobe.

Image Segmentation Pipeline Module

This module processes uploaded clothing images through multiple stages:

1. **Preprocessing**: resizing, normalization.
2. **GroundingDINO**: detects clothing regions with bounding boxes.
3. **SAM2**: generates precise segmentation masks for each clothing item.
4. **Postprocessing**: refines masks and creates transparent PNGs. Users confirm the segmented images before they are forwarded to the AI Categorization Module.

4.2.4 AI Categorization Module

The AI Categorization Module analyzes segmented clothing and AI-generated outfit images to extract structured metadata. Attributes include clothing type, category, color, pattern, style tags, and additional properties such as sleeve length or collar type. The module receives inputs through REST APIs, processes images using Python scripts, and outputs JSON documents stored in Firestore along with the corresponding images

in Firebase Storage. This ensures all items are searchable, filterable, and consistently organized.

Clothing Try-On Module (KolorVTO)

The Clothing Try-On Module allows users to select wardrobe items and upload their photos to generate photorealistic try-on images using KolorVTO. Results are stored in Outfit Storage, and metadata is sent to the AI Categorization Server for consistency. The module integrates with TryOnScreen and TryOnViewModel to manage image display and user interactions.

Makeup Module (Banuba SDK)

The Makeup Module uses the Banuba SDK to provide real-time AR makeup application. Users can try lipstick, eyeshadow, eyeliner, and preset looks. Makeup results are saved in Makeup Storage, and associated metadata, such as applied effects and base skin tone, is stored in Firestore. This allows users to reapply or edit makeup results independently of clothing try-ons.

Outfit Management Module

The Outfit Management Module provides an interface to view, edit, delete, rename, and combine saved outfits. Metadata from AI Categorization is used to enable searching and filtering. Users can mark favorites and reuse previously generated try-on images. The module interacts with Outfit Storage for image management.

Makeup Storage Module

The Makeup Storage Module stores saved makeup looks separately from outfits. It allows users to quickly reapply, edit, or share previously saved makeup results. Metadata is synchronized with Firestore to maintain consistency and enable search/filtering functionality.

Event Management Module & Notifications

This module allows users to schedule events and attach saved outfits and makeup looks. Event information, including Storage URLs for selected looks, is saved in Firestore.

The integrated Notification Worker, implemented via Android WorkManager, schedules local push notifications to remind users of upcoming events.

Profile Module

The Profile and Settings Module allows users to view and update profile information, and style preferences. All updates are synchronized with Firestore to maintain data consistency across the application.

Firebase Backend

The Firebase Backend acts as the core infrastructure supporting authentication, structured data storage, and secure image storage. It integrates all modules, including Wardrobe, Try-On, AI Categorization, Makeup, Outfit Management, and Event Planner, ensuring a scalable, efficient, and seamless experience for the user.

4.3 Database Design and Firebase Schema

This application uses **Google Cloud Firestore** as the primary NoSQL database for managing user data, wardrobe metadata, outfit information, makeup info and event planning. In addition, **Firebase Storage** is used to store all binary image files, such as uploaded clothing, AI-generated try-on images and makeup looks. This integrated setup allows for scalable, secure, and efficient handling of both structured metadata and large media assets

4.3.1 Firestore Data Structure Overview

The top-level Firestore collection is users, where each document is uniquely identified by a user ID (userId). Each user document contains four subcollections:

- **userdata**: Stores personal user information such as name, email, date of birth and style preferences.
- **wardrobe**: Each user document has a subcollection of clothing items, storing image URLs, segmentation masks, and AI-generated metadata (type, style, color, pattern, attributes).

- **outfits**: Contains saved try-on results, linking images in Outfit Storage with associated metadata extracted by AI Categorization.
- **events**: Stores event details, including date/time, linked outfit/makeup references, and notification settings.

4.3.2 Firebase Storage Data Structure Overview

Firebase Storage is used for managing image files. Each uploaded clothing item and AI-generated outfit image is stored in user-specific folders in the following structure:

Table 4.1 Data sturcture

Image Type	Storage Path Format
Clothing Uploads	users/{userId}/wardrobe/{clothingId}.jpg
Outfit Results	users/{userId}/outfits/{outfitId}.jpg
Makeup Results	users/{userId}/makeup/{makeupId}.jpg

These images are referenced in Firestore via the imageUrl field to maintain a lightweight database while supporting rich media content in the app.

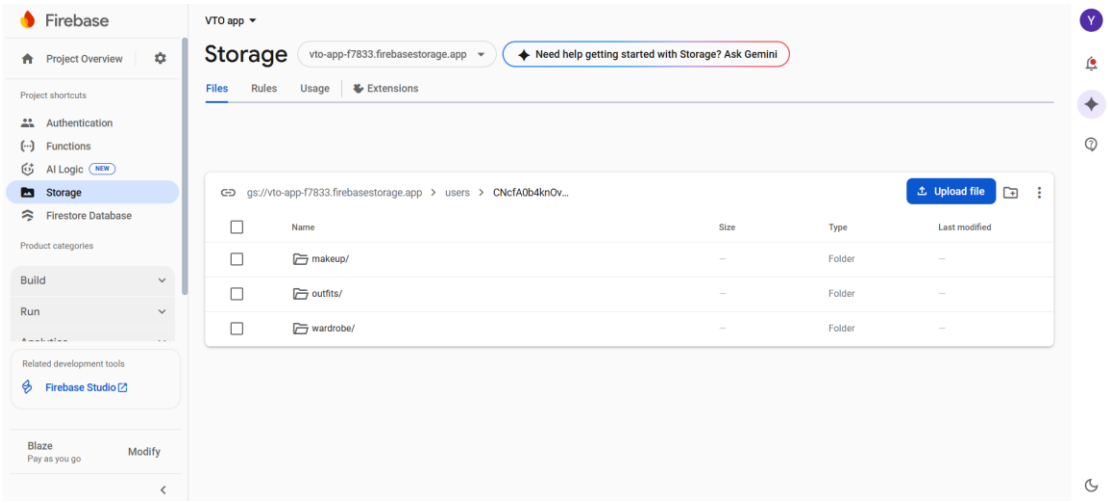


Figure 4.2 Firebase Storage

4.3.3 Firestore Path and Field Design

Users Collection

Firestore Path: users/{userId}/userdata

Field Name	Data Type	Description	Sample Value
fullName	String	User's full name	"john"
dateOfBirth	String	User's date of birth	"16/01/2000"
createdAt	Number	Account creation timestamp	1758142870124
hasCompletedOnboarding	Boolean	Whether onboarding is completed	true
stylePreference	String	Primary style preference	"Party"
stylePreferences	Array[String]	List of style preferences	["Party", "Business", "Wedding"]

Table 4.2 User collection

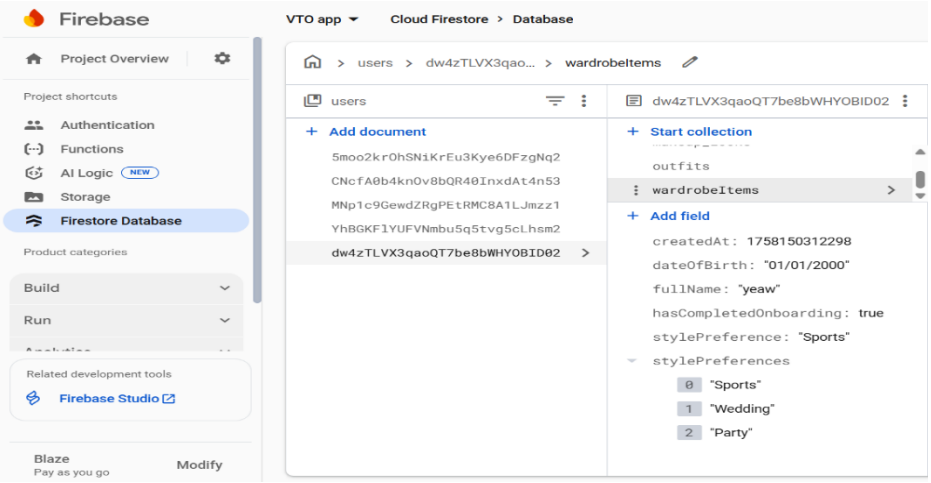


Figure 4.3 User collection

Wardrobe Collection

Firestore Path: users/{userId}/wardrobe

Field Name	Data Type	Description	Sample Value
display_name	String	Descriptive name of the clothing item	"Casual 2"
image_name	String	File name of the uploaded image	"temp_image_1758151248713_294.png"
image_url	String	Firebase Storage URL of the clothing image	"https://firebasestorage.googleapis.com/.../wardrobe/3f011fd8-..."
metadata	Map	Clothing attributes	{category: "top", color: "dark green", pattern: "text print", style: "hooded top", occasion: "Casual", description: "A dark green long-sleeved hooded top..."}
timestamp	String	Upload time	"2025-09-17T23:21:04.188491"

Table 4.3 Wardrobe collection

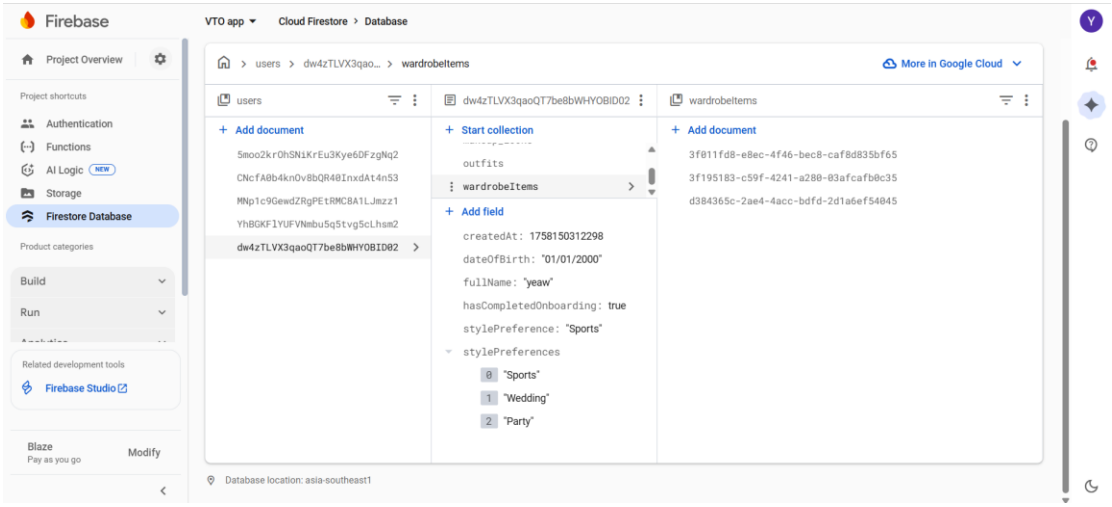


Figure 4.4 Wardrobe collection

Outfits Collection

Firestore Path: users/{userId}/outfits

Field Name	Data Type	Description	Sample Value
image_url	String	Firebase Storage URL of the outfit image	"https://firebasestorage.googleapis.com/.../outfits/de9134a6-..."
isFavorite	Boolean	Whether the outfit is marked as favorite	false
metadata	Map	Outfit attributes and description	{color: "Olive Green", description: "A person is wearing a long-sleeved olive green hooded top...", occasion: "Loungewear / Home", style: "Relaxed and casual loungewear.", outfit_name: "hoody"}
timestamp	String	Creation time	"2025-09-17T23:23:45.452570"

Table 4.4 Outfit collection

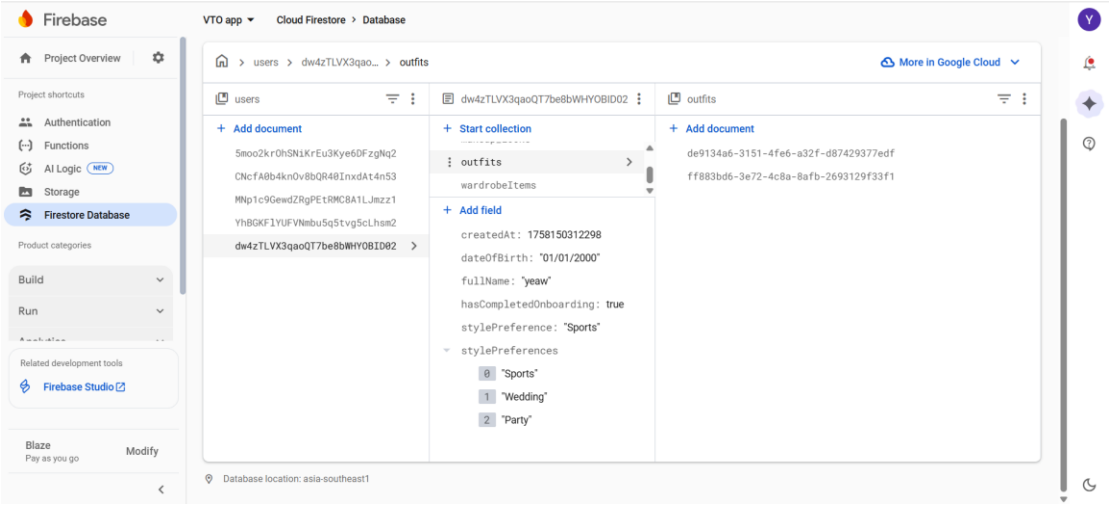


Figure 4.5 Outfit collection

Makeup Collection

Firestore Path: users/{userId}/makeups

Field Name	Data Type	Description	Sample Value
id	String	Unique makeup identifier	"3a28356a-1871-4a59-af2f-6ba686ac5bad"
imageUrl	String	Firebase Storage URL of the makeup result	"https://firebasestorage.googleapis.com/.../makeup/3a28356a-..."
name	String	Name of the makeup look	"look 1"
dateCreated	Timestamp	Creation date/time	"September 18, 2025 at 8:07:04 AM UTC+8"
likesCount	Number	Number of likes	0
public	Boolean	Visibility flag	false
stability	Number	AR effect stability score	0
userId	String	Owner reference	"dw4zTLVX3qaoQT7be8bWHYOBID02"

Table 4.5 Makeup collection

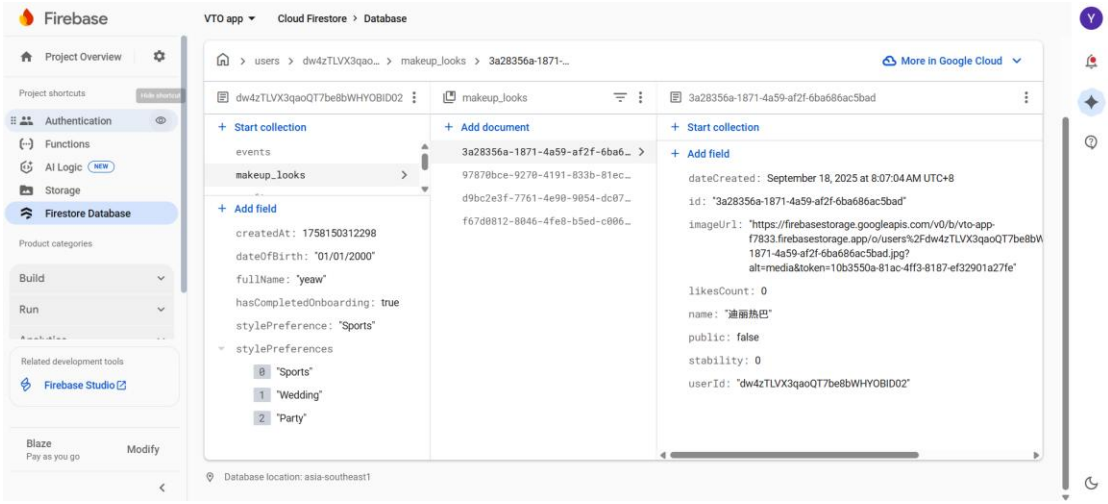


Figure 4.6 Makeup collection

Events Collection

Firestore Path: users/{userId}/events

Field Name	Data Type	Description	Sample Value
id	String	Unique event identifier	"3Iu09TYO7Tv5rR42cMYx"
title	String	Event title	"oo"
date	String	Event date	"2025-09-18"
time	String	Event time	"10:59"
allDay	Boolean	All-day event flag	false
eventType	String	Type of event	"WEDDING"
description	String	Optional event description	""
location	String	Optional event location	""

plannedOutfitId	**String / Null**	**Linked outfit reference**	**null**
outfitImageUrl	**String / Null**	**URL of linked outfit**	**null**
reminderEnabled	**Boolean**	**Whether reminders are active**	**true**
reminderTime	**Number**	**Minutes before event to notify**	**1**
createdAt	**Number**	**Event creation timestamp**	**1758160794146**

Table 4.6 Event collection

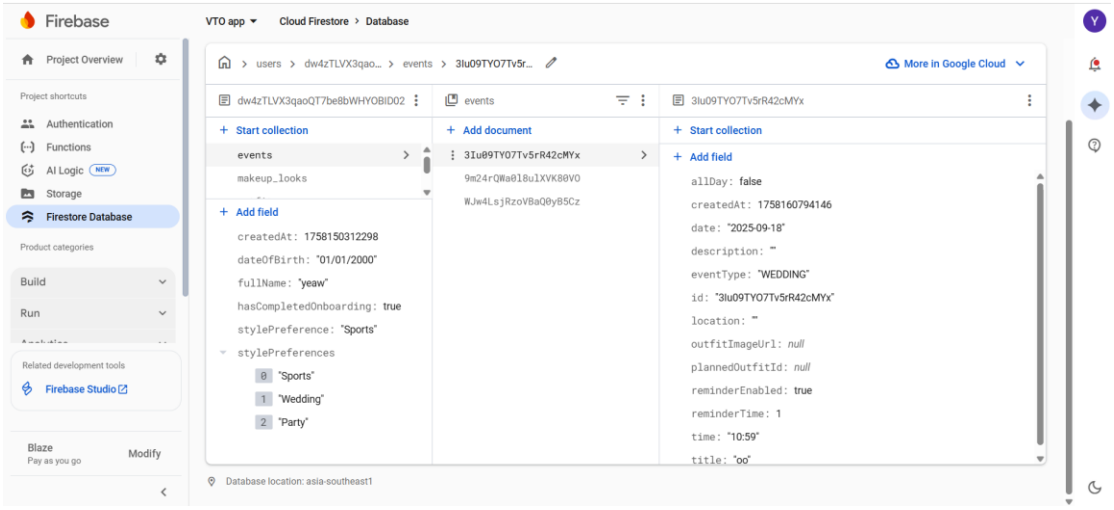


Figure 4.7 Event collection

Firestore offers real-time synchronization and scalable document-based storage, which is well-suited to user-centric data like clothes and events. Firebase Storage complements this by hosting high-resolution media files without bloating the database. This separation of concerns ensures fast access, efficient storage, and easier file management.

4.4 Model Selection and Architecture

4.4.1 GroundingDINO + SAM2 (Segmentation Pipeline)

Our system first applies a two-stage segmentation pipeline to isolate garments in the user's photo. GroundingDINO is used to detect clothing items with open-vocabulary prompts, outputting high-quality bounding boxes and labels. These detections are then passed to Meta's Segment Anything Model (SAM2), which generates precise pixel-level masks for each garment. This combination was chosen because it leverages the strengths of each model – GroundingDINO's zero-shot object detection and SAM2's accurate segmentation – to build a robust pipeline for complex masking tasks. The resulting segmented garment images and masks are saved in cloud storage, with metadata (labels, bounding box coordinates) recorded in the database for later use.

4.4.2 KolorVTO (Virtual Try-On Model)

For generating the final try-on images, we use the Kolors Virtual Try-On (KolorVTO) engine. The app supplies the user's photo and the segmented garment image to KolorVTO, which synthesizes a realistic image of the user wearing that clothing. In practice, the system base64-encodes the person and garment images and sends them via a JSON API request to the KolorVTO service. The service returns a job identifier and, upon completion, a composite image. This try-on image is then retrieved by the app; it is stored in Firebase Storage and indexed in Firestore along with any relevant parameters (such as the random seed or processing status) so that it can be displayed or queried later.

4.4.3 Banuba SDK (Makeup Try-On Module)

We integrate Banuba's AR Face SDK to provide real-time virtual makeup. Banuba's SDK performs fast, precise face tracking and builds a 3D face mesh in real time, allowing digital cosmetics (lipstick, eyeshadow, blush, etc.) to be overlaid correctly onto the user's facial features. Underlying neural networks segment fine facial regions (lips, cheeks, skin, etc.) at the pixel level so that makeup effects align accurately on any skin tone. When the user applies virtual makeup, the resulting augmented image (e.g. a camera frame with the effects) is captured and saved. The captured makeup-try-

on images are uploaded to Firebase Storage, and their metadata (chosen filters, timestamp, etc.) are recorded in Firestore for later retrieval.

4.4.4 Google Gemini (AI Categorization Model)

Finally, we use Google's Gemini multimodal AI to categorize clothing and outfit images. Gemini supports zero-shot object detection and visual understanding, meaning we can send it an image and ask open-ended questions about it. In our system, each segmented garment image and each generated try-on outfit image is sent to Gemini with prompts requesting descriptive attributes. Gemini returns labels and descriptions, which we parse into structured metadata. These attributes are stored in Firestore alongside references to the images, while the image files remain in Firebase Storage. In this way, Gemini's output provides searchable metadata (type, color, style, etc.) for each item, integrated with our database to support product categorization and user queries.

CHAPTER 5

System Implementation

5.1 Hardware Setup

To build and test the Android-based Virtual Try-On application, the following hardware components are required:

Laptop (Development)

Table 5.1 Specifications of laptop

Description	Specifications
Model	Nitro AN515-45
Processor	AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
Operating System	Windows 11
Graphic	NVIDIA GeForce RTX 3060 Laptop GPU
Memory	16GB RAM
Storage	475GB

Mobile Device for Testing

Table 5.2 Specifications of mobile device

Description	Specifications
Model	HONOR X9b 5G
Processor	Snapdragon 6 Gen 1
Operating System	Android 12
Memory	8GB RAM
Storage	256GB

5.2 Software Setup

The software setup for the Virtual Try-On (VTO) system required different tools for mobile development, backend AI processing, and cloud service integration. Each component was installed and configured separately to ensure modularity and smooth system integration.

Android Development Tools

The Android application was developed using Android Studio as the primary IDE, with Java Development Kit (JDK), Gradle, and the Android SDK with Emulator Tools. These tools were used to build, test, and run the Jetpack Compose–based mobile interface.

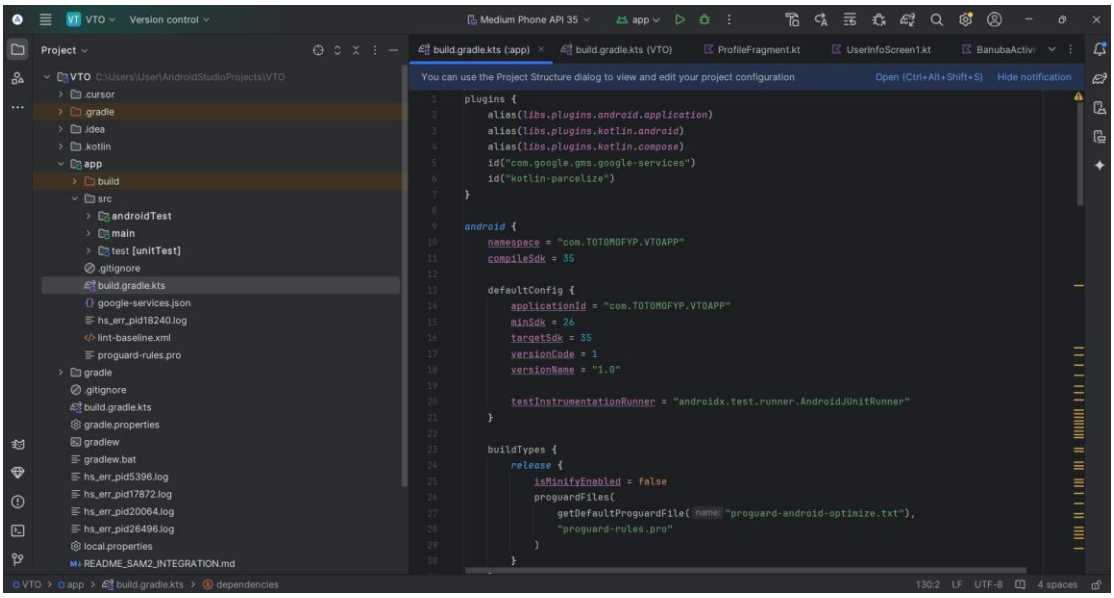


Figure 5.1 Android Studio setup

Firebase Integration

Firebase was configured to handle Authentication, Firestore Database, and Storage. The google-services.json file was downloaded from the Firebase Console and added to the application project to enable backend connectivity. This ensured that user login, clothing metadata, and wardrobe images were synchronized between the app and the server.

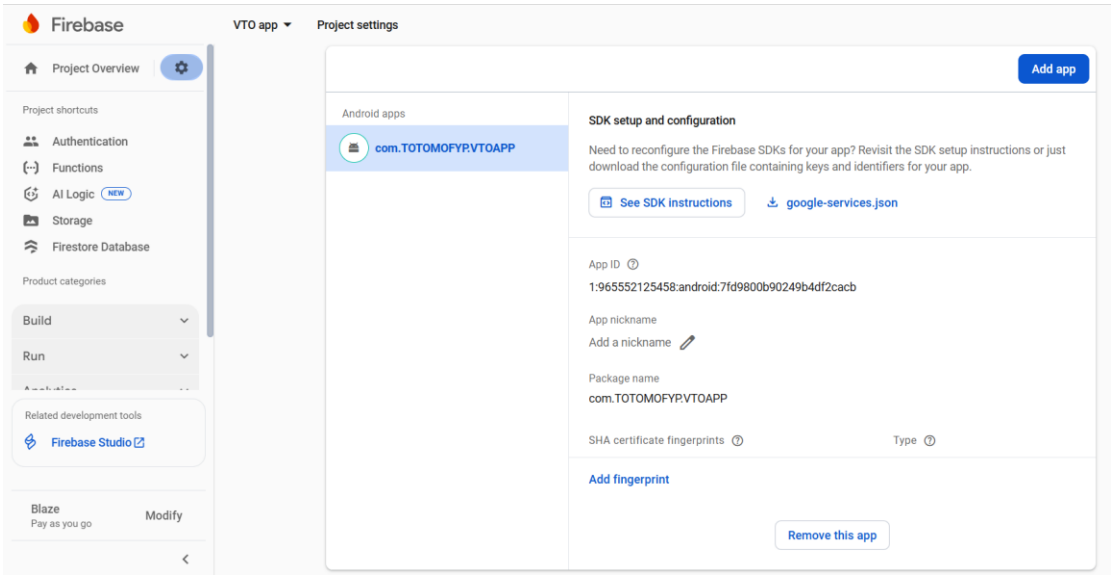


Figure 5.2 Integration of Firebase

Banuba SDK for Makeup Try-On

The Banuba SDK was integrated into the mobile app to provide AR-based makeup try-on functionality. It enabled real-time rendering of lipstick, eyeshadow, and other effects on the user's face. The SDK required license activation and was embedded within the Android Studio project.

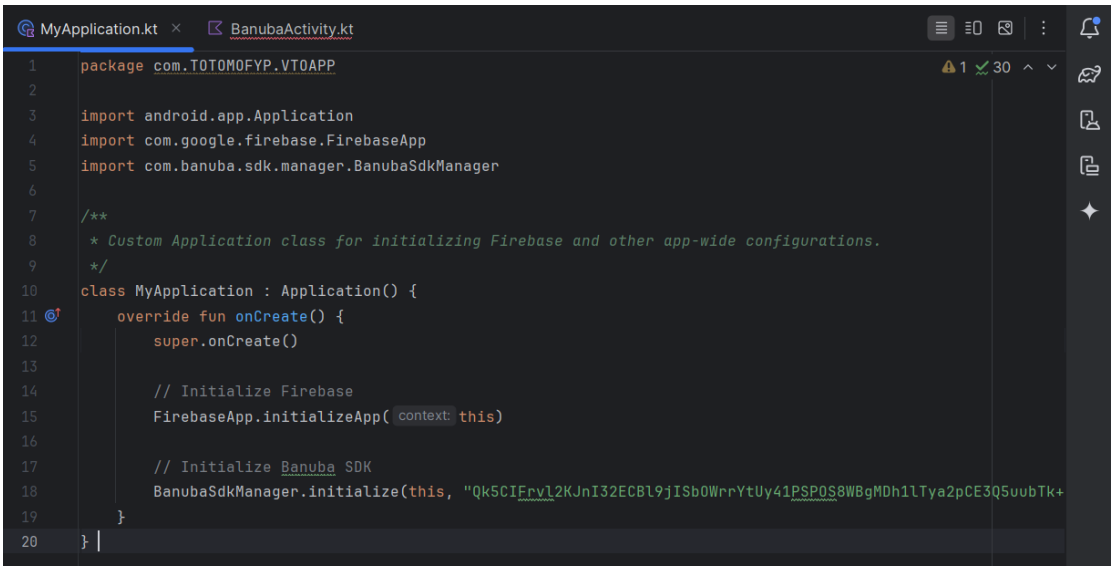


Figure 5.3 Integration of Banuba SDK

AI Backend – env1 (Segmentation & Try-On)

A dedicated Anaconda environment, named env1, was created to run the core segmentation and try-on pipeline. This included installing dependencies such as PyTorch, Torchvision, OpenCV, Transformers, and FastAPI. Within this environment:

- Grounding DINO was used for clothing detection.
- SAM2 was applied to generate precise segmentation masks.

The FastAPI server exposed these functionalities, and **Ngrok** was used to provide a secure public API endpoint for mobile app integration.

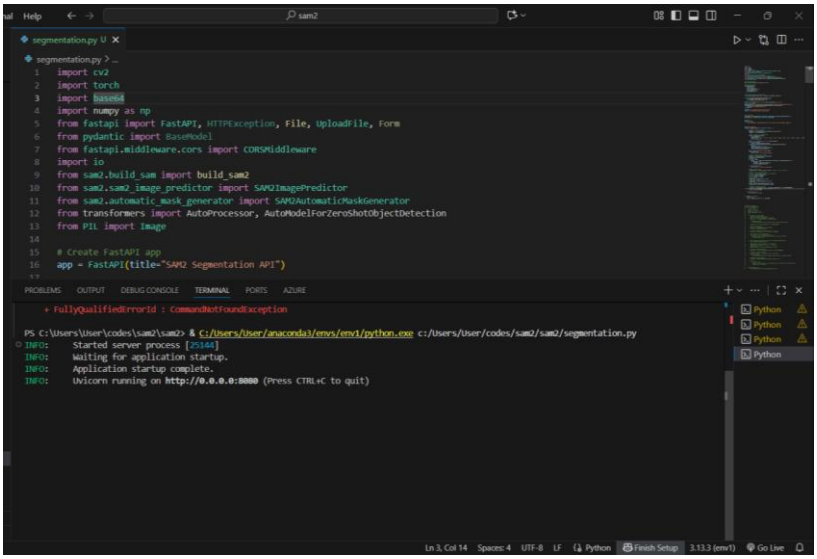


Figure 5.4 Anaconda environment (env1) for Grounded Sam2

KolorsVTO model

Instead of a local installation, the KolorsVTO model is accessed via Kling AI's platform. This service generates try-on images by combining user-uploaded photos with clothing items stored in the wardrobe. Integration with Kling AI is achieved through API calls, where the segmented clothing (from Anaconda env1) and user photo are sent to the KolorsVTO endpoint. The processed output is then returned to the app. Since KolorsVTO is a hosted service, no additional local environment is required beyond configuring API access.

AI Backend – .venv (Gemini Categorization & Outfit Management)

A second environment, named **.venv**, was created specifically for the **Gemini AI backend**. This FastAPI microservice handled clothing categorization, metadata extraction (type, color, style, occasion), and smart naming for wardrobe management. Separating this environment from segmentation ensured better stability and reduced dependency conflicts.

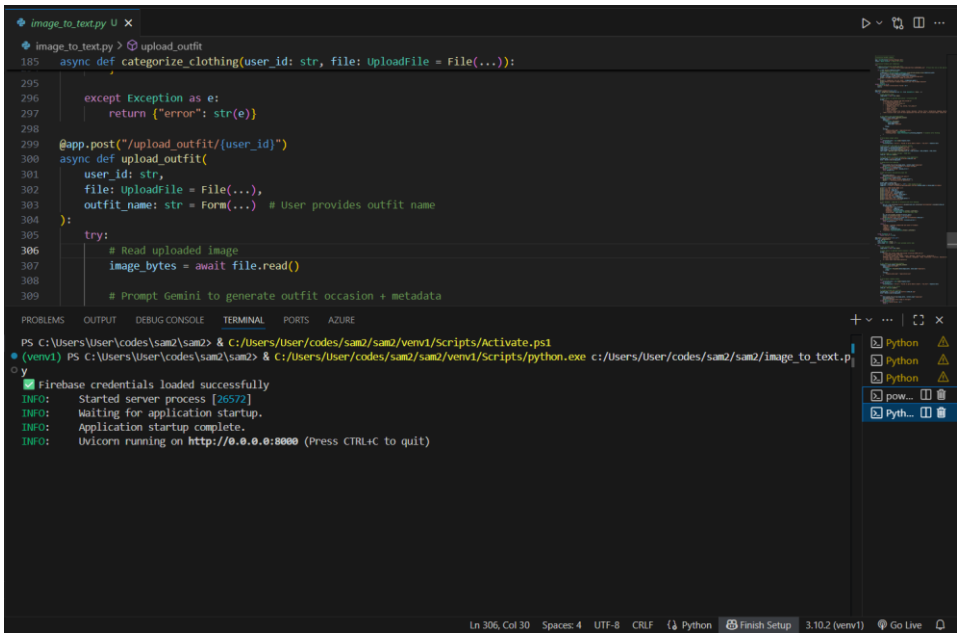


Figure 5.5 Gemini AI categorization backend (.venv) with Firebase

5.3 Settings and Configuration

After setting up the core software components, additional configuration steps were carried out to ensure proper integration. In Android Studio, the google-services.json file downloaded from Firebase was added to the app/ folder, linking the project with Firebase services. Gradle dependencies were configured for Firebase SDKs, Banuba SDK, and networking libraries. Figure 5.x presents the Android Studio configuration with Firebase integration.

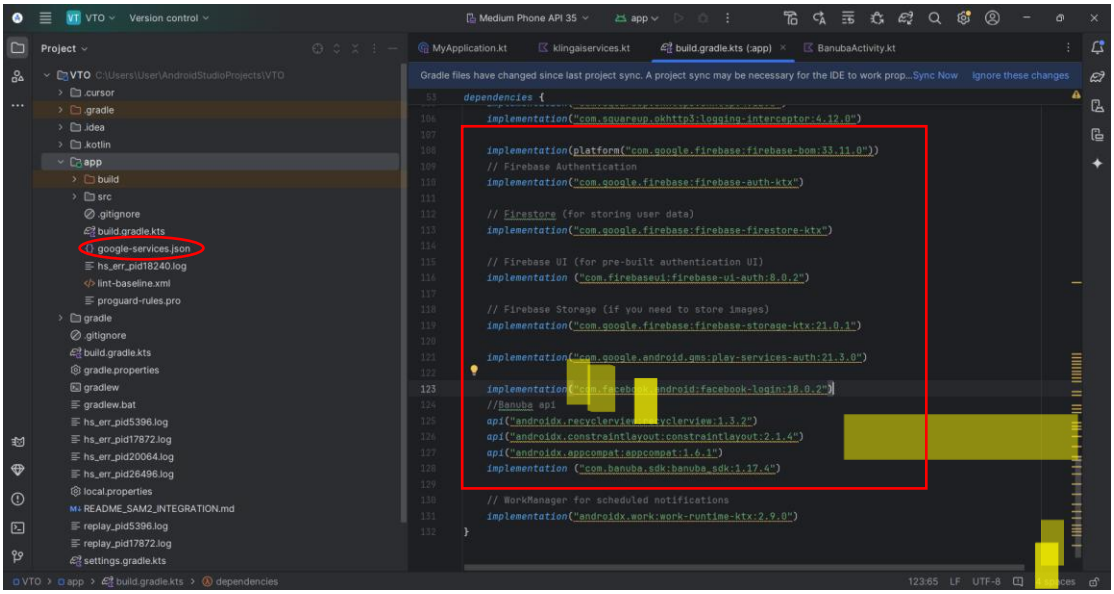


Figure 5.6 Gradle dependencies for Firebase and Banuba SDK.

In the Firebase Console, **Authentication** was enabled with both Email/Password and Google Sign-In, while **Firestore Database** and **Firebase Storage** were configured for structured and unstructured data storage. This ensured seamless synchronization between metadata (e.g., clothing categories) and media files (e.g., segmented clothes, try-on outputs). Figure 5.x shows the Firebase Console setup.

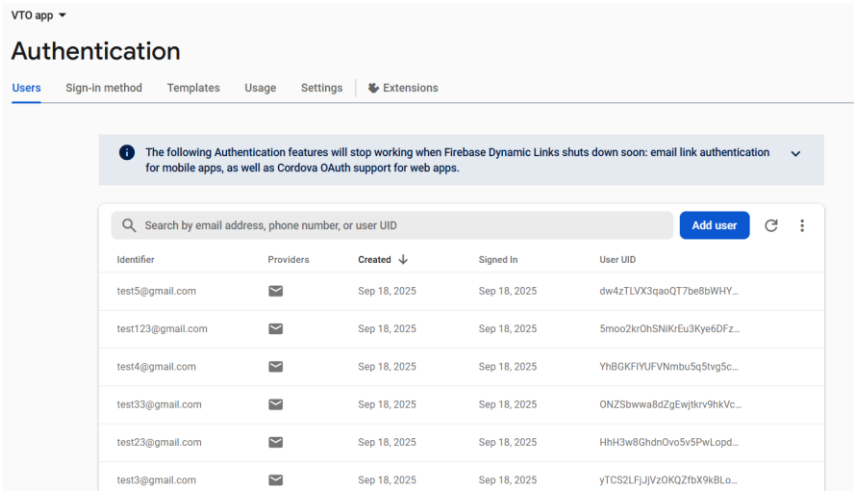


Figure 5.7 Firebase Authentication

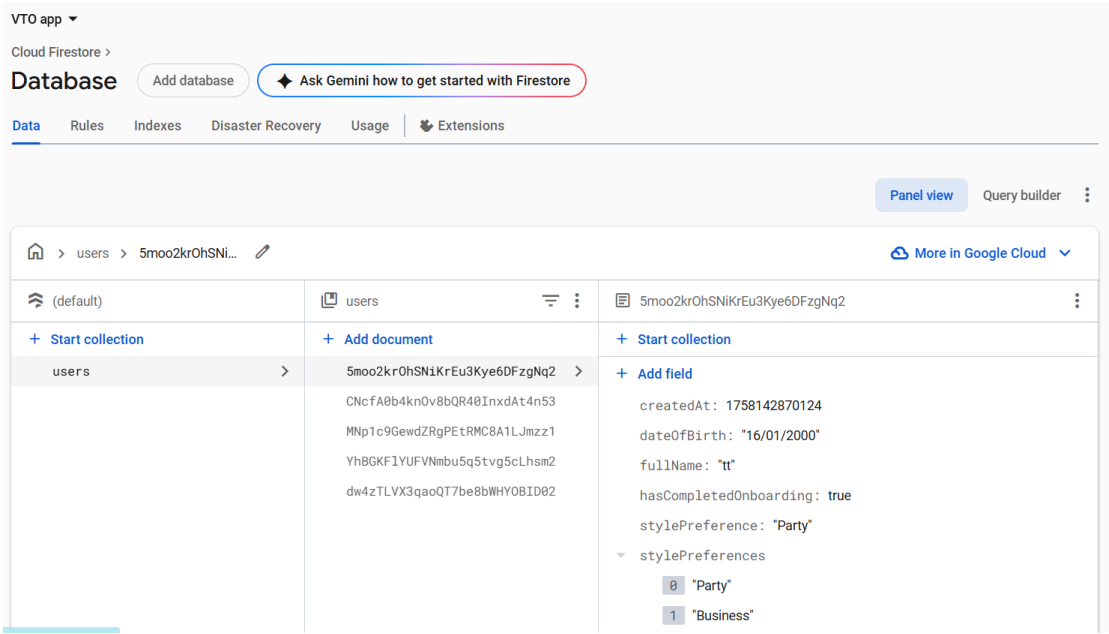


Figure 5.8 Firestore

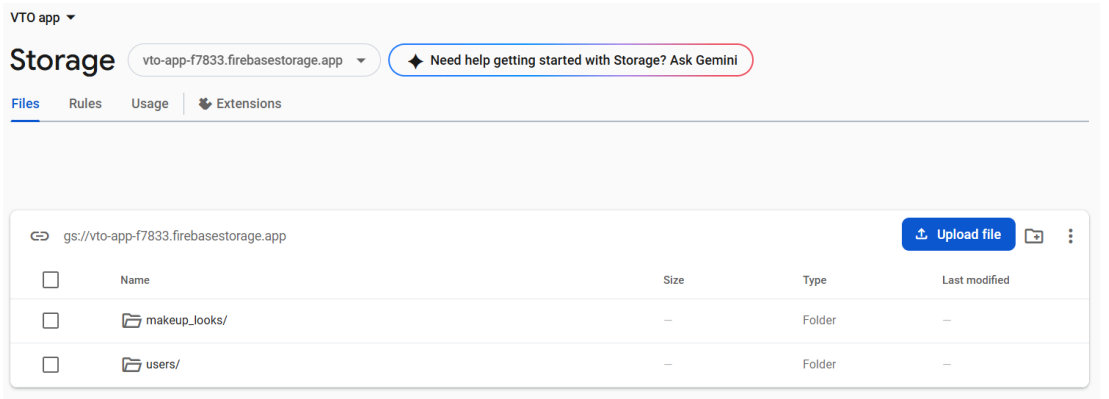
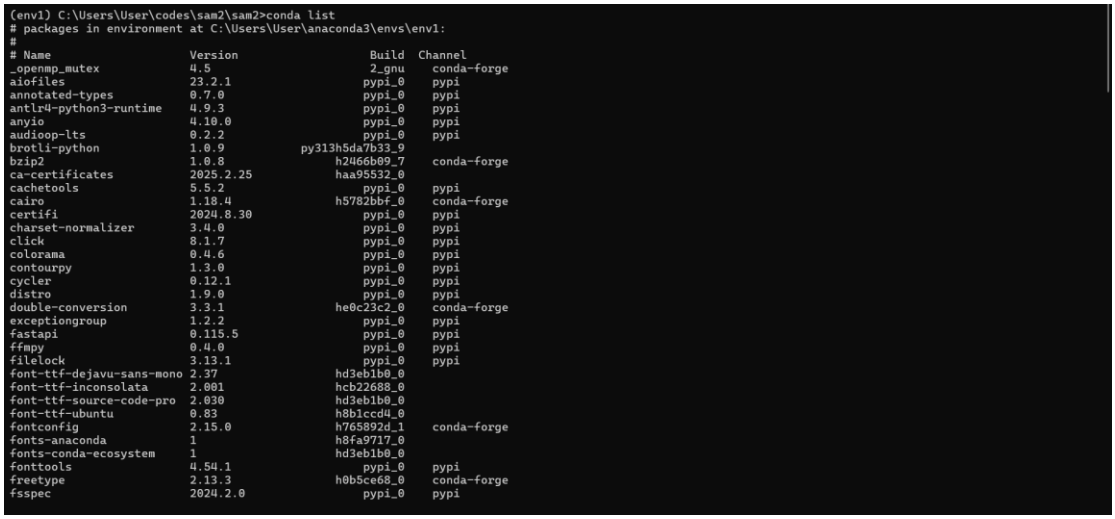


Figure 5.9 Firebase Storage

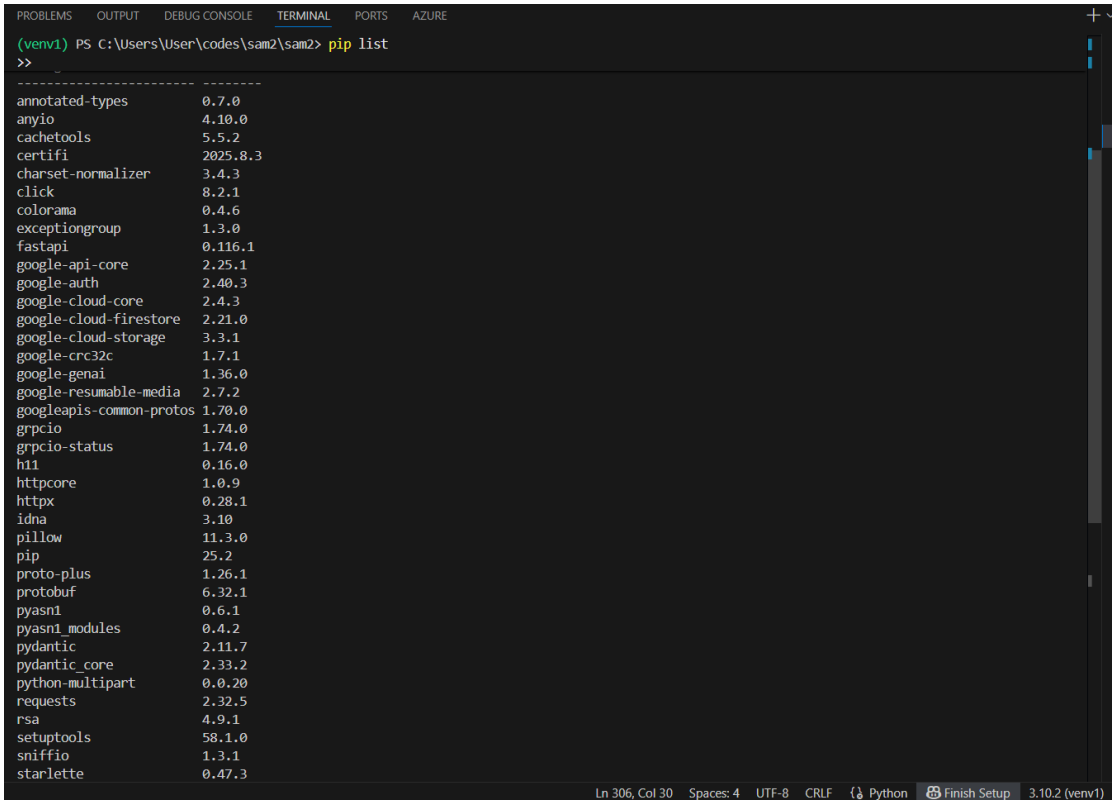
In **Anaconda env1**, dependencies such as PyTorch, TorchVision, and OpenCV are installed to support the **Grounding DINO** and **SAM2 models**. The segmentation service is initiated by executing segmentation.py, which loads the models and provides API endpoints for the mobile application.



```
(env1) C:\Users\User\codes\sam2\sam2>conda list
# packages in environment at C:\Users\User\anaconda3\envs\env1:
#
# Name                    Version                   Build  Channel
_openmp_mutex             4.5                       2_gnu    conda-forge
aiofiles                  23.2.1                   pypi_0    pypi
annotated-types           0.7.0                    pypi_0    pypi
antlr4-python3-runtime    4.9.3                    pypi_0    pypi
anyio                     4.10.0                   pypi_0    pypi
audioop-lts               0.2.2                    pypi_0    pypi
brotli-python             1.0.9           py313h5da7b33_9
bzip2                     1.0.8                h2466b09_7    conda-forge
ca-certificates           2025.2.25            haa95532_0
cachetools                5.5.2                    pypi_0    pypi
cairo                     1.18.4               h5782bbf_0    conda-forge
certifi                   2024.8.30                pypi_0    pypi
charset-normalizer        3.4.0                    pypi_0    pypi
click                     8.1.7                    pypi_0    pypi
colorama                  0.4.6                    pypi_0    pypi
contourpy                 1.3.0                    pypi_0    pypi
cycler                    0.12.1                   pypi_0    pypi
distro                    1.9.0                    pypi_0    pypi
double-conversion         3.3.1                he0c23c2_0    conda-forge
exceptiongroup            1.2.2                    pypi_0    pypi
fastapi                   0.115.5                  pypi_0    pypi
ffmpy                     0.4.0                    pypi_0    pypi
filelock                  3.13.1                   pypi_0    pypi
font-ttf-dejavu-sans-mono 2.37                 hd3eb1b0_0
font-ttf-inconsolata      2.001                hcb22688_0
font-ttf-source-code-pro  2.030                hd3eb1b0_0
font-ttf-ubuntu           0.83                 h8b1ccd4_0
fontconfig                2.15.0               h765892d_1    conda-forge
fonts-anaconda            1                    h8fa9717_0
fonts-conda-ecosystem     1                    hd3eb1b0_0
fonttools                 4.54.1                   pypi_0    pypi
freetype                  2.13.3               h0b5ce68_0    conda-forge
fsspec                    2024.2.0                 pypi_0    pypi
```

Figure 5.10 Dependency installation in Anaconda env1

In the **Gemini backend (.venv)**, the environment is configured with authentication keys, API SDKs, and supporting libraries for recommendation and categorization. This ensures smooth communication with Google's Gemini API.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE

(venv1) PS C:\Users\User\codes\sam2\sam2> pip list
>>
Package                   Version
------------------------- --------
annotated-types           0.7.0
anyio                     4.10.0
cachetools                5.5.2
certifi                   2025.8.3
charset-normalizer        3.4.3
click                     8.2.1
colorama                  0.4.6
exceptiongroup            1.3.0
fastapi                   0.116.1
google-api-core           2.25.1
google-auth               2.40.3
google-cloud-core         2.4.3
google-cloud-firestore    2.21.0
google-cloud-storage      3.3.1
google-crc32c             1.7.1
google-genai              1.36.0
google-resumable-media    2.7.2
googleapis-common-protos  1.70.0
grpcio                    1.74.0
grpcio-status             1.74.0
h11                       0.16.0
httpcore                  1.0.9
httpx                     0.28.1
idna                      3.10
pillow                    11.3.0
pip                       25.2
proto-plus                1.26.1
protobuf                  6.32.1
pyasn1                    0.6.1
pyasn1_modules            0.4.2
pydantic                  2.11.7
pydantic_core             2.33.2
python-multipart          0.0.20
requests                  2.32.5
rsa                       4.9.1
setuptools                58.1.0
sniffio                   1.3.1
starlette                 0.47.3

Ln 306, Col 30  Spaces: 4  UTF-8  CRLF  {} Python  Finish Setup  3.10.2 (venv1)
```

Figure 5.11 Gemini backend environment configuration

The **KolorVTO service from Kling AI** does not require local environment installation but instead relies on **API integration**. The system is configured with Kling AI

credentials and endpoints, enabling seamless transmission of input images and reception of generated try-on results.

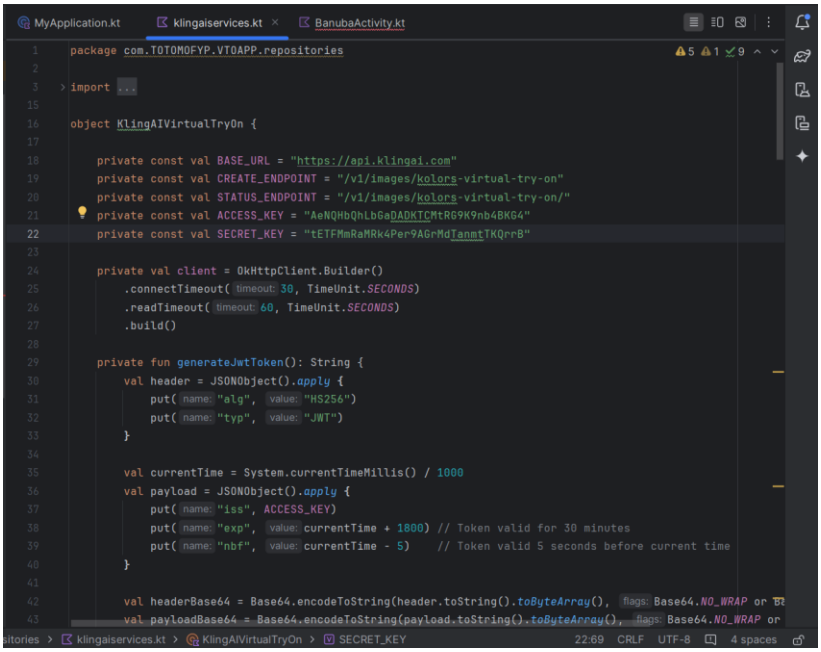


Figure 5.12 Configuration of KolorVTO API

For the **Banuba SDK**, the configuration is done in the **Android Studio project** by importing the SDK dependencies into build.gradle. The SDK is then initialized in the application code with secure API keys to unlock makeup and accessory features.

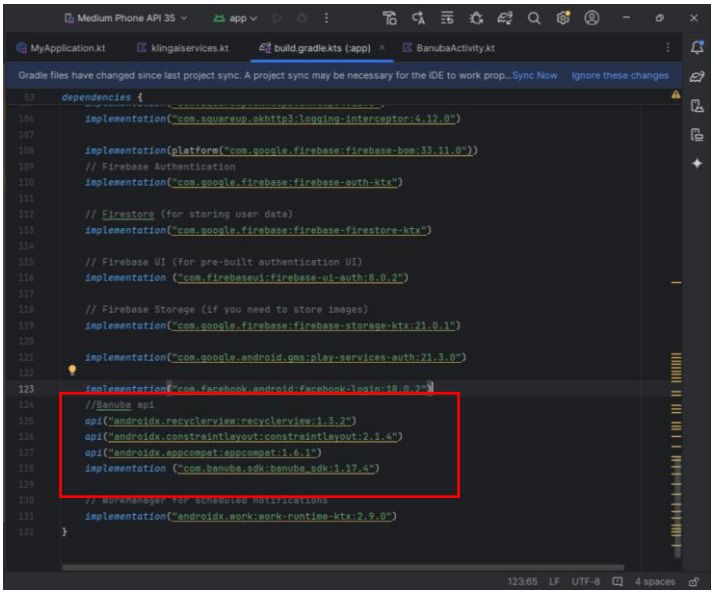


Figure 5.13 Banuba SDK configuration

Lastly, **Ngrok** is set up with authentication and assigned tunnels. The forwarding URL generated by Ngrok is embedded in the mobile application, allowing the app to send requests to locally running services (segmentation and Gemini AI backend).

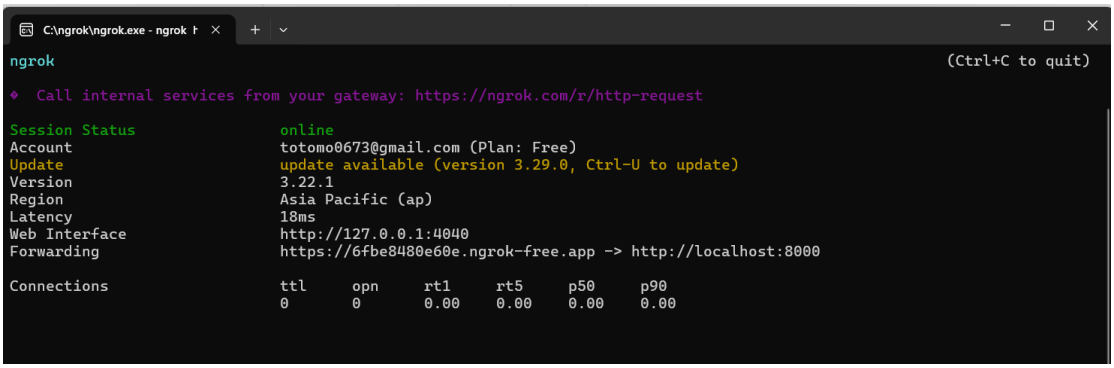


Figure 5.14 Ngrok configuration with the generated forwarding URL

5.4 System operation

5.4.1 User Authentication System

The user authentication system was implemented using Firebase Authentication, enabling users to register and log in securely with their email and password. The Firebase Authentication service was integrated into the Android application through the Firebase SDK and the configuration file google-services.json, which links the mobile app to the Firebase project.

In this implementation, users are able to register an account by providing their email and password. Upon successful registration, Firebase creates a new user profile, and the app immediately redirects the user to the onboarding flow. Similarly, during the login process, registered users can enter their credentials, which are verified by Firebase. If authentication is successful, the system performs an onboarding status check by querying Firestore to determine whether the user has completed the personal information entry required during the onboarding process.

If the onboarding is incomplete, the user is redirected to the onboarding screen to fill in their personal details. Once completed, the information is stored in Firebase Authentication and linked to the user's account. If the onboarding is already completed, the system bypasses this step and directly loads the main application interface. This integration ensures a seamless flow from registration and login to onboarding, providing a secure and personalized entry point into the Virtual Try-On application.

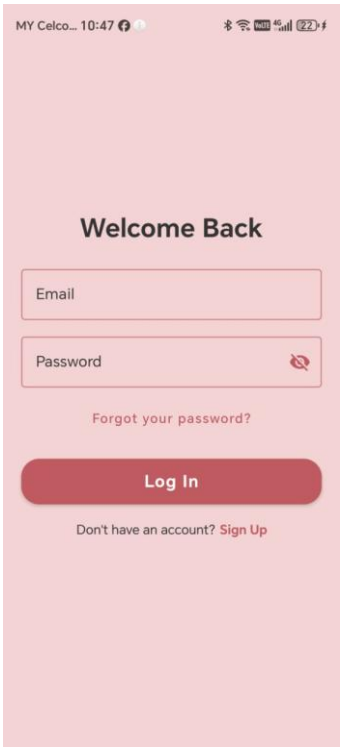


Figure 5.15 Login Screen

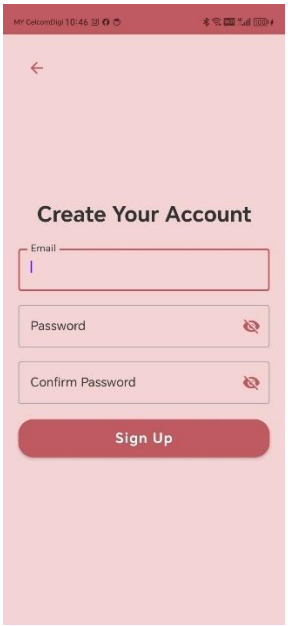


Figure 5.16 Registration Screen

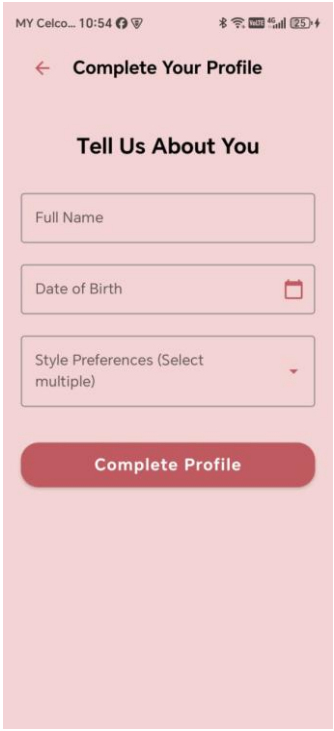


Figure 5.17 Onboarding Redirection Screen

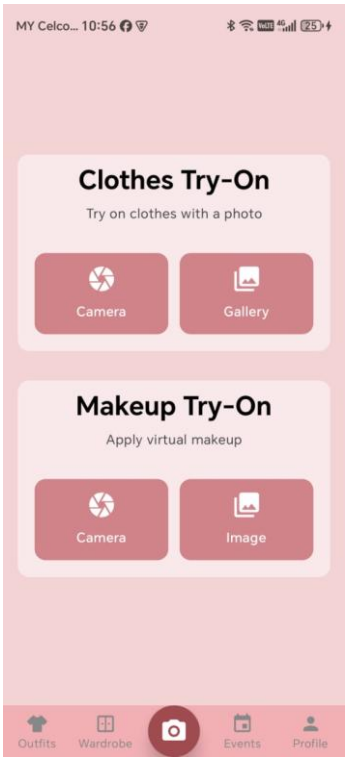


Figure 5.18 Main Screen

5.4.2 Wardrobe Management and Clothing Segmentation

Overview of the Process

In this section, the Clothing Segmentation process in the Virtual Try-On (VTO) application is automated using a combination of Grounding DINO and SAM2. When a user uploads a clothing image, Grounding DINO first detects and localizes the clothing regions, after which SAM2 generates precise segmentation masks. The segmented items are then categorized by the Gemini AI backend, which assigns clothing type, color, and style attributes. The final segmented images are stored in Firebase Storage, while metadata such as category and attributes are saved in Firestore, allowing efficient organization and retrieval within the wardrobe system.

Workflow of Clothing Segmentation

The process begins when a user selects or captures a clothing photo within the mobile application. This image is uploaded to the backend server, where Grounding DINO performs object detection to identify and localize clothing regions. Once bounding boxes are detected, SAM2 processes these regions to generate precise segmentation masks, accurately isolates the clothing item from the background. The segmented image is then returned to the mobile application for user confirmation.

On the confirmation page, the user is presented with the segmented clothing and can review its quality. At this stage, users can visually inspect the segmented clothing and, if necessary, use the rotate button to adjust its orientation before saving. Once the confirmation is made, the finalized clothing image is sent to the Gemini AI backend for automatic categorization, where it is analyzed and assigned attributes such as category, color, and style. After categorization, the image is uploaded to Firebase Storage, and the corresponding metadata is stored in Firestore. Finally, the clothing item is displayed in the user's wardrobe within the application, ready to be managed or combined into outfits. To enhance usability, the wardrobe page includes a filtering feature that enables users to quickly locate clothing items by category. Finally, users can select items from their wardrobe to create and save new outfits in the system.

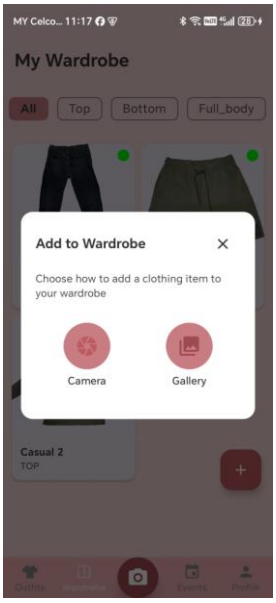


Figure 5.19 Add Clothes

Figure 5.20 Clothes to be segment

Figure 5.21 Confirm segmented clothes

Figure 5.22 Clothing item

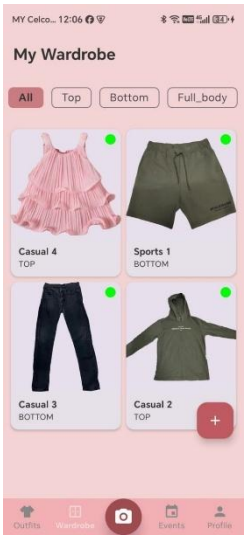


Figure 5.23 Segmented clothing item displayed in the wardrobe

5.4.3 Try-On Image Generation with KolorVTO

Overview of the Process

The Try-On Image Generation feature allows users to virtually try on clothing items using AI-generated images. It supports both single-item and combination outfits. Generated images can be saved, categorized, and stored in the app for future reference.

Workflow of Try-On Image Generation

The workflow begins with the user selecting a personal photo, either uploaded previously or captured live. In the single-item mode, the chosen clothing item is displayed at the bottom right of the user photo as a preview for confirmation. In the combination mode, the user selects both a top and a bottom, which are displayed as separate previews at the bottom right. Once confirmed, the app sends the user image and the clothing image via a POST API request to the backend server. On the server, the KolorsVTO model processes the inputs and generates an image of the user wearing the selected clothing item or outfit combination. The generated try-on image is returned to the app and displayed. When the user clicks save, a dialog appears to enter the outfit name. The outfit is then processed by the Gemini AI categorization system, stored in Firebase along with the user's data, and displayed in the Outfit Page for convenient retrieval.

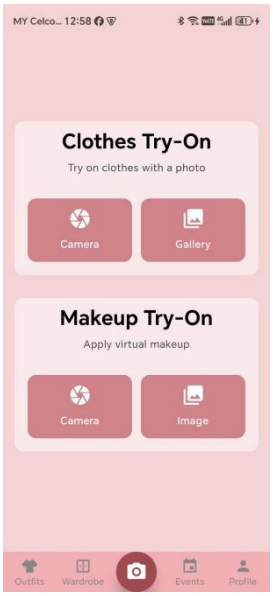


Figure 5.24 Try on page

Figure 5.25 clothing try on page

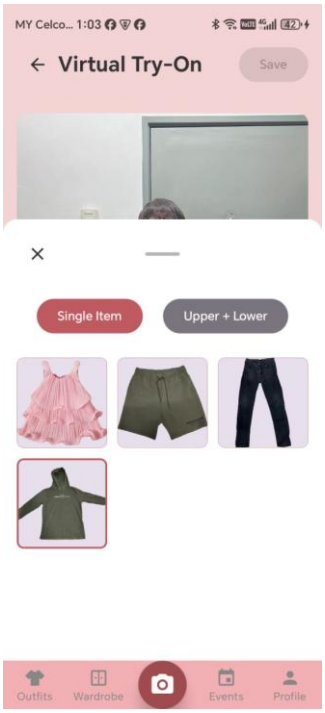


Figure 5.26 Choose a clothes

Figure 5.27 Try on screen

Figure 5.28 Try on result

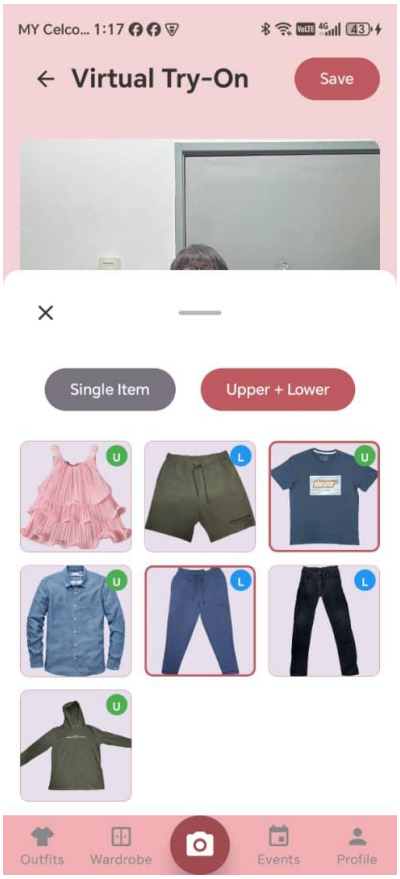


Figure 5.29 Combine mode

Figure 5.30 Try on result

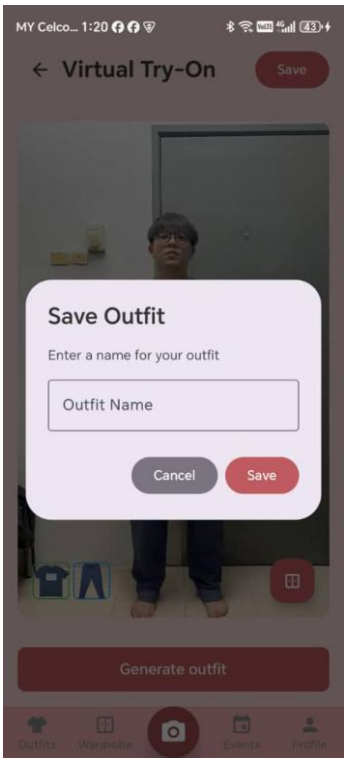


Figure 5.31 Saving result

Figure 5.32 Successfully save

5.4.4 Makeup Try-On

Overview of the Process

The Makeup Try-On feature allows users to virtually apply makeup effects using either a live camera feed or uploaded images. It supports real-time virtual try-on with the camera or applying makeup to existing photos. Users can preview the effects, make adjustments, and save the final images to Firebase for future access.

Workflow of Makeup Try-On

The workflow begins with the user choosing the input method: camera or image. If the image option is selected, a dialog appears allowing the user to choose a photo either from their saved outfits or from the device gallery. If the camera option is selected, the user can try on makeup effects live.

In the camera mode, the Makeup Try-On page displays the live camera feed along with a capture button and a makeup icon. Clicking the makeup icon opens a selection panel with predefined makeup effects that the user can try on in real time. After applying the desired effects, the user can capture the image, which is then saved to Firebase.

In the image mode, after selecting a photo, a makeup selection panel is displayed at the bottom of the screen, and a save button is available at the top right corner. Users

can apply built-in makeup effects, preview the result on the selected image, and save the final image to Firebase.

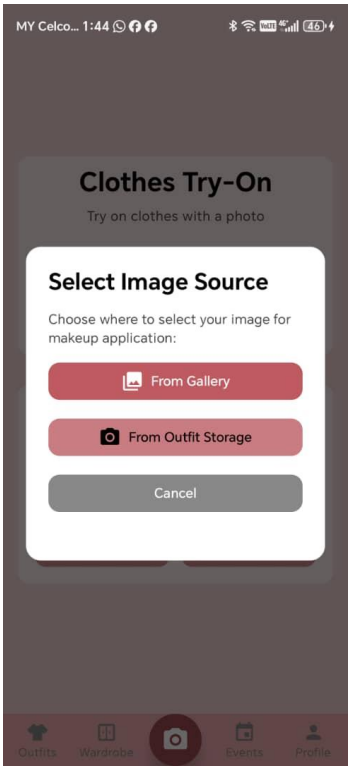


Figure 5.33 Selection

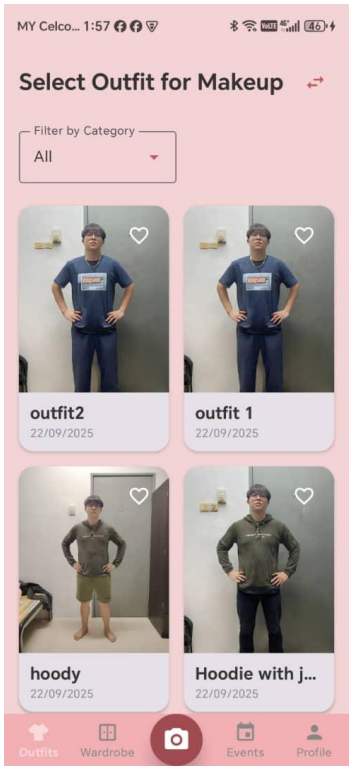


Figure 5.34 Select outfit

Figure 5.35 Before makeup

Figure 5.36 After makeup

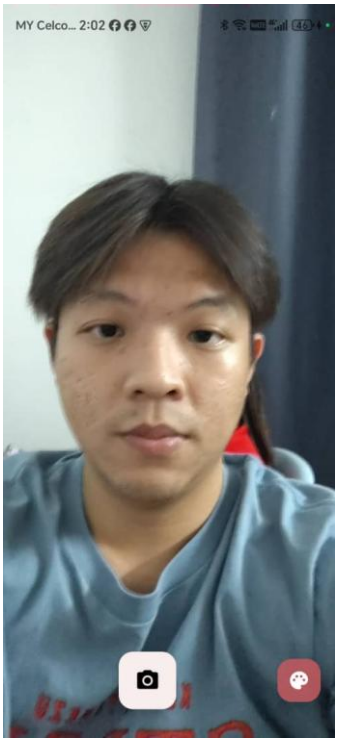


Figure 5.37 Before makeup

Figure 5.38 After makeup

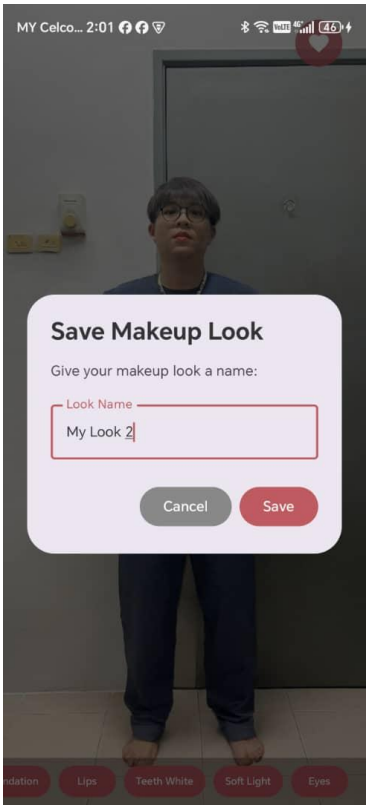


Figure 5.39 Saving Look

5.4.5 Outfit Management

Overview of the Process

The Outfit Management feature allows users to save, review, and manage their try-on results, including both clothing and makeup try-ons. Users can access a personalized history of their generated images and perform further actions such as viewing details or editing the saved items.

Workflow for Outfit Management

After a try-on image is generated whether through the KolorVTO API for clothing or the makeup try-on feature the image is uploaded to Firebase Storage and linked to the user's record in the Firestore "Outfits" and "Makeup" collection. In the Outfit Management page, an exchange button beside the page title allows users to toggle between clothing try-on results and makeup try-on results. Below the title, a filter panel lets users filter the displayed items based on categories such as clothing type, outfit combinations, or makeup type.

When a user selects an item from the displayed list, a new page opens showing the item details, including the try-on image, associated metadata (e.g., clothing IDs or makeup effect), and timestamp. On this page, users can also edit the saved entry, for example by updating the outfit name. This workflow ensures a comprehensive and interactive way to manage both clothing and makeup try-on results, providing an organized and user-friendly interface for wardrobe and makeup history.

Figure 5.40 Outfit page

Figure 5.41 Outfit detail

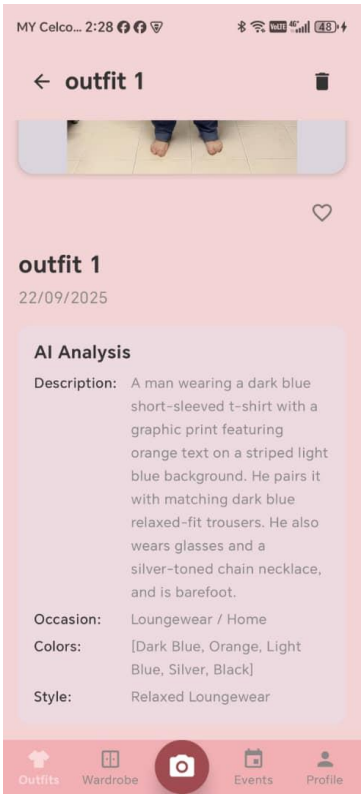


Figure 5.42 Outfit information

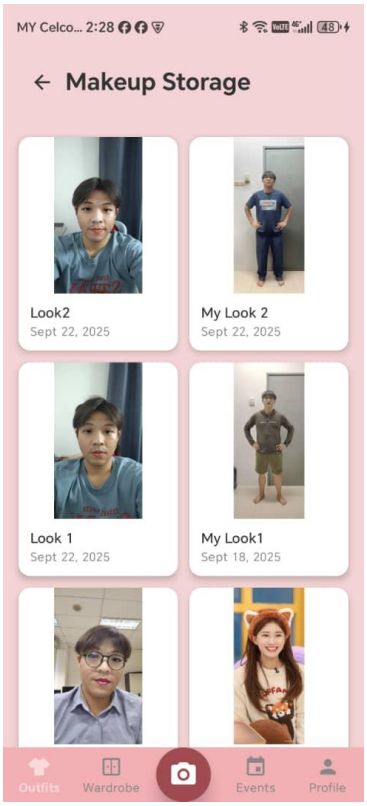


Figure 5.43 Makeup Storage Page

Figure 5.44 Look detail

5.4.6 Event Screen

Overview of the Process

The Event Screen provides users with a centralized interface to view, manage, and interact with their scheduled events. It combines a calendar view, event lists, filtering options, and notifications to ensure users stay informed about upcoming events. Users can select a date, add new events, view details, and receive timely reminders.

Workflow of the Event Screen

When the Event Screen is opened, the EventViewModel retrieves all events from the backend, including upcoming and past events, and updates the UI automatically via collectAsState().

The workflow begins with the user selecting a date on the calendar. Events for the selected date are filtered and displayed in a dedicated section. Users can also view all upcoming events or past events in separate sections.

Each event is represented by an EventCard, displaying the outfit image or type indicator, title, date and time, location (if available), and event type label. Clicking an EventCard opens the event's detail page for review or editing.

Users can add new events using the "+" button on the calendar. When an event is created, a notification is automatically scheduled using the device's notification system, ensuring the user receives timely reminders before the event occurs. A calendar legend is also provided to clarify event markers. All updates to the event lists such as adding, editing, or deleting events are dynamically reflected in the UI without manual refresh, maintaining a responsive experience.

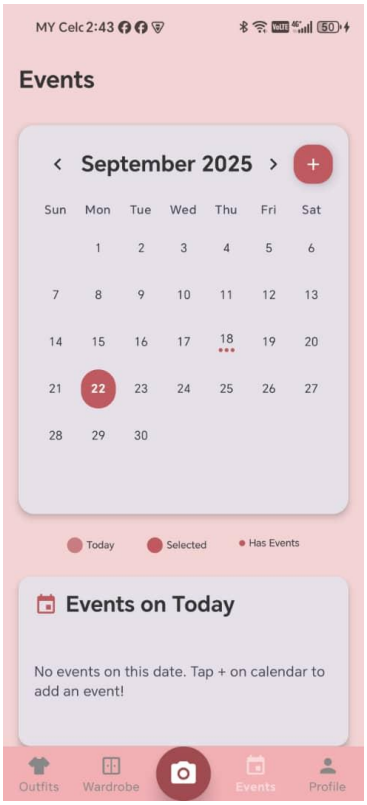


Figure 5.45 Event Page

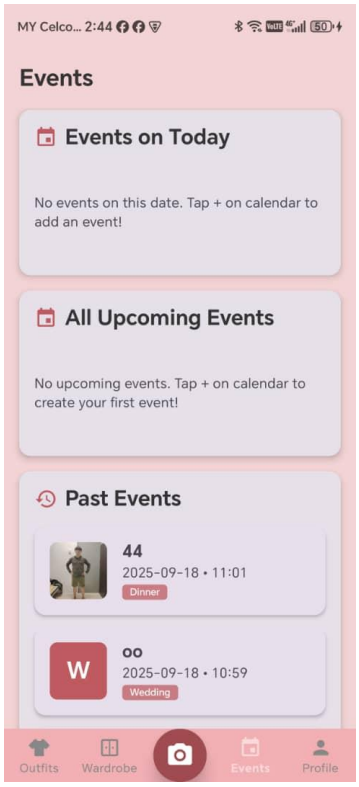


Figure 5.46 Event page

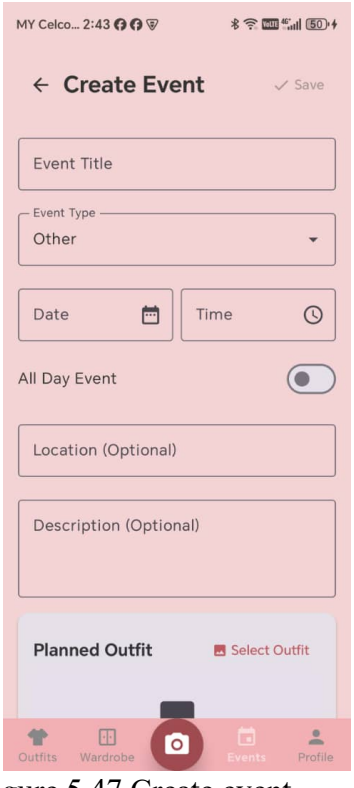


Figure 5.47 Create event

Figure 5.48 Event Detail

Figure 5.49 Notifications

5.4.7 Profile Screen

Overview of the Profile Screen

The Profile Screen allows users to view, manage, and update their personal information, including full name, date of birth, email, profile picture, and style preferences. It provides a centralized interface for managing personal data and preferences, ensuring a personalized experience within the app. The screen also offers access to additional app settings and sign-out functionality.

Workflow of the Profile Screen

When a user navigates to the Profile Screen, the application automatically loads the user's profile information from Firebase Firestore through the ProfileViewModel. Once the data is fetched, the screen displays the user's full name, email address, date of birth, and style preferences. Style preferences are presented in a chip layout, providing a clear and visually organized overview of the user's selected styles. If the user has not set any style preferences, the screen displays prompts encouraging them to update their preferences to enhance personalized recommendations.

The user can edit their profile by tapping the "Edit Profile" button, which opens a dialog that allows modifications to the full name, date of birth via a date picker, and style preferences using a multiple-choice selection panel. After confirming the changes, the updated data is sent to Firebase Firestore, ensuring persistence across devices, and the screen is refreshed to display the new information. A dedicated sign-out button

allows the user to securely log out, with the AuthViewModel handling authentication state and redirecting the user to the login screen.

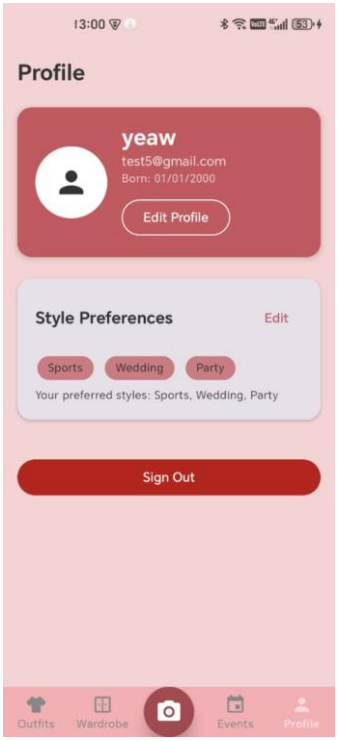


Figure 5.50 Profile page

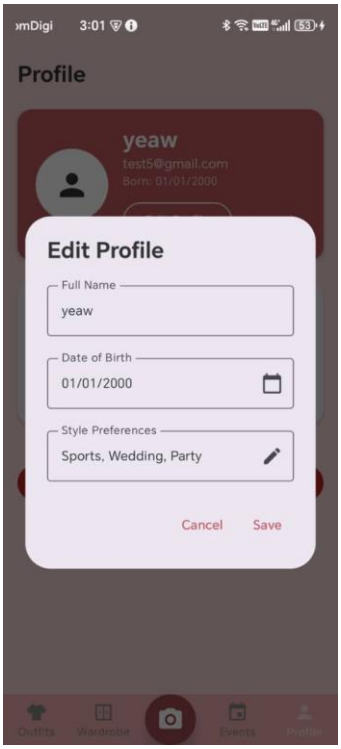


Figure 5.51 Edit Profile

5.5 Implementation Issues and Challenges

Within the creation and launch of the Virtual Try-On application, there were a number of aspects and challenges that were faced especially concerning the operations at the back end, AI processing, and performance of the devices. The use of cloud-based services to process and store the user data was among the primary challenges. Applications like clothing segmentation and try-on image generation are highly reliant on backend APIs like the KolorVTO model that generates realistic try-on images. Any latency or instability in such backend services has a direct impact on the performance and user experience of the application.

Connection problems were a major problem particularly when the users had poor or bad Wi-Fi or cellular data. Because the procedure of user images transmission, of retrieving segmented clothing items and of processing AI-generated try-on images require the uploading and the downloading of data stored in Firebase Storage and in

external APIs, the slow network speed greatly contributes to the processing time. Such latency may be annoying to users, and in other instances the app may seem non-responsive. Moreover, the unstable connection may cause the unsuccessful transmission of all data, and more than that, the requests will not be fulfilled, or the files will be corrupted, which also affects the quality of the try-on results.

The other problem that was identified when testing was the accuracy of AI-generated images with the KolorVTO model. Network delays can cause the results generated to be significantly different to the original clothing item or target appearance, in addition to creating the results generated when the image resolution is decreased to support slow uploading. Some of the cases have artifacts or unpredicted outputs of the model, and this reduces the apparent reality of the try-on feature. To achieve regular and high-quality AI results, high-speed internet connectivity should be stable, image information should be managed cautiously, and error-processing systems must be effective to resend or re-improve unsuccessful requests.

Besides backend and connection problems, some of the real-time functionalities like try-on makeup, which is powered by the Banuba SDK, generated device performance problems. The AR processing of Banuba is very demanding taking much computational power to execute smoothly. The AR rendering may lag on devices with less processing power or during operation of many processes with high load at the same time leading to slower updates and jittery displays. This heavy processing load has an impact on user experience where the live makeup preview might not look fluid and responsive in low-end hardware, thus limiting users to interact with the application in a more natural way.

Overall, the combination of network dependency, AI model sensitivity, high device processing requirements for AR features, and backend processing demands presented significant challenges. Addressing these issues involved optimizing image handling, improving error handling for failed or delayed requests, managing device performance, and providing user feedback during long operations to ensure that the app remains functional and user-friendly even under suboptimal conditions.

Chapter 6

System Evaluation And Discussion

6.1 System Testing and Performance Metrics

6.1.1 Black-Box Testing

System testing is a crucial stage in evaluating the effectiveness and reliability of the Virtual Try-On mobile application. This phase ensures that all implemented features operate according to the design specifications and satisfy the user requirements. The testing process focuses on assessing functional correctness, performance efficiency, and overall user experience. Performance metrics were established to quantify the system's effectiveness, including response time for try-on image generation, accuracy of clothing segmentation, and the success rate of wardrobe management operations. These metrics provide a measurable way to evaluate the responsiveness and stability of the system under different conditions.

Black-box testing was used as the primary method to evaluate the system. This testing approach examines the application's behavior from the user's perspective without considering the internal code structure. By focusing on inputs and expected outputs, black-box testing ensures that the system functions correctly under normal and edge-case scenarios. For the Virtual Try-On app, black-box testing covered the authentication module, clothing segmentation, try-on image generation, and wardrobe management. In the authentication module, tests were conducted for login, registration, and password recovery using both valid and invalid credentials. The system was expected to grant access for correct login information and display appropriate error messages for incorrect credentials.

The clothing segmentation functionality was also tested using various clothing images. Users could upload images of shirts, dresses, or jackets, and the system was expected to generate accurate masks that isolate the selected clothing item. Similarly, the try-on image generation feature was evaluated by uploading different user photos and selecting wardrobe items. The system's output was expected to produce realistic try-on images with correct positioning and proportions. In addition, wardrobe and outfit

management functions were tested to ensure that users could add, view, and save outfits, while invalid inputs, such as unsupported file types, were appropriately handled.

Performance testing was carried out to examine the system's behavior under multiple concurrent requests. The primary goal was to measure response time and server stability, especially since the KolorVTO model requires heavy computational processing. Metrics such as the average response time for image generation and success rates for segmentation and wardrobe operations were recorded. Overall, the black-box testing results demonstrated that the system meets functional requirements and provides reliable, accurate, and user-friendly performance. Minor delays in try-on image generation were noted due to the computationally intensive nature of the AI model, which is expected given the complexity of the processing involved.

6.2 Testing Setup and Result

6.2.1 Testing Setup

The testing environment was designed to simulate realistic user conditions and measure the performance of the system. The app was tested on Android devices running Android 12 and above, with a network connection of 4G and Wi-Fi. The backend server was deployed on Google Cloud Platform, handling clothing segmentation and try-on image generation requests.

Testing Setup:

- Devices: Android smartphone as the client device for running the mobile application.
- Backend: Python FastAPI server hosting the Grounded SAM2 model for clothing segmentation, the KolorVTO model for try-on image generation, and the AI categorization service for wardrobe organization

.

- Database: Firebase Firestore for storing user information, wardrobe metadata, outfit history, makeup looks, and event details.
- Storage: Firebase Storage for managing segmented clothing images, generated try-on images, and saved makeup results.
- Metrics: Evaluation was based on response time (time taken for model inference and result delivery), success rate (ratio of successful operations to

total attempts), and image accuracy (quality of segmented masks, try-on images, and makeup effects).

6.2.2 Testing Results

Module	Test Scenario	Expected Result	Observed Result	Pass/Fail
User Authentication	Register new account with valid email/password	Account created and redirected to onboarding	Account created and redirected correctly	Pass
User Authentication	Login with correct credentials	User logged in and redirected to main interface	Logged in successfully	Pass
User Authentication	Login with incorrect password	Error message displayed	Error message displayed	Pass
Wardrobe Management & Clothing Segmentation	Upload clothing image	Clothing correctly segmented and returned to app for confirmation	Segmented image returned successfully	Pass
Wardrobe Management & Clothing Segmentation	User confirms segmented image	Confirmed segmented clothing ready for upload	Confirmation works correctly	Pass
AI Categorization	Upload confirmed segmented clothing	Server generates category, color, style metadata, stores image and metadata in Firebase	Metadata generated correctly; stored successfully	Pass
Try-On Image Generation	Generate try-on image (single item)	AI server generates accurate try-on image, returned to app, stored in Firebase	Generated image accurate; stored successfully	Pass
Try-On Image Generation	Generate try-on image (combination outfit)	AI server generates try-on image for outfit combination, returned	Mostly accurate; minor inconsistencies in some combinations	Pass with minor issues

Makeup Try-On (Banuba)	Real-time makeup try-on via camera	Smooth application in live feed	Smooth on high-end devices; lag on mid-range devices	Pass with caution
Makeup Try-On (Banuba)	Apply makeup to uploaded image	Makeup applied correctly	Applied correctly	Pass
Outfit Management & History	Save try-on results	Images uploaded to AI Categorization server, processed, and stored in Firebase	Successfully saved and linked to user	Pass
Outfit Management & History	Edit saved outfit	Updates reflected in Firebase	Updates applied correctly	Pass
Event Screen	Add new event	Event created and notification scheduled	Event created, notification received	Pass
Event Screen	Edit event	Event updated and synced	Updates applied correctly	Pass
Profile Screen	Edit profile information	Profile updated and reflected in UI	Profile updated successfully	Pass
Profile Screen	Logout	User redirected to login screen	Redirected successfully	Pass

Table 6.1 Testing Results

6.3 Objective Evaluation

The project objective that were set at the onset of the development process were tackled and evaluated in a systematic manner during the implementation and testing processes. The initial goal, which is to create an Android-based Virtual Try-On (VTO) application is achieved. The Android Studio and Jetpack Compose were used in the complete creation of the mobile app, which is compatible with a great number of devices. The app is well compatible with the latest smartphones and remains accessible to cheap smartphones, indicating that the Android platform was an appropriate decision to reach a maximum number of users.

The second objective, which was to incorporate deep learning technologies, was also achieved by using the latest AI models. Grounded-SAM2 was utilized in accurate segmentation of clothing, KolorVTO was utilized in creating lifelike try-on pictures,

and Gemini AI was incorporated in intelligent division of clothing products. These technologies, combined, offer users an interactive and accurate virtual try on experience. It was also tested that clothing items were segmented with a high level of accuracy and that the try-on images generated were similar in appearance to those that were anticipated though there were instances of inaccurate results in case of poor connectivity, or when the server was overloaded.

The third objective, which facilitated complete body customization, was reached when the system was expanded to include mostly more than clothing try-on to make-up simulation. The camera feature allows the users to virtually apply different makeup styles in real time or make changes to the images posted by them by applying filters. This will enable the user to see their entire appearance, including clothes, and products in the same platform. Moreover, the features of outfit history, wardrobe management, and event reminders added to the features of the original objective increased the features of the application, making it a more ecstatic and functional user experience.

On balance, the three project objectives were all achieved, and the created system has shown high competence in Android application, AI implementation, and complete customization of the body. Solvable minor issues, including the delays in the processing and the high computational load of the makeup rendering, do not compromise the accomplishment of the goals but point out to the areas of potential optimization of the future wor

Chapter 7

Conclusion and Recommendation

7.1 Conclusion

The VTO mobile application prototype is an effective approach to digital styling that will solve the drawback of the real-life store experiences and pre-determined fashion collections. The system gives the user a hygienic, contactless, and highly personal styling experience, which employs both clothing and makeup try-on by utilizing both advanced AI technologies and an Android-based platform.

The app combines GroundingDINO and SAM2 to detect and segment clothes, KolorVTO to generate realistic try-ons, and Banuba to apply makeup effects, thus providing users with a full-body customization experience. Besides that, the platform has helped with AI-driven classification of the wardrobe, event-based outfit planning with notifications, and outfit history management where users can plan, save and revisit their styles with ease. All these properties are cleverly connected with Firebase services that provide authentication, storage, and data management, and the FastAPI backend provides an efficient deployment of AI models and modules interconnection.

The opportunity to add their own clothing pieces, identify and divide them with the help of GroundingDINO + SAM2, and create realistic try-ons with the help of KolorVTO are one of the most important innovations of the project that allows users to rely not only on the offered catalogs but also create personalized products and outfits on their own. This gives the users more freedom, personalization and creativity in regard to their own wardrobe and style in general. The experience organizer and history are other features that move the app beyond the try-on functions, making it more of a personal digital stylist.

Although these achievements have been made, there are still challenges. Banuba makeup SDK has high processing power that might cause performance problems with mid-range devices. Also, network stability is of key importance to the try-on performance; a problematic connection can slow down the processes and lower the quality of the obtained results. It will be important to resolve such problems by implementing optimization and adaptive processing techniques to improve them in the future.

On the whole, the Virtual Try-On mobile application is an indicator of the future potential of AI-powered styling platforms to transform the way in which people experience fashion and beauty. The system, with its scalable architecture, rich features, and user-centered design, gives a solid platform on which it can be extended into a full-immersion and intelligent virtual fashion platform.

7.2 Recommendation

For future improvements, several recommendations are proposed to enhance the Virtual Try-On application. First, performance optimization should be prioritized by reducing AI model processing time to ensure a smoother and faster user experience across a wide range of Android devices. This can be achieved through more efficient model deployment strategies and lightweight preprocessing techniques. Second, the scalability of backend services needs to be addressed by deploying APIs to robust cloud servers, enabling the system to support large-scale usage while maintaining stability and responsiveness even during peak demand.

In terms of functionality, the project would benefit from feature expansion, including the integration of additional makeup styles, hairstyles, and accessories such as jewelry, eyewear, and bags. The introduction of social media sharing for completed outfits would further enrich the user experience by allowing individuals to showcase their styles and receive feedback. Alongside this, implementing an AI-powered recommendation system would provide personalized outfit suggestions based on users' wardrobes, events, and preferences, making the app more intelligent and user-centric.

Finally, for long-term innovation, future research should explore 3D virtual try-on capabilities using AR/VR technologies. Such advancements would create a more immersive and realistic experience, bridging the gap between digital styling and real-world fashion. By adopting these recommendations, the application can evolve into a comprehensive, intelligent, and interactive fashion ecosystem**.**

REFERENCES

[1] H. Ghodhbani, M. Neji, I. Razzak, and A. M. Alimi, “You can try without visiting: a comprehensive survey on virtually try-on outfits,” Multimedia Tools and Applications, vol. 81, Mar. 2022, doi: https://doi.org/10.1007/s11042-022-12802-6.

[2] A. L. Roggeveen and R. Sethuraman, “How the COVID Pandemic May Change the World of Retailing,” Journal of Retailing, vol. 96, no. 2, pp. 169–171, Apr. 2020, doi: https://doi.org/10.1016/j.jretai.2020.04.002.

[3] A. Kavin, T Jithesh, G. Kalaiarasi, M. Selvi, R. Yogitha, and T.N. Suresh Babu, “Virtual Trial Room for Online Shopping,” Apr. 2024, doi: https://doi.org/10.1109/iccsp60870.2024.10544238.

[4] “IEEE COMSOC MMTC E-Letter MULTIMEDIA COMMUNICATIONS TECHNICAL COMMITTEE IEEE COMMUNICATIONS SOCIETY,” 2011. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c570ecd9eca1db203b808218e98bd65b15930279#page=37

[5] J. Gray, “A look inside Perfect Corp.'s YouCam Makeup app and the rise of beauty AR,” www.businessofbusiness.com, Mar. 08, 2021. https://www.businessofbusiness.com/articles/perfect-corp-youcam-makeup-app-beauty-tech-ar/

[6] S. Daiani, “The Role of Context Congruency in Smart Mirrors’ Virtual Try-On at Clothing Stores: Enhancing Customer Decision-Making and Satisfaction,” Ohiolink.edu, 2024. https://etd.ohiolink.edu/acprod/odb_etd/etd/r/1501/10?clear=10&p10_accession_num=ucin1721398757906967 (accessed Sep. 10, 2024).

[7] S. Degree, “We Tried 3 Closet Organization Apps & Here’s What We Think Of Them - Style Degree,” Style Degree, Mar. 25, 2021. https://styledegree.sg/closet-organization-apps-review/#:~:text=Absence%20Of%20In%2DApp%20Catalogue (accessed Sep. 10, 2024).

[8] “Virtual Fitting and Styling,” Style.me. https://style.me/virtual-fitting/

[9] X. Han, Z. Wu, Z. Wu, R. Yu, and L. Davis, “VITON: An Image-based Virtual Try-on Network.” Available: https://openaccess.thecvf.com/content_cvpr_2018/papers/Han_VITON_An_Image-Based_CVPR_2018_paper.pdf

[10] S. Jandial, A. Chopra, K. Ayush, M. Hemani, A. Kumar, and B. Krishnamurthy, “SieveNet: A Unified Framework for Robust Image-Based Virtual Try-On,” arXiv.org, 2020. https://arxiv.org/abs/2001.06265 (accessed May 02, 2025).

Appendix A

A.1 Code Sample

Segmentation.py

```
import cv2
import torch
import base64
import numpy as np
from fastapi import FastAPI, HTTPException, File, UploadFile, Form
from pydantic import BaseModel
from fastapi.middleware.cors import CORSMiddleware
import io
from sam2.build_sam import build_sam2
from sam2.sam2_image_predictor import SAM2ImagePredictor
from sam2.automatic_mask_generator import SAM2AutomaticMaskGenerator
from transformers import AutoProcessor, AutoModelForZeroShotObjectDetection
from PIL import Image

# Create FastAPI app
app = FastAPI(title="SAM2 Segmentation API")

# Add CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Setup CUDA and model configurations
```

```
torch.autocast(device_type="cuda", dtype=torch.bfloat16).__enter__()

if torch.cuda.get_device_properties(0).major >= 8:
    torch.backends.cuda.matmul.allow_tf32 = True
    torch.backends.cudnn.allow_tf32 = True

DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# Load GroundingDINO
grounding_processor = AutoProcessor.from_pretrained("IDEA-Research/grounding-dino-tiny")
grounding_model = AutoModelForZeroShotObjectDetection.from_pretrained(
    "IDEA-Research/grounding-dino-tiny"
).to(DEVICE)

CHECKPOINT = "checkpoints/sam2.1_hiera_tiny.pt"
CONFIG = "configs/sam2.1/sam2.1_hiera_t.yaml"

# Load the model
sam2_model = build_sam2(CONFIG, CHECKPOINT, device=DEVICE, apply_postprocessing=False)
mask_generator = SAM2AutomaticMaskGenerator(sam2_model)

@app.get("/")
async def root():
    return {"message": "SAM2 Segmentation API is running. Use POST /segment endpoint."}

@app.post("/segment")
async def segment_image(file: UploadFile = File(...)):
    try:
```

```
    contents = await file.read()
    image_pil = Image.open(io.BytesIO(contents)).convert("RGB")
    image_np = np.array(image_pil)

    # Step 1: detect all clothing items
    inputs = grounding_processor(
        images=image_pil,
        text="clothes . shirt . t-shirt . pants . dress . jacket . coat . skirt . blouse . sweater . hoodie . jeans . trousers . shorts . gown . jumpsuit .",
        return_tensors="pt"
    )
    inputs = {k: v.to(DEVICE) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = grounding_model(**inputs)
    results = grounding_processor.post_process_grounded_object_detection(
        outputs,
        input_ids=inputs["input_ids"],
        box_threshold=0.35, text_threshold=0.25,
        target_sizes=[image_np.shape[:2]]
    )
    boxes = results[0]["boxes"].cpu().numpy()
    if len(boxes) == 0:
        return {"error": "No clothes detected"}

    # Step 2: segment each with SAM2
    predictor = SAM2ImagePredictor(sam2_model)
    predictor.set_image(image_np)
    all_masks = []
    for box in boxes:
        masks, _, _ = predictor.predict(box=np.array([box]), multimask_output=False)
```

```
            if len(masks) > 0:
                mask_bool = masks[0].astype(bool)
                all_masks.append(mask_bool)

        if len(all_masks) == 0:
            return {"error": "No masks generated"}

        # Step 3: merge masks
        combined_mask = np.any(all_masks, axis=0)  # shape: H x W, boolean mask

        # Step 4: crop or prepare output
        # optional: find bounding box around combined_mask
        ys, xs = np.where(combined_mask)
        y1, y2 = ys.min(), ys.max()
        x1, x2 = xs.min(), xs.max()
        # crop image_np
        cropped_img = image_np[y1:y2, x1:x2]
        # apply mask to crop
        cropped_mask = combined_mask[y1:y2, x1:x2]
        # build RGBA
        alpha = (cropped_mask.astype(np.uint8) * 255)
        cropped_rgba = np.dstack((cropped_img, alpha))

        # encode as base64 if you want
        _, buffer = cv2.imencode(".png", cv2.cvtColor(cropped_rgba,
cv2.COLOR_RGBA2BGRA))
        b64_img = base64.b64encode(buffer).decode("utf-8")
        return {"mask": b64_img}

    except Exception as e:
```

```
        return {"error": str(e)}



if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8080)
```

Image_to_text.py

```
from fastapi import FastAPI, File, UploadFile, Form
from google import genai
from google.genai import types
from google.cloud import firestore, storage
from pydantic import BaseModel
import enum
import uuid
import datetime
from fastapi.encoders import jsonable_encoder
import json
import os
from google.oauth2 import service_account


# ----------------------------
# Define structured schema
# ----------------------------
class Category(enum.Enum):
    ALL = "all"
    UPPER = "top"
    BOTTOM = "bottom"
    FULL_BODY = "full_body"
```

```
class Occasion(enum.Enum):
    CASUAL = "Casual"
    FORMAL = "Formal"
    BUSINESS_OFFICE = "Business / Office"
    PARTY_CELEBRATION = "Party / Celebration"
    WEDDING = "Wedding"
    SPORTS_ACTIVE = "Sports / Active"
    TRAVEL_VACATION = "Travel / Vacation"
    LOUNGEWEAR_HOME = "Loungewear / Home"
    TRADITIONAL_CULTURAL = "Traditional / Cultural"
    SEASONAL_WEATHER = "Seasonal / Weather-based"


class ClothingItem(BaseModel):
    description: str
    category: Category
    color: str
    pattern: str
    style: str
    occasion: str


# ----------------------------
# Smart Name Generation Function
# ----------------------------
def generate_smart_item_name(user_id: str, item_occasion: str, item_category: str,
item_style: str) -> str:
    """
    Generate smart names based on occasion like 'Casual 1', 'Formal 2', 'Business 3',
etc.
    based on existing items in the user's wardrobe
    """
    try:
```

```
# Map occasion text to proper display name
occasion_mapping = {
    "casual": "Casual",
    "formal": "Formal",
    "business": "Business",
    "office": "Business",
    "business / office": "Business",
    "party": "Party",
    "celebration": "Party",
    "party / celebration": "Party",
    "wedding": "Wedding",
    "sports": "Sports",
    "active": "Sports",
    "sports / active": "Sports",
    "travel": "Travel",
    "vacation": "Travel",
    "travel / vacation": "Travel",
    "loungewear": "Loungewear",
    "home": "Loungewear",
    "loungewear / home": "Loungewear",
    "traditional": "Traditional",
    "cultural": "Traditional",
    "traditional / cultural": "Traditional",
    "seasonal": "Seasonal",
    "weather": "Seasonal",
    "seasonal / weather-based": "Seasonal"
}

# Determine base name from occasion first, then style, then category
occasion_lower = item_occasion.lower().strip()
```

```
    style_lower = item_style.lower().strip()

    base_name = None

    # Try to match occasion first
    for key, value in occasion_mapping.items():
        if key in occasion_lower:
            base_name = value
            break

    # If no occasion match, try style
    if not base_name and (style_lower and
        style_lower != "unknown" and
        style_lower != "test clothing item" and
        style_lower != "not specified" and
        style_lower != ""):

        # Map common styles to occasions
        if "casual" in style_lower:
            base_name = "Casual"
        elif "formal" in style_lower:
            base_name = "Formal"
        elif "business" in style_lower or "office" in style_lower:
            base_name = "Business"
        elif "party" in style_lower or "evening" in style_lower:
            base_name = "Party"
        elif "sport" in style_lower or "active" in style_lower:
            base_name = "Sports"
        else:
            base_name = item_style.capitalize()
```

```
        # If no occasion or style match, use category as fallback
        if not base_name:
            if item_category.lower() == "top":
                base_name = "Top"
            elif item_category.lower() == "bottom":
                base_name = "Bottom"
            elif item_category.lower() == "full_body":
                base_name = "Outfit"
            else:
                base_name = "Item"

        # Query existing items with the same base name pattern
        user_items_ref =
db.collection("users").document(user_id).collection("wardrobeItems")
        existing_items = user_items_ref.get()

        # Count items with similar style/base name
        count = 0
        for item_doc in existing_items:
            item_data = item_doc.to_dict()
            existing_display_name = item_data.get("display_name", "")

            # Check if this item has the same base name
            if existing_display_name.startswith(base_name + " "):
                try:
                    # Extract number from existing name like "Casual 3"
                    number_part = existing_display_name.split(" ")[-1]
                    if number_part.isdigit():
                        item_number = int(number_part)
```

```
                    count = max(count, item_number)
                except:
                    continue

        # Generate new name with next number
        new_number = count + 1
        smart_name = f"{base_name} {new_number}"

        print(f"Generated smart name: '{smart_name}' for occasion='{item_occasion}', style='{item_style}', category='{item_category}'")
        return smart_name

    except Exception as e:
        print(f"Error generating smart name: {e}")
        # Fallback to simple naming
        return f"Item {datetime.datetime.now().strftime('%m%d%H%M')}"

# ----------------------------
# Initialize FastAPI & Gemini
# ----------------------------
app = FastAPI(title="Clothing Metadata API")
client = genai.Client()  # Gemini API client

# Initialize Firebase with credentials
try:
    # Path to your service account key file
    credentials_path = "vto-app-f7833-firebase-adminsdk-fbsvc-ea90c9d06d.json"  # Place this file in the same directory

    if os.path.exists(credentials_path):
        # Load credentials from file
```

```
        credentials =
service_account.Credentials.from_service_account_file(credentials_path)
        db = firestore.Client(credentials=credentials)
        storage_client = storage.Client(credentials=credentials)
        bucket = storage_client.bucket("vto-app-f7833.firebasestorage.app")
        print("✅ Firebase credentials loaded successfully")
    else:
        print(f"❌ Credentials file not found: {credentials_path}")
        print("Please download firebase-credentials.json from Firebase Console")
        exit(1)
except Exception as e:
    print(f"❌ Firebase initialization failed: {e}")
    exit(1)


@app.post("/categorize/{user_id}")
async def categorize_clothing(user_id: str, file: UploadFile = File(...)):
    try:
        # Read uploaded image
        image_bytes = await file.read()


        # Prompt Gemini to generate description + structured JSON
        prompt = (
            "Interpret this clothing item and provide:\n"
            "1. A simple description.\n"
            "2. Structured JSON with fields:\n"
            "   - description (text)\n"
            "   - category (enum: all, top, bottom, full_body)\n"
            "   - color (text)\n"
            "   - pattern (text)\n"
            "   - style (text)\n"
```

```
        "   - occasion (choose from: Casual, Formal, Business / Office, Party / Celebration, Wedding, Sports / Active, Travel / Vacation, Loungewear / Home, Traditional / Cultural, Seasonal / Weather-based)\n"
        "\nFor occasion field, pick the most appropriate from the list above. If multiple apply, choose the primary one."
    )

    # Call Gemini with structured output
    response = client.models.generate_content(
        model="gemini-2.5-flash",
        contents=[
            types.Part.from_bytes(
                data=image_bytes,
                mime_type="image/png"
            ),
            prompt
        ],
        config={
            "response_mime_type": "application/json",
            "response_schema": ClothingItem,
            "thinking_config": types.ThinkingConfig(thinking_budget=0)  # disables extra thinking
        }
    )

    # Parse Gemini output safely
    try:
        structured_json = json.loads(response.text)
    except Exception:
        structured_json = {"error": "Failed to parse Gemini output", "raw_text": response.text}
```

```
        # Generate smart name based on AI analysis results
        item_occasion = structured_json.get("occasion", "")
        item_style = structured_json.get("style", "")
        item_category = structured_json.get("category", "other")
        smart_name = generate_smart_item_name(user_id, item_occasion,
item_category, item_style)

        # Upload image to Firebase Storage - FIXED PATH
        item_id = str(uuid.uuid4())

        # Use PNG format to preserve transparency from segmentation
        storage_path = f"users/{user_id}/wardrobe/{item_id}.png"
        blob = bucket.blob(storage_path)

        # Upload with error handling
        try:
            blob.upload_from_string(image_bytes, content_type="image/png")
            print(f"✅ Successfully uploaded to Firebase Storage")
        except Exception as upload_error:
            print(f"❌ Upload failed: {upload_error}")
            raise upload_error

        # Make blob public and generate proper URL
        try:
            blob.make_public()
            print(f"✅ Successfully made blob public")
        except Exception as public_error:
            print(f"❌ Make public failed: {public_error}")
            print(f"⚠️  Continuing with URL generation...")
```

```
        bucket_name = bucket.name
        blob_name = blob.name.replace("/", "%2F")  # URL encode the path
        image_url =
f"https://firebasestorage.googleapis.com/v0/b/{bucket_name}/o/{blob_name}?alt=me
dia"


        print(f"=== IMAGE UPLOAD DEBUG ===")
        print(f"User ID: {user_id}")
        print(f"Item ID: {item_id}")
        print(f"Storage path: {storage_path}")
        print(f"Bucket name: {bucket_name}")
        print(f"Blob name (encoded): {blob_name}")
        print(f"Generated URL: {image_url}")
        print(f"File name: {file.filename}")
        print(f"Image bytes size: {len(image_bytes)}")
        print("==========================")


        # Save metadata + image URL to Firestore with error handling
        try:
            doc_ref =
db.collection("users").document(user_id).collection("wardrobeItems").document(item
_id)
            firestore_data = {
                "image_name": file.filename,
                "image_url": image_url,
                "metadata": structured_json,
                "timestamp": datetime.datetime.utcnow().isoformat(),
                "display_name": smart_name  # Generated smart name
            }
            doc_ref.set(jsonable_encoder(firestore_data))
            print(f" ✅ Successfully saved to Firestore")
```

```
            print(f"Document path: users/{user_id}/wardrobeItems/{item_id}")
        except Exception as firestore_error:
            print(f"❌ Firestore save failed: {firestore_error}")
            raise firestore_error

        return {
            "message": "Clothing categorized and saved to Firebase",
            "item_id": item_id,
            "image_url": image_url,
            "metadata": structured_json,
            "timestamp": datetime.datetime.utcnow().isoformat()
        }

    except Exception as e:
        return {"error": str(e)}

@app.post("/upload_outfit/{user_id}")
async def upload_outfit(
    user_id: str,
    file: UploadFile = File(...),
    outfit_name: str = Form(...)  # User provides outfit name
):
    try:
        # Read uploaded image
        image_bytes = await file.read()

        # Prompt Gemini to generate outfit occasion + metadata
        prompt = (
            "Analyze this outfit image and provide structured JSON with:\n"
            "1. description (short text)\n"
```

```
        "2. occasion (choose one: Casual, Formal, Business / Office, Party / Celebration, "
        "Wedding, Sports / Active, Travel / Vacation, Loungewear / Home, Traditional / Cultural, Seasonal)\n"
        "3. color (main colors)\n"
        "4. style (short style description)\n"
    )

    # Call Gemini for structured metadata
    response = client.models.generate_content(
        model="gemini-2.5-flash",
        contents=[
            types.Part.from_bytes(data=image_bytes, mime_type="image/png"),
            prompt
        ],
        config={
            "response_mime_type": "application/json"
        }
    )

    # Parse Gemini response safely
    try:
        structured_json = json.loads(response.text)
    except Exception:
        structured_json = {"error": "Failed to parse Gemini output", "raw_text": response.text}

    # Generate unique ID for outfit
    item_id = str(uuid.uuid4())

    # Firebase Storage path (outfits)
```

```
        storage_path = f"users/{user_id}/outfits/{item_id}.png"
        blob = bucket.blob(storage_path)

        try:
            blob.upload_from_string(image_bytes, content_type="image/png")
            blob.make_public()
            print(f"✅ Uploaded outfit image to Storage")
        except Exception as e:
            print(f"❌ Upload failed: {e}")
            raise e

        # Generate public URL
        bucket_name = bucket.name
        blob_name = blob.name.replace("/", "%2F")
        image_url =
f"https://firebasestorage.googleapis.com/v0/b/{bucket_name}/o/{blob_name}?alt=me
dia"

        # Firestore path (outfits)
        try:
            doc_ref =
db.collection("users").document(user_id).collection("outfits").document(item_id)
            firestore_data = {
                "outfit_name": outfit_name,  # user-provided
                "image_url": image_url,
                "metadata": structured_json,
                "timestamp": datetime.datetime.utcnow().isoformat()
            }
            doc_ref.set(jsonable_encoder(firestore_data))
            print(f"✅ Saved outfit metadata to Firestore")
        except Exception as firestore_error:
```

```
            print(f"❌ Firestore save failed: {firestore_error}")
            raise firestore_error

        return {
            "message": "Outfit uploaded and saved successfully",
            "item_id": item_id,
            "outfit_name": outfit_name,
            "image_url": image_url,
            "metadata": structured_json
        }

    except Exception as e:
        return {"error": str(e)}
# ----------------------------
# Run locally
# ----------------------------
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

A.2 Poster


UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
Virtual Try On
Developer
Yeaw Wooi Tong
Supervisor
Ms Tong Dong Ling
Module
User Authentication
Clothing Segmentation
Wardrobe
Try On
Outfit
Event Planner
Profile Management
Make Up
AI Categorized
Introduction
Trying on clothes in stores is often time-consuming, and inefficient. Virtual Try On offers a more convenient and modern shopping experience, with makeup try-on included to enhance personalization.
Project Objective
To develop a mobile application that allows users to virtually try on clothes using AI-powered clothing segmentation and image generation, enhancing the personal styling experience..
Proposed method
Integrates virtual wardrobe and makeup try-on in one system. Clothing is segmented with the Grounded SAM2 model and stored in a virtual wardrobe, while facial images are processed with a face parsing model to detect key regions. Users can then select clothes and makeup styles, which are applied using GAN-based transfer or simple overlays.
Result
Virtual Try-On
Virtual Try-On
Clothing Try On
Make Up Try On