INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER

Lufyco: A Mobile-First, AI-Powered Fashion App for Image-Based Discovery, Context-Aware Outfit Recommendations, and Digital Wardrobe Management

A Project Proposal by

Nihinsa Bandara

Supervised by

Ms. Akarshani Amarasinghe

September 2025

Abstract

E-commerce shopping for apparel is very popular, but most people still have problems like not finding the right outfits , not getting personalized suggestions, and being in doubt about product fitting. Most shopping apps do not have advanced features like image search or AI-based styling, and therefore customers are less satisfied and also tend to leave products in the shopping cart without making a purchase. Shopkeepers also have problems showcasing their products and achieving sales. Lufyco solves these by utilizing AI, image search, and wardrobe management to give users a smarter shopping experience.

Lufyco is built as a cross-platform app using React Native (mobile), Node.js and Python (Backend). It uses MongoDB for database storage. The app enables the user to upload a photo to find similar clothes using deep learning. There is an AI Stylist that suggests outfits based on mood, weather, occasion, and one's own wardrobe. It also enables search filtering, suggestions, order tracking, and payment securely. It uses cloud services (AWS S3, Firebase) and secure payment gateways for payments and storage. The system was tested for image search accuracy and outfit suggestions, with satisfactory performance in augmenting the online shopping experience.

Subject Descriptors

- Information systems → Information retrieval → Recommender systems → Personalized recommendation
- Computing methodologies → Machine learning → Computer vision → Image classification
- Human-centered computing → Human computer interaction (HCI) → Mobile computing

Keywords: Lufyco, Clothing Marketplace, Image-Based Search, AI Stylist, Personalized Recommendation

Declaration

I confirm that the subsequent project report, as well as all accompanying artefacts and supporting documentation, is my own individual work. It is not a prior submission nor is it being submitted, for the purpose of obtaining any degree or diploma from any other institution. Any material taken or adapted from the work of others has been duly acknowledged and properly cited in accordance with academic standards.

Student Name: Dunithi Nihinsa Saruwa Bandara

Student ID: 20220921

UOW ID: w1999466

Acknowledgement

This research project has been a testing but rewarding experience that called for commitment, effort, and perseverance. Its successful completion could not have been achieved without the direction, inspiration, and support of numerous individuals for whom I am most thankful.

Finally, I would also like to offer my sincere thanks to my supervisor, Ms.Akarshani Amarasinghe, for his constant guidance, useful criticism, and invaluable advice throughout the course of this research. His expertise, insightful suggestions, and constant motivation have been instrumental in directing this work and in helping me to overcome the many challenges faced throughout the process.

I also appreciate my university lecturers and guides who, during all these years, have shared their experience and knowledge, providing me with the academic background and abilities necessary for successfully completing this project.

Finally, I would like to express my heartfelt appreciation to my friends and family for their ongoing support, patience, and encouragement. Their faith in me gave me the strength and willpower to complete this work, and I thank them for being part of every step throughout this journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
CV	Computer Vision
NLP	Natural Language Processing
CNN	Convolutional Neural Network
ViT	Vision Transformer
ANN	Approximate Nearest Neighbor (vector search)
FAISS	Facebook AI Similarity Search
ScaNN	Scalable Nearest Neighbors
HNSW	Hierarchical Navigable Small World
mAP	mean Average Precision
R@K	Recall at K
P@K	Precision at K
MRR	Mean Reciprocal Rank
NDCG	Normalized Discounted Cumulative Gain
AUC-ROC	Area Under the ROC Curve
F1	Harmonic mean of Precision and Recall
SUS	System Usability Scale
A/B	Split testing of two variants
UI	User Interface
UX	User Experience
RN	React Native
JS / TS	JavaScript / TypeScript
API	Application Programming Interface
REST	Representational State Transfer
JSON	JavaScript Object Notation

CDN	Content Delivery Network
CDN-Cache	Cached content served via CDN for speed
JWT	JSON Web Token
OAuth	Open standard for secure authorization
PII	Personally Identifiable Information
GDPR	General Data Protection Regulation
HTTPS / TLS	Encrypted web transport protocol / Transport Layer Security
S3	Amazon Simple Storage Service
CRUD	Create, Read, Update, Delete
EDA	Exploratory Data Analysis
QoS	Quality of Service
p50 / p95	50th / 95th percentile latency
PQ	Product Quantization
DDP	Distributed Data Parallel
ONNX	Open Neural Network Exchange
CI/CD	Continuous Integration / Continuous Delivery
KPI	Key Performance Indicator
TTFB	Time To First Byte
TTI	Time To Interactive

1 CHAPTER 01: INTRODUCTION

1.1 Chapter Overview

Fashion e-commerce has gone mobile-first while users still experience choice overload, poor personalization, and difficulty matching items to mood, occasion, weather, or wardrobe. Traditional "search → filtering → scrolling" flows can create a significant cognitive load for users when the tagging is inconsistent or, at a minimum, the user is not aware of how to name their styles. Lufyco responds to the problems of e-commerce through image-based discovery, and an AI Stylist using mood/occasion/weather as well as a digital wardrobe for explainable and editable outfit recommendations. This chapter will frame the problem, motivations, summarize related work, and draw out research gaps, contributions, and challenges, setting out research questions and aims, objectives, scope, core tools, and a brief summary leading into the literature review.

1.2 Problem Background

1.2.1 Challenges in Online Clothing Discovery and Purchase

Purchasing clothing through mobile apps is generally common practice, yet a complicated experience. Fashion is fundamentally tied to personal identity and is impacted by specific context such as going to the office, attending a ceremony or party, attending a night out, or traveling as well as more functional factors such as climate, seasonality, dress codes, and how well something fits into the existing wardrobe.

Current mobile shopping experiences utilize a keyword-based search with static filters. When consumers do not know the exact word to use they receive either no action or spend too long searching through categories. Mobile app shopping for clothing frequently involves large catalogs with variable support for attributes (color/pattern/fabric/silhouette) and photography (lighting, pose, and background). Such variation leads to product selection overload and results in decision fatigue, or simply abandoning the cart altogether.

A closely related obstacle created by the broad decision scope in clothing merchandise is an uncertainty in styling: just because you are drawn to a single product, does not mean you are able to decide what to style with the new product for a specific context (for example, semi-formal evening out in humid weather, or rainy weekday commute). Size and fit guides, as well as images of models in the clothing item being considered do not provide reasoning for style cohesion, which includes color coordination, touch coordination, seasonally appropriate styling, and some fit with a personal aesthetic, particularly when coordinating with partially-

owned items. Without guidance, the user is left going back and forth between scrolling for goodness sake and doing an impulse buy, both of which result in diminishing long-term satisfaction.

1.2.2 Limitations of Existing Fashion Apps and Tools

While some progress has been made in the mobile experience of the overall user experience (UX) experience, the majority of apps remain anchored to presenting a catalog experience first, then checking out the items, but not much more in terms of intelligence that is context-aware(Wang & Chen, 2023; Abugabah et al., 2020). Recommendation styles in most interfaces (co-view, co-purchase, etc.), or generic "similar items," are just ways we might repeat popularity and reinforce history as opposed to inferring intent-in-the-moment (the user's mood, a special event, local weather, etc.). Taxonomies are uneven and incomplete (even for attributes), and there are inconsistent quality/differences with photography styles(Liu et al., 2016; Ge et al., 2019). We may end up with weak keyword searches and blunt filters.

While there are apps that offer some editorial "style edits" or "complete the look" style, these are still static and not particular to the user experience, and the user experience. When image-based search is an option, it is often an after-thought, limited not only to domain shift (the user's snapshots vs. studio images), but also has the challenge of long-tail categories and weak cross-category pairings (i.e. pairing footwear and accessories to apparel)(Kiapour et al., 2015; Ge et al., 2019). Users can rarely get rationalization ("why this top with that skirt?"), limiting trust and ongoing editability.

1.2.3 Potential of AI for a Context-Aware, Wardrobe-Centric Mobile App

Contemporary computer vision (Convolutional Neural Networks, Vision Transformers) and hybrid recommenders (content + behavior + context) create a strong consumer experience. The vision models learn visual embeddings taking into account silhouette, pattern, color, and texture. The metric learning + vector search makes "find similar" quick from a photo or screen shot uploaded by the user. One can also include context - mood, occasion, weather, and time horizon along with a personal digital wardrobe (the user uploads their own items which are then auto-tagged by category/color/season/fabric). The result is a situational AI Stylist that can not only suggest a complete and editable outfit, but also calls out alternatives and provides compact explanations ("balances textures; picks up on your tan trousers; suggests water-resistant shoes based on weather") to increase consumer trust, speed to decision, and post-purchase satisfaction.

1.2.4 Challenges in Engineering a Consumer-Grade AI Fashion App

In order for the system to be actually significant, it will need to handle non-studio photos, a range of lighting conditions, occlusion, changing poses, regional styles, and long-tail categories while still having stable, timely embeddings that reflect the changing catalog. Context fusion will require some consideration for feature design, real-time inferences with latency costs for the user, and cold start (users/items) behavior. UX will also have to support reasoning through complex outputs - lightweight mood/occasion capture, explanations that aren't lengthy but are reasonable and reliable, and edits that are each one tap (swap, recolor, re-style). Last, and not least, privacy, security, and fairness considerations matter - wardrobes and preferences are sensitive data. So flows will need to start encrypted and rankings can't indicate unfairness to either brand, or across styles, or narrow echo chambers.

1.3 Project Goals and Significance

This project proposes Lufyco, a mobile-only experience that brings together:

- Image-Based Product Discovery
  Users are able to upload a photograph or screenshot that captures their inspiration, and the application will provide products from several catalogs that are visually and semantically similar to the image, including extracting attributes (category, color, pattern, season).

- Context-Aware AI Stylist (“My Smart Wardrobe” + Emotion-Aware Outfit Advisor)
  The user records mood, occasion, and time, while the app automatically detects weather. The application utilizes the user's digital wardrobe to create complete outfits and will only suggest shop items when necessary. The user will also have the flexibly to customize and save a style card.

- Explainable, Editable Recommendations
  The style suggestion will always have a brief rationale for the recommendation (e.g., color pairing, texture/weight balance, weather appropriate), along with interaction opportunities (swap item, change color palette, reorder recommendations based on comfort, etc.).

Impact: Lufyco reduces effort, increases personalization and is reduces the uncertainty in styling. The experience will assist users with not only finding better products, but also quicker decisions, as well as greater satisfaction and loyalty, especially for families who do not know how to put together clothing in a way that is an appropriate outfit for the context or event based on inspiration.

1.4 Problem Definition

There is a distinct shortage of integrated fashion mobile apps that are consumer centric, have strong image-based discovery, context-aware outfit planning, wardrobe-aware personalization, and explainability in a single production-ready experience. The fashion mobile apps that exist mostly focus on catalogs and transactions, while any filters are syntactic, not semantic or contextual. Although popularity-based recommendation heuristics exist, they miss the commonly relied upon variables of mood, event, climate, and wardrobe these variables are at the heart of a fashion decision. When image search is even offered by a fashion mobile app, it was fragile at best, and rarely paired with a context-aware re-ranking of items or wardrobe graphs. The absence of an integrated, explainable AI stylist introduces friction, indecision, and low confidence and undermines the allure of mobile fashion shopping.

1.5 Problem Statement

The majority of fashion mobile apps do not offer a cohesive, AI-based experience that (i) understands user context (mood, occasion, weather) and wardrobe in order to provide explainable and editable outfit recommendations; and
(ii) provides strong image-based discovery across heterogeneous catalogues. Resultantly, users suffer from wearable fatigue and a reduced sense of confidence while shopping.

1.6 Research Motivation

Lufyco is motivated by the need to humanize mobile fashion shopping.

- For users, clothing is an identity, norms, and constraints situated decision rich with context. Recommendations based on a history alone is out of favor, because it forces consideration of intent now and requires tedious browsing. Considering

mood/occasion/weather and wardrobe could increase user satisfaction, reduce time-to-find, and build confidence in post-purchase plans.

- For research, Lufyco provides a natural testbed to improve the experience of translating deep vision + hybrid recommendation + context fusion for a trustworthy, explainable, and scalable consumer app. We will evaluate user experience metrics (time to find, system usability scale, task success) as well as behavioral outcomes (engagement and revisit rate) to bring more reproducible research methods for the application of AI in retail.
- For ethics/trust, the project is built around explanations, privacy-by-design, and user agency (clear controls, no surprise edits) to represent a model for applied AI for "everyday" use.

1.7 Existing Work

Table 1:Existing Work

Citation	Summary	Contribution	Limitation (for mobile consumer use)
Hidayati et al. (2019)	Deep CNN embeddings for fashion similarity.	Validated image-based retrieval for apparel.	Focused on single-item retrieval no mood/occasion/weather or wardrobe context.
Liu et al. (2020)	Collaborative filtering using purchase history.	Improved prediction with behavioral signals.	Struggles with cold start weak intent-now modeling in mobile sessions.
Kang et al. (2021)	Outfit compatibility learning.	Shifted from item-level to outfit-level thinking.	Lacked personalization to user mood/wardrobe; no rationale/explainability.

Zhou et al. (2022)	ANN-based visual search over large catalogs.	Demonstrated scalable, fast retrieval.	Sensitive to non-studio images not fused with context-aware re-ranking.
Wang & Chen (2023)	Hybrid models (visual + text + context).	Advanced semantic reasoning for recommendations.	High complexity caused latency limited guidance for mobile UX constraints.

Summary of Gap : Prior research has presented visual similarity, collaborative filtering, and outfit compatibility individually, yet there remains a collective gap for mobile: incorporating powerful image-based search, contextual outfit planning (mood/occasion/weather/time), wardrobe-aware personalization, and brief explanations in a low-latency application. Lufyco bridges this gap.

1.8 Research Gap

Major mobile applications make limited use of wardrobe-oriented, situational reasoning. Prototypes do exist on image retrieval and style recommendation ideas and while they may be successful they are often used in isolation, are computationally heavy and are removed from live mobile specifications (e.g. bandwidth, latency, incremental indexing). There are very few systems that allow the use of:

- Visual Embeddings + vector search that are robust to user photos,
- Situational (i.e. mood, event, weather, time horizon) fusion,
- Wardrobe graphs (i.e. complementarity, capsule planning), and
- Explainability to increase trust and editability in a single, mobile-first low latency framework.
- Lufyco is deliberately positioned to fill this integration gap.

1.9 Contribution to Body of Knowledge

Visual Retrieval at Mobile Scale: A built in method to generate fashion embeddings and perform nearest-neighbor searching recommendations that support non studio images as well

as appearance and lighting fluctuations occurring in long tailed categories while being easy to incrementally update in a mobile context for something new and fresh.
Context Fusion for Complete Outfit: A flexible Layer of context that will encode commune, occasion, weather, time, and wardrobe constraints while issuing outputs of full outfits vs. simply one item still below required latency.
Explainability for Trust and Control: Brief, rationales and auditable traces user facing (ex. attributes, compatibility cues), creating a greater sense of relevance, trust, and editability, revealing part of the secret of responsible AI for consumer facing applications.
Evaluative Protocols for Mobile Fashion AI: Extraction of relevant retrieval metrics and recommendation metrics, user experience (UX) metrics and behavioral outcomes (engagement/retention), as well as to establish a reproducible evaluation template.

1.10 Research Challenges

Designing Lufyco involves several interconnected challenges:

- Visual Robustness & Freshness – To ensure that embeddings are still valid when the user's photos are troublesome and to allow raking to be incrementally reindexed without needing to take the system down.

- Context & Outfit Optimization – The ability to combine explicit rules (such as order of dress) and learned preferences (such as color combos or play with silhouettes) while being open to creativity from the user.

- Cold Start & Sparsity – Providing a strong content-based default for new users and items using the wardrobe and context to overcome data sparsity.

- Latency & Mobile Constraints – Getting performance for under a second in mobile networks by leveraging caching, compressing content, etc.

- Data Quality – Normalizing attributes such as color or fabric, cleaning near duplicate items, or variants and multi-attribute conflict.

- Explainability & UX – To keep recommendations concise and simple consistent with garter ranking and simple to edit without overwhelming the user.

- Privacy & Fairness - Protects user's wardrobes and preferences from other users, provide transparency into settings, protect against narrow loop recommendations.

1.11 Research Questions

- RQ1: How can image-based fashion retrieval (embeddings, indexing, re-ranking) remain accurate, fast, and fresh on mobile across heterogeneous catalogs?

- RQ2: Which contextual features (mood, occasion, weather, wardrobe embeddings, time horizon) most improve complete outfit quality, and what fusion strategy (rules vs. learned weights vs. graph constraints) performs best under latency constraints?

- RQ3: How do explainable recommendations (visual/attribute rationales, wardrobe-compatibility traces) affect user trust, satisfaction, edit behavior, and conversion intent compared to black-box baselines?

- RQ4: What architectural and caching strategies ensure low latency, high availability performance for retrieval and recommendation in a mobile-first environment?

1.12 Research Aim

The Coaltion's task was to generate, design, and validate a mobile-first, consumer exclusive fashion application named Lufyco that led itself to strong image retrieval process, context aware AI Stylist that draws from the user's digital wardrobe, and provide explainable and editable outfit selections to reduce decision fatigue, enhance shopper confidence, while being compliant with mobile latency and privacy.

1.13 Research Objectives

Table 2:Research Objectives

Objective ID	Research Objective	Description	LOs Mapped	RQs Mapped
R01	Literature survey	Review prior works in image retrieval, outfit compatibility, contextual recommenders, and explainable AI for fashion.	LO1, LO4	RQ1, RQ2, RQ3

R02	Gap analysis for mobile UX	Identify limitations in existing mobile fashion apps (latency, explainability, intent capture, wardrobe use).	LO2	RQ1, RQ2
R03	Data collection & preprocessing	Build multimodal datasets: user-like photos, catalog images, attributes; weather/mood/occasion schemas; wardrobe metadata.	LO3	RQ1, RQ2
R04	Image-based retrieval	Train ViT/CNN embeddings; implement ANN search with re-ranking; evaluate under user-photo noise.	LO1, LO4	RQ1
R05	Context-aware outfit recommender	Fuse mood/occasion/weather/wardrobe/time; optimize for latency; evaluate Precision & ablations.	LO1, LO4	RQ2
R06	Explainability module	Generate concise rationales consistent with ranking; test effects on trust, satisfaction, edit behavior.	LO1, LO3	RQ3
R07	Mobile app UX & performance	Implement low-latency flows, caching, and offline-friendly states; measure SUS and time-to-decision.	LO5, LO7	RQ1–RQ4
R08	End-to-end evaluation	Compare against baselines report UX + behavioral outcomes; document reproducible pipelines.	LO1, LO3, LO4	RQ1–RQ4

1.14 Project Scope

1.14.1 In Scope

Mobile app (Android/iOS) for customers that has the following:

- Image-based clothing retrieval from user images/screenshots.
- Context-aware AI Stylist + user inputs based on mood, occasion, weather, time of day.
- Digital wardrobe: upload items with auto-tagging by category, color, season, fabric.
- Editable outfit recommendation with alternatives displayed and a simple swap interaction.

- Save/share style cards, wishlist, and simple browsing/search filters.

1.14.2 Out Scope

- Shop-owner/merchant dashboards, inventory or pricing tools (per request).
- Full-scale payments, order logistics, and marketplace operations.
- AR/VR try-on and 3D avatars.
- Heavy cross-border localization, and complexity in tax/shipping compliance.
- Large commercial deployment across retailers globally (beyond project timeline).

1.15 Hardware/Software Requirements

1.15.1 Hardware Requirements

Table 3:Hardware Requirements

Component	Specification
Development Machine	Processor: Intel Core i7 / AMD Ryzen 7 or higher RAM: Minimum 16 GB GPU: NVIDIA RTX 3060 or higher (for model training) Storage: 1 TB SSD (datasets + embeddings index)
Model Training Environment	NVIDIA RTX 3060/3070 or higher with 6–8 GB VRAM
Inference / Hosting (Cloud)	4–8 vCPUs, 16–32 GB RAM, optional GPU for batched embedding generation Scalable object storage for images and indices (e.g., AWS S3) CDN integration for low-latency mobile delivery
Operating System	Windows 10/11 (64-bit) or Ubuntu 20.04 LTS (for development and testing)
Mobile Device (Testing)	Android smartphone, 6 GB RAM, Android 11 or later

1.15.2 Software Requirements

Table 4:Software Requirements

Category	Software / Tool	Purpose
Mobile Application	React Native	Mobile UI and frontend development
Backend Framework	Python and Node.js	REST API and business logic implementation
Machine Learning Stack	PyTorch	Model training and inference
Data Storage	MongoDB	User profiles, wardrobe, and contextual data
Caching	Redis	Session management and fast data access
Integrations	Weather API, optional calendar intent	Context-aware outfit and event recommendations
DevOps / Deployment	GitHub, Swagger / OpenAPI	Containerization, version control, documentation
Security & Privacy	TLS/HTTPS encryption, device keychain for tokens	Secure communication and authentication
Data Protection	Encrypted wardrobe data, opt-in telemetry, user data export	Privacy compliance and transparency
Design / UI Tools	Figma	UI/UX design and marketing poster creation
Testing Tools	Postman	API and system-level testing

1.16 Chapter Summary

This chapter presented Lufyco, a mobile only, consumer facing fashion app aimed at reducing decision fatigue and styling indecision by bringing together image-based discovery, context-aware AI styling, and reasoning in a digital wardrobe along with explainable recommendations. We presented the problem and background, formed the problem statement, and discussed the motivation for a context-based, digital closet-based approach. Included a review of prior work,

demonstrated a clear gap in research in regard to integrated mobile systems, contributed to practice and research, and touched on a range of important considerations; namely, robustness, latency, explainability, and privacy.

Next, we framed the research questions, aim and objectives, corresponding to learning outcomes; considered the scope of the work, ruling out merchant tooling, and limiting the focus to the end-user experience on mobile; and delineated the hardware/software requirements for development, inference, and evaluation as the basis for further work. With the foundation laid for this research, we will move on to Chapter 02: Literature Review, where we will review the state-of-the-art in fashion image retrieval, modeling outfit compatibility, context-aware recommendations, and explainable AI, and derive from this review the methodological decisions that underpin Lufyco's architecture and experiments.

2 CHAPTER 2: LITERATURE REVIEW

2.1 Chapter Overview

This chapter assesses work that is relevant to Lufyco, which we define as a mobile-only app that allows users to discover fashion images and that provides context-aware outfit suggestions for the user. In technical terms, we want to clarify the formulation of the problem we presented - filtering is not enough (even with Augmented Reality) - you must consider mood, events, weather (including outdoor conditions) and what the custom parts of the user's wardrobe. The literature we reviewed was broken down into five major categories of prior work: image retrieval, outfit compatibility, modeling context, digital wardrobes, and explainability, addressing constraints of mobile latency and bandwidth. Guided by practical questions for Lufyco, we wanted to know if similar items can truly compliment each other, can we recommend a complete outfit for a rainy evening outside, and can users edit and see their reasons? The open problems (robust snapshots, cold-start, fast, trustworthy recommendations on-the-device) lead in several order increments to sketch a coherent architecture for a mobile-first Lufyco app.

2.2 Problem Domain

2.2.1 Overview of AI-Powered Fashion Discovery and Styling

Fashion shopping is not simply about purchasing a single item. More often people need a complete outfit that is appropriate for a specific situation (work, wedding, date night, interview), suitable given the weather, and can be worn with other items they already own. Conventional mobile shopping flows require users to type keywords and apply filters. This is difficult when users do not know the exact style vernacular. Additionally, conventional mobile shopping is also context-neutral, so results may be relevant in name only, but incorrect for the user in their actual context.

AI changes this in two aspects:

1. Image based discovery: A user uploads a picture or screenshot of an outfit they like. A vision model creates a set of numbers representing an “embedding” of that image, which searches the catalog for items that are visually similar. This is useful when users cannot express what they want in words (Kiapour et al., 2015; Liu et al., 2016; Ge et al., 2019).
2. Context aware outfit recommendation: at recommendation layer, contextual factors such as the user mood, occasion, weather and time (now it is an event in the future), as

well as the full digital wardrobe (what clothes they have). The algorithm will then recommend complete outfits (not just a single item) and rationalize why each item is suitable in the specific situation (He & Hu, 2018; Wang et al., 2019; Abugabah et al., 2020).

Clothily's problem domain lies at the intersection of these two aspects, within mobile constraints (low latency; limited bandwidth and small screen) and privacy expectations (wardrobe photographs and preferences are personal information) (Johnson et al., 2017; Google Research, 2020–2024).

2.2.2 Why Context Matters in Clothing Decisions

Context is all about realizing the difference between a “nice dress” and the “right dress for tonight

There are four key things to consider:

- Mood: Customers may want to look confident, poised, fun, or formal. Their mood impacts their choice of color, fabric, or the overall silhouette (Abugabah et al., 2020).

- Occasion: Each occasion and location has its style norms and comfort requirements for clothes, including the office, an interview, a wedding, a party, a religious ceremony or next event, or for travel purposes (He & Hu, 2018; Wang et al., 2019).

- Weather & Season: Temperature, rain, humidity, wind, and daylight creates considerations around fabric and layering choices (i.e.- breathable linen vs. water-resistant or standing outer) (Abugabah et al., 2020).

- Time Horizon: Now means they need recommendations for immediate wear that lead with a wardrobe first solution; the future means the app can offer recommendations for items to shop for to fill wardrobe gaps (Wang et al., 2019; Abugabah et al., 2020).

Most applications ignore these signals and/or measure them indirectly. Lufyco considers them first-class inputs, so that the app's recommendations appears suitable, comfortable and timely.

2.2.3 The Role of the Digital Wardrobe

A digital wardrobe is a means of systematically storing a list of things that a user owns by uploading photographs of their clothes, and auto-tagging (e.g., category, color, pattern, fabric, season). This is essential for the real-world usability aspect:

- The app can create outfits from what the user already has, saving them potential costs and potential returns (Wang et al., 2019; Graph Outfit, 2023–2024).
- The app can identify gaps in what the user owns (e.g., An alert that states “you have no neutral outer layer for rain” will pop up only when faced with a rain day) and provide suggestions only when needed (Abugabah et al., 2020).
- Over time, the wardrobe acts as a memory of the user's personal style and will become an upward feedback loop of improved suggestions independent of prior purchase history (He & Hu, 2018; Wang et al., 2019).

It also enables explainability (e.g., "this blouse goes with your navy trousers, and it is humidity-friendly") and editability (e.g., swap the shoes out, and change the palette) (Hou et al., 2019).

2.2.4 Application Areas

2.2.5 Mobile-First Consumer Shopping

Lufyco is designed for quick decision-making while on the go: taking a picture in a shop, saving a pic of an outfit you want to wear from social media, picking an outfit during your commute, or scouring for an outfit online during a break at work. The app needs to provide useful suggestions in less than a second, a short description, and can be edited with the tap of the finger (Johnson et al., 2017; Google Research, 2020–2024).

2.2.6 Personal Styling and Outfit Planning

Users will sometimes (in fact, they will often) anticipate events (wedding, interviews, festivals) or follow routines (a workweek, gym, or school run). The app can preemptively schedule multiple looks per event, account for the weather forecast, and provide options if the user is more comfortable with changes He & Hu, 2018; Wang et al., 2019; Abugabah et al., 2020).

2.2.7 Confidence, Returns, and Sustainability

Offering clear, context-aware options reduces waste and mis-purchase, which may reduce returns. Supporting users in maximizing what they own leads to sustainable behavior (fewer duplicate purchases and smarter fills for actual gaps in wardrobe) (Wang et al., 2019; Abugabah et al., 2020).

2.3 Challenges in Context-Aware Fashion Recommenders

2.3.1 Data Quality and Catalog Variability

Colour, fabric, and silhouette are some attributes that are labeled differently in catalogs. Lighting, background, and pose change the way images look. The system needs to normalize attributes, and de-duplicate nearly identical items, and reconcile conflicts (e.g., "rust" vs. "terracotta"). Bad data results in bad recommendations (Liu et al., 2016; Ge et al., 2019; Gao et al., 2020).

2.3.2 Visual Retrieval Robustness

User images can contain noise: for example, captured images could include mirrored self-portraits, insufficient lighting from indoor capture, partial occlusion, or background clutter. For models that were trained on studio images, many will fail when presented user images. The retrieval system must be robust with the domain shift and be able to identify relevant visual features despite the noise (Kiapour et al., 2015; Ge et al., 2019).

2.3.3 Real-Time Performance on Mobile

People expect quick responses. Embedding generation, vector searches, re-ranking, and explanations should occur within tight latency limits. Smart caching, progressive loading, and smart indexes are critical to optimizing performance (Johnson et al., 2017; Google Research, 2020–2024).

2.3.4 Cold Start and Sparsity

New users and new items do not have much historical context. The system needs content first strategies (visual and attribute signals) and lightweight context capture (mood/occasion/weather) to make great recommendations, quickly (Abugabah et al., 2020; Gao et al., 2020).

2.3.5 Privacy, Security, and User Control

Wardrobe images and style visualizations can be particularly sensitive. The application must ensure zip data is encrypted while in transit and in use, must limit personal data collection, and

must explicitly offer users the ability to edit or delete data. Explanations should also be useful but should not disclose private information today.

2.3.6 Explainability and Trust

If users cannot ascertain why the system made a suggestion, they will not be able to trust the application or follow the recommendations. Explanations should be brief and specific, and editable by the user (“I have suggested water-resistant shoes due to the rain.”). Explanations must also be honest - an explanation of why an item of clothing was ranked or matched should reflect how the model actually ranked the items (Hou et al., 2019).

2.4 Proposed Architecture (Problem-Domain View)

The suggested architecture for Lufyco is a high-level , vendor-neutral architecture that illustrates how each component of the system interacts and contributes to a smart, secure, and seamless fashion experience. It describes how data flows through the system, from mobile application, backend services, and AI models, without depending on vendor-specific technologies (Liu et al., 2016; Ge et al., 2019).

2.4.1 Mobile App (Client)

The mobile application serves as the primary user interface, where users can upload photographs or screenshots for visual search or set a mood, occasion, and time, or upload items in their wardrobe. The design emphasizes simple forms, toggle chips to provide context, swipeable outfits, and one-tap edits like a swap or recolor. Privacy is maintained by offering local previews, clarifying uploads with explicit consent, and clearing an upload with an easy delete option.

2.4.2 API Gateway

The API Gateway serves as a connection between the mobile application and backend services. It receives all requests from the application and routes the request to the correct internal service,

gathers its responses, applies authentication, and manages rate limits. This layer ensures secure, robust, and scalable communication between components.

2.4.3 Image Embedding Service

The service takes images from users or images in the catalog and creates visual embeddings. These are built by deep vision models like CNN (Convolutional Neural Network) or ViT (Vision Transformer). These deep neural networks also do preprocessing tasks such as cropping, resizing, and normalization. As a further safeguard in obtaining reliable results, the service employs techniques such as background removing capabilities and data augmentation (Liu et al., 2016; Ge et al., 2019.

2.4.4 Vector Search Index

The Vector Search Index holds all embeddings and has support for approximate nearest neighbour (ANN- Approximate Nearest Neighbor) so that users can quickly "find similar" items. The index is intended to update itself automatically when new items are added to the catalog, ensuring that all search results are new and relevant (Johnson et al., 2017; Google Research, 2020–2024).

2.4.5 Catalog & Attribute Normalization

This module sorts through all catalog information to ensure cleanliness, completeness, and accuracy. All relevant catalog attributes, which could include color, fabric, pattern, and silhouette, are normalized, including merging near-duplicates and resolving conflicting attributes (e.g., if two different vendors use different names for the same color). This process is essential for a dependable, comprehensive product database (Gao et al., 2020; Hou et al., 2019).

2.4.6 Digital Wardrobe Service

The Digital Wardrobe Service stores item information in the user's wardrobe, including the user-uploaded items themselves. The Digital Wardrobe Service will automatically tag these

items with a range of relevant information: first, generalized information for the wardrobe item, such as season and color, and potentially even style category (e.g., dress, pants, and sweater). Additionally, it can tag items in terms of material and fabric. The Digital Wardrobe Service generates embeddings for these wardrobe items, which give a veritable dimension allowing the platform to make comparisons in the form of wardrobe to catalog combinations (Wang et al., 2019; Graph Outfit, 2023–2024).

2.4.7 Context Service

The Context Service collects user inputs, e.g., mood, occasion, etc., while also automatically getting weather data. The context is then converted into machine-readable features, e.g., "humidity high," "semi-formal," and/or "evening outdoor." Context is critical so that the provided recommendations are aligned with a user's real-world context (Abugabah et al., 2020).

2.4.8 Full-Set Generation & Re-ranking

This is the primary recommendation engine. It generates full outfits using the user's closet in the first priority and uses catalog items to fill in the gaps. It invokes both rules (e.g., "no suede in rain") and learned preferences for items such as color, and shape balance in style. Finally, it re-ranks the output based on contextual relevance, compatibility with closet contents, and user preferences to output the best outfit (He & Hu, 2018; Wang et al., 2019; Graph Outfit, 2023–2024; Abugabah et al., 2020).

2.4.9 Explanation Generator

The Explanation Generator provides clear reasons for the recommendation (e.g., "pairs with your navy chinos; lightweight fabric for humidity; neutral colors pair with your calm mood"). It will also provide suggestions for swapping items (e.g., "swap to white sneaker for comfort"). This makes AI choices straightforward to understand and trustworthy (Hou et al., 2019; Gao et al., 2020).

2.4.10 Feedback, Telemetry, and A/B Testing

The system receives feedback from users to learn user preferences in an anonymous way (e.g., accepted, skipped, or edited sets of outfits). A/B testing also occurs to determine different styles of explanations or strategies of ranking outfits. All data gathering is optional. All data gathering has strict privacy policies.

End-to-end flow (example)

When a user uploads a street-style image and selects mood = confident, occasion = dinner, and time = tonight (and weather = light rain) the process begins with image embedding. The remixed wardrobe, and each item's Vector Search Index returns visually similar items. Then, the Wardrobe and Context Services add personal data and environmental data. Next, the Outfit Engine builds three complete outfits, utilizing the wardrobe first, without a moisture layer then adds a water-resistant top layer. During a re-ranking step prioritizing outfit fabrics (i.e., moisture-friendly), it favors fabrics over a preferred color. Then shows the approll out the best outfit with a short a "why" and option of garments to swap (Kiapour et al., 2015; Johnson et al., 2017; Abugabah et al., 2020).

Summary
This architecture is fast, private, and reliable, giving the user a personalized, weather-aware, explained fashion recommendation supported through their privacy and preferences.

2.5 Existing Work

Research that informs the development of Lufyco is examined in this section. We describe six components of the workflow: Image Embedding elucidates how a model transforms clothing images into multidimensional vectors (numbers), representing shape, color, pattern and style Liu et al., 2016; Ge et al., 2019). ANN / Indexing portrays how we are able to query rapidly in millions of vectors on a phone-speed connection (Johnson et al., 2017; Google Research, 2020–2024). Context Features demonstrates what mood, occasion, weather and time are incorporated into a recommendation to make it applicable in the real world (Abugabah et al., 2020). Outfit Composition / Compatibility blends aspects focused on how to construct combinations of pieces into a cohesive ensemble that "works together." Explainability discusses how the system informs the user why a suggested item was made, in short, credible explanations (Hou et al.,

2019). Evaluation & Datasets represent what data and metrics we will use to test if the system works (Kiapour et al., 2015; Liu et al., 2016; Ge et al., 2019).

2.6 Literature Survey (pipeline-aligned)

Pipeline columns used below: Image Embedding, ANN/Indexing, Context Features, Outfit Composition/Compatibility, Explainability, Evaluation & Datasets.

Table 5:Literature Survey

No	Citation (short)	Year	Method	Main Contribution	Main Limitation for a Mobile App
1	DeepFashion (Liu et al., CVPR)	2016	Large labeled fashion dataset for retrieval	Standard benchmark for clothing retrieval; widely used	Images are mostly curated, less like messy phone photos
2	DeepFashion 2 (Ge et al., CVPR)	2019	New benchmark with detection, pose, segmentation, retrieval	Handles pose/occlusion closer to real world	Heavy models; compute-intensive
3	Street-to-Shop (Kiapour et al., ICCV)	2015	Cross-domain retrieval between street and shop photos	Pioneered real-world fashion retrieval problem	Not outfit level; little context
4	FashionNet / NOR (He & Hu, arXiv)	2018	Outfit recommendation and comment generation	Moves beyond single items to outfits	No explicit weather/mood; latency not discussed

5	MLCN (Wang et al.)	2019	Multi-layer comparison for outfit compatibility	Better outfit scoring and mismatch detection	Lacks real-time context and wardrobe awareness
6	Graph Outfit (GNN-based)	2023 – 2024	Graph neural network for outfit composition	Captures complex item relationships in sets	Data-hungry; slow updates for real-time
7	FashionBERT (Gao et al.)	2020	Cross-modal Transformer (image + text)	Strong image–text matching with attention	Heavy model; needs distillation for mobile
8	SAERS (Hou et al.)	2019	Attribute-based explainable recommendations	Clear visual explanations via attributes	Needs clean attributes; adds complexity
9	Context-Aware Outfit Rec (Abugabah et al., MDPI)	2020	Combine user reviews and image regions	Shows context improves recommendation relevance	Weather/mood not explicit; latency unknown
10	FAISS (Johnson et al.)	2017	GPU-optimized ANN for vector search	Makes large-scale retrieval practical	Requires index tuning and ops effort
11	ScaNN (Google Research)	2020 – 2024	ANN with score-aware search for high recall and low latency	Great recall/latency trade-off for production	Needs infrastructure and setup

2.7 Critical Evaluation

Below we review what the literature tells us, in plain language, and what it means for Lufyco.

2.7.1 Visual Fashion Retrieval (from user photos to shop items)

- What the papers did: Early studies such as Street-to-Shop set out to define a difficult, though feasible task: take a messy, real photograph (mirror-selfie photo, street photo) and find that same item in a tidy shop photograph. This uncovered the domain shift problem: lighting, pose, background, and cropping are markedly different between user-facing and studio photographs (Kiapour et al., 2015).
- Datasets that aided: DeepFashion and DeepFashion2 provided a large labeled dataset and good baselines. DeepFashion2 included notable pose and occlusion as well as consumer↔commercial pairs closer aligned to actual use. These datasets are now standard for training and testing visual search (Liu et al., 2016; Ge et al., 2019).
- What's still hard: Even with those datasets, nevertheless, models will fail on variations of non-studio photographs (odd angles, no light, occlusion, off angles) (Ge et al., 2019).
- What Lufyco should do: Pre-train on DeepFashion/DeepFashion2 and then fine-tune on something with more user-like photographs combined with strong augmentations (random crops, color jitter, some blur, and background clutter), ultimately reducing the domain gap and yielding better reliability of "upload-a-photo-find-similar" (Liu et al., 2016; Ge et al., 2019).

2.7.2 Outfit Compatibility and Full-Look Recommendation

- Importance: When it comes to buying clothes, users usually buy more than a single item and ask each purchase fits in with their existing wardrobe.

- What the papers did: Papers, such as FashionNet/NOR and MLCN, determine if items are compatible or compatible with each other. Newer graph models, however, represent outfits as graphs in which items are nodes, and edges represent if the pieces work together or not like. These methods outperform against scores based on item-item similarity, particularly in a ranking of complete looks (He & Hu, 2018; Wang et al., 2019; Graph Outfit, 2023–2024).

- What's missing: Most of these books do not include the context (like weather, mood, or event) or the wardrobe of the individual user (Wang et al., 2019; Graph Outfit, 2023–2024).

- What Lufyco should do: Implement a compatibility model (for example graph or and layered comparison) that provides a compatibility model layered with context and the users own digital wardrobe graph. This moves the model from "compatible in front of the module" to "compatible with you and in context with your situation." (Graph Outfit, 2023–2024; Abugabah et al., 2020)

2.7.3 Multimodal / Cross-Modal Understanding (image + text)

- The focus of the papers: FashionBERT leverages a transformer component for aligning images and text (title, attributes), treating image patches in a manner akin to tokens and fusing those tokens with the words, thus improving the matching process and the understanding of the attributes. Attention maps can also be used for weak explanations of the learned model. (Gao et al., 2020)

- Trade-off: These models tend to be large and slower than you'd like for phones. (Gao et al., 2020)

- What Lufyco should do: Use cross-modal models to improve the retrieval process (e.g., better understanding of attributes like "linen" or "A-line", or "midi"), but use a relying on lighter, distilled versions or push only heavy parts to a server with some caching to keep the app snappy. (Gao et al., 2020)

2.7.4 Adding Context (mood, occasion, weather, time)

- What the papers did: In some of the work, there is useful context based on using reviews or text from the scene, which also show that providing context to your recommendations supports them. Even so, for the most part, the papers do not even acknowledge context around weather, mood, occasion, or time horizon in a proper,

first-class way. Even more rarely is latency discussed, especially on mobile. (Abugabah et al., 2020)

- What we [Lufyco] should do: Elevate context as a first-class citizen. Use hard rules (e.g., don't suggest suede if it's going to rain) coupled with learned preferences around color, silhouettes, etc. Do ablation studies to prove to myself/our team which context signals provide the largest gain. (Abugabah et al., 2020; Wang et al., 2019)

2.7.5 Explainability (short, honest reasons)

- What the articles described: SAERS provided semantic regions (neckline, sleeves, etc.) as a reason the item was suggested. Also, there are methods that have a way to articulate which specific feature of which item made the outfit cumulate better or worse. Explanations help build trust with users and allow for edits to the look. (Hou et al., 2019)

- Caution: It is important explanations match the actual ranking signals or users feel tricked. (Hou et al., 2019)

- What Lufyco can do: Give a brief, uniform reason that aligns with an actual attribute (e.g., “matches your tan trousers; breathable for humid weather”). Include one-tap edits (swap color, new shoes) for user control. (Hou et al., 2019)

2.7.6 Large-Scale Indexing and Mobile Performance (speed matters)

- Why this matters: When conducting an image search on a phone, it must feel instant.
- What the papers/tools did: FAISS (Meta/FAIR) and ScaNN (Google) are the two primary libraries to obtain very fast nearest-neighbor search over embeddings. FAISS supports GPUs and multiple index options: IVF-PQ, HNSW, etc. ScaNN was developed to optimize inner product similarity search, and it has developed even newer techniques like SOAR to yield faster results. (Johnson et al., 2017; Google Research, 2020–2024)
- What Lufyco should do: Lufyco could feasibly leverage FAISS or ScaNN to search over compressed indexes and use warmed caches. It could maintain a fresh index as the

catalog changes (incremental updates), and ensure some rudimentary balance between recall and latency in search results. (Johnson et al., 2017; Google Research, 2020–2024)

2.8 What's still missing — and how Lufyco is different

Contextual fusion does not receive noticeable attention in the literature. A lot of models understand what is a match, but do not understand what is a match for this user, for this event, in this weather, right now (Abugabah et al., 2020). Lufyco incorporates mood/occasion/weather/ time into the main scoring function, as opposed to an afterthought.

Wardrobe reasoning is an uncommon perspective. Most work, recommends from public catalogs. Lufyco builds, and uses a digital wardrobe, so the app recommends from wardrobe outfits first, then adds shop items where necessary (Wang et al., 2019; Graph Outfit, 2023–2024).

Explaining while in phone time, has not received much attention in the literature. Lufyco will keep reasons short and connected to reality ("goes nice with your navy chinos; suggested water-resistant shoe due to rain") and a link to actionable edits. (Hou et al., 2019)

Mobile latency engineering (sub-second feel) has also been an omitted concern in the literature. Lufyco will combine lighter models, ANN tuning (FAISS/ScaNN), caching, and then progressive responses (show top results faster, refine in the background if necessary). (Johnson et al., 2017; Google Research, 2020–2024)

2.9 Technological Review

This section teaches, in layman terms, the main technical requirements to make Lufyco functional on a mobile phone device. Lufyco must allow a user to upload a photo or screen capture, and be instantly populated with similar items of clothing, and then build out a complete outfit that is consistent with users mood, occasion, weather and items they already have in their digital wardrobe. To accomplish this in an efficient manner, we rely on three types of data: images of clothing (photos), text data such as titles, colors, fabrics, and categories, and light context such as simple weather flags and user input (i.e. mood, event). We then clean and

normalize data, and standardize the data, so that we can evenly compare item similarity, and respond quickly. Behind the scenes, we create compact vectors for the images and text data so the phone can quickly search, use simple rules to merge context, and to generate brief but honest explanations. ("matches with your navy trousers; good for rain"). All of these decisions were made based on accuracy, speed, privacy, and overall a quiet interaction with the user (Liu et al., 2016; Ge et al., 2019; Johnson et al., 2017; Google Research, 2020–2024; Hou et al., 2019).

2.9.1 Dataset and Preprocessing

Lufyco utilizes multiple data sources and runs rigorous preprocessing so the models have consistent and quality inputs. First, we use catalogs with product images and attributes (e.g., category, color, fabric, and season) and we align all naming to a common vocabulary. Second, with the user's permission, we include inspiration images and digital-wardrobe images to work through the messy data of real-life images. Third, we pretrained on public fashion datasets to learn solid visual features, followed by tuning based on our own data. Fourth, we create a simple weather feed which we adjust to generate small and stable signals (e.g., rain true/false and hot/comfortable/cool). Preprocessing includes resizing images, normalizing colors, reducing background noise, cleaning text, compressing vectors for fast searching, and removing duplicates or nothing data. Good preprocessing helps with accuracy and speed, and also helps deliver consistent explanations, because the underlying labels and features are clean (Kiapour et al., 2015; Liu et al., 2016; Ge et al., 2019; Johnson et al., 2017).

2.9.2 Data Sources and Acquisition

We collect our data from four sources which together are complementary. (1) Catalog data: partner shops (or a pilot catalog) give us multiple photos of each item, and a short text (title, category, color, fabric, season). We normalize all terms - for example navy and dark blue become simply navy. (2) User photos: users upload a street photo or screenshot they want “to find similar, and they can also photograph items in their own closet and build a personal digital closet; we securely save photos, only with consent, and have users delete any photos simply. (3) Public fashion datasets: large, publicly available and well-known sets are used to teach the models about clothing shapes, patterns, and characteristics, which is particularly useful when modeling from messy phone images, or styles from vernacular sources are broad. (4) Weather

feed: we can request a lightweight data feed via API, with simple context signals such as, rain or heat, so the AI Stylist, can help to make practical decisions. We appropriately document licenses, any personal data remains private by default, and always elicits only the minimum context, to convene with the user (Liu et al., 2016; Ge et al., 2019; Kiapour et al., 2015).

2.9.3 Preprocessing Techniques

Preprocessing converts raw inputs into tidy features for the model while keeping the elements users care about: silhouette, color, pattern, and seasonality. For images, we standardize sizes and aspect ratios, normalize color channels, and where possible center the item in the frame (crop, or simple segmentation) so that backgrounds do not distract from the model. For text, we lowercase, trim punctuation, and unify synonyms into a concise, human-readable vocabulary. For context, we convert raw weather values into small, consistent features (e.g., temperature bands, rain flag) so recommendations do not jump around. We also dedupe items, clean up the conflicting labels, and compress embeddings so that queries are fast even on mobile networks. In the end, we reduce distracting variables to maintain fairness across items and allow ranking and explanations to remain consistent and understandable (Ge et al., 2019; Gao et al., 2020; Johnson et al., 2017).

2.9.3.1 Noise Filtering and Normalization

User images can be darker, blurrier, tilted, or have background clutter, so our first step is to make images consistent and clearer. We resize user images to the same size, center crop or pad to the same shape, and apply gentle color normalization, so images taken on different phones and in different lighting conditions will match. As necessary, we can do light denoising or exposure correction, applying minimal adjustments to preserve true color and textures. Where applicable, we crop or mask around the garment, so the model's attention is directed to the garment shape and pattern, delaying background focus until attention to garment shape and pattern is made first. In the event documented text describes catalogue items, we have removed extraneous characters as appropriate, standardized to one spelling of documents, and mapped related words to one stable term in the event of user language variations. We translate weather from raw numbers, to easily understandable flags and bands (e.g. rainy, hot). This combination

reduces noise and provides the model with consistent, meaningful inputs (Kiapour et al., 2015; Ge et al., 2019).

2.9.3.2 Feature Scaling and Standardization

After an image goes through a vision model, it is transformed into a vector (a list of numbers) that represents the appearance of the image. We L2-normalize the vector, resulting in a length of 1, making cosine similarity equitable and robust for comparing items. This is essential to continue giving quick "find similar" search results. Simple numeric context features (temperature, humidity) are scaled via min-max or z-score, and categorical features (mood, occasion) are one-hot or small learned embeddings to integrate into, and produce completive similarities with the visual vector. Text features are kept lean and capped in length to accommodate speed. For the search engine, vector compression (i.e. product quantization) is often deployed, and an ANN index that favors recall and latency is employed. Each of these procedures optimizes the app to provide quality results fast, uses memory efficiently, and meets expectations for consistent behavior across devices and connectivity (Johnson et al., 2017).

2.9.3.3 Outlier Detection and Removal

Outliers can confuse both the training process and the end user, so we like to address them early and simply. Very low quality images (i.e., tiny images, extreme blur, heavy occlusion) are flagged for removal or re-upload. For duplicates and near-duplicates, we use perceptual hashes and embedding distance to group them, so top results do not have excessive similarity to others, and the resulting set is diverse. We check for attribute conflicts (e.g., "linen" and wool in one object) and either fix them or emphasize those attributes less, so they are less likely to mislead either the recommendations or explanations. We filter impossible values and smooth sudden spikes for context so the AI Stylist does not change between choices minute to minute. During pilot evaluations, if we see a wave of not-relevant feedback after a model change, we treat that as behavioral outlier feedback and quickly roll it back, then adjust the weights of its impact. These filtering strategies help ensure the system is accurate, stable, and trustworthy (Johnson et al., 2017; Gao et al., 2020).

2.10 Benchmarking and Evaluation

2.10.1 Evaluation Metrics for Lufyco Models

We evaluate Lufyco through a series of metrics that correlate with actual activity undertaken by users in the mobile app. The first category is the quality of clothing similarity search. When a user submits a photo or screenshot of an item, we want to know if the app is appropriately displaying actual similar clothing items at the top of their results. The second category is the quality of outfit recommendations. In this instance we want to evaluate if full outfits make sense for the user's mood and for the occasion, local weather, and clothing already saved in the wardrobe. The third category is attribute recognition. Here we test if the app correctly labels color, and appreciates fabric, category, and season, because clean labels and categories should make filter options and short why this explanations reliable. The fourth category is speed and efficiency. We assess whether results come through fast on typical phones and networks, and whether the model is small and lightweight enough to not have too much battery or data use. The fifth category is user experience. We look for signals that people find items more quickly, edit suggested items less often, trust the suggested explanations, save or share style cards, and come back to use the app again. We do both offline tests on data sets (so results can be reproduced) as well as small, controlled tests in the app on real devices (so results reflect real life) (Kiapour et al., 2015; Liu et al., 2016; Ge et al., 2019; Johnson et al., 2017; Hou et al., 2019).

2.10.1.1 Accuracy, Precision, and Recall

Accuracy indicates overall correctness. It is a straightforward measure, but it can obscure issues when labels are infrequent. For example, if the overwhelming majority of items are obfuscating black and only a minute number are burgundy, having accuracy on many items is not necessarily a good thing because the model could be accurate by guessing black too often. Precision is all about getting the question of “of the things we said were positive, how many were actually positive? answered correctly and avoids any misrepresentation like saying a dark navy dress is black or that a cotton shirt is linen. Recall is all about getting the question of all the things that were truly positive, how many of we good? answered correctly, and avoids missing the good options that the user would prefer. These are numbers that can be reported for each class (e.g., color or fabric) and an overall average. This way all or small classes aren’t

ignored because of poor accuracy for all classes, and the models learn to respond fairly across a variety of items (Liu et al., 2016; Ge et al., 2019).

2.10.1.2 F1-Score and AUC-ROC

F1-Score merges precision and recall into a single figure. It is useful when the data is not balanced because it rewards models that do not generate false alarms, whilst not generating too many missed cases. AUC-ROC shows how well the model separates positives from negatives across a number of different score thresholds. In practice we use F1 and AUC-ROC to help us choose sensible cut offs for the attribute checks, such as is this a wool? or is this water resistant?. Chose good cut-offs makes the labels less noisy before feeding items into search and outfit ranking. It reduces abrupt changes in behavior, preventing confusing results, such as fluctuating back and forth between navy and black for the same item as the lighting changes (Gao et al., 2020).

2.10.1.3 Computational Efficiency and Processing Time

Lufyco is a mobile application, therefore speed is not just an option; it is part of the user experience. We measure end-to-end time from when the user taps to the first visible results for image search and for outfit suggestions. We also decompose this time into portions server compute, search index lookup, and network transfer so we can fix the real bottleneck. We monitor model size and memory usage to keep the app lightweight on-device and in the cloud. We test battery impact during common activities, such as uploading a photo, viewing similar items, and saving a style card. We assess index freshness to determine how quickly new or updated items in our catalog appear in search and recommendations, because stale results lead to reduced trust. Lastly, we monitor throughput (how many queries per second a small server can handle) to ensure pilot tests go smoothly, with no smoothdowns (Johnson et al., 2017; Google Research, 2020–2024).

2.10.1.4 Ranking Metrics for Retrieval and Outfit Lists

To measure image search effectiveness, we look at Recall@K for the question "did at least one good match appear in the first K results?" and Precision@K for the question "of the first K results, how many are really relevant?" We also use MRR (Mean Reciprocal Rank) to gives

points where the first good match appears at the top of results, saving scrolling for the user. When we can, we summarize with mAP (mean Average Precision), which provides coverage for quality and order. For the outfits we leverage NDCG (Normalized Discounted Cumulative Gain) and precision@K to see if the best suggestions are near the top rather than buried. We also have measures for duplicate items so none of the top results are just a bunch of near-duplicates, along with coverage measures to ensure we still show rare styles, sizes, and colors. Good ranking means that users can find what they want more quickly and see better ideas sooner (Kiapour et al., 2015; Liu et al., 2016; Ge et al., 2019).

2.10.1.5 UX and Behavioral Metrics (SUS, Time-to-Decision, Edits, Saves/Shares)

We also assess how the app feels. We use the System Usability Scale (SUS) in order to provide a straightforward, standard score on usability. We record time-to-decision so we can assess how much time it takes a user to choose an outfit or add something to a wishlist. We track edit rate (i.e., how frequently a user swaps/tweaks suggested items), save/share rate (i.e., how frequently users save/share style cards) and return rate (i.e., how frequently they return for more). We conduct small, careful A/B tests where one group gets to see explanations or weather-aware re-ranking and the other does not. If the group with explanations makes a decision faster, edits less, and saves more, we know those features are helping in practice, not just theory (Hou et al., 2019; Abugabah et al., 2020).

2.11 Comparative Analysis of Existing Models

2.11.1 Benchmarking Studies

We maintain fairness in comparisons in the lab. We compare various image encoders on the same training and test splits, consistent image sizes, and consistent augmentations. We compare the search index types (i.e., HNSW, IVF-PQ, or a ScaNN like setup) under the same memory budget so the trade-offs of speed versus quality are genuine, rather than the result of additional resources. For outfit building, we compare simple pairwise compatibility models against set-based or graph-based models on the same outfit datasets. We then incorporate a small context re-ranker that considers mood, occasion, and weather to evaluate its contributions to list quality. For attribute labeling, we measure a small image-only head against a small cross-modal (image + text) head to evaluate whether titles and descriptions improve reliability for color and fabric. Lastly, we enable explanations and determine whether the reason presented is

compatible with the signals that actually produced the ranking. If the explanation is unfaithful, we discard that configuration, even if it has higher unadjusted scores, because we want models that we trust Johnson et al., 2017; Google Research, 2020–2024; He & Hu, 2018; Wang et al., 2019; Gao et al., 2020; Hou et al., 2019).

2.11.2 Real-World Performance and User Experience

Next, after testing in the lab, we look at real phones. We are measuring time-to-first-result, 95th-percentile latency, top-K diversity, and index freshness while doing small updates to the catalog, to mimic real change. We're looking at cold start scenarios, where a new user has no wardrobe and limited history, to ensure the content-based defaults and context-based defaults continue to feel useful. We'll be testing a simple, fast rules-plus-rank context layer versus a heavier learned fusion context layer selection and go with the one that meets our speed objectives without sacrificing quality. In short sessions with users, we look at whether or not explanations remove edits, whether weather-aware suggestions make sense, and whether or not people complete the styling function faster than if they just used basic search and filters. If we observe repeated near-duplicates, miscolor claims, or laggy responses, we treat that as blockers, even if the offline metrics are high, because the in-hand experience withstands what the user needs (Johnson et al., 2017; Google Research, 2020–2024; Hou et al., 2019; Abugabah et al., 2020).

2.11.3 Challenges in Benchmarking Multimodal Systems

Lufyco pulls from images, text, context, and a personal closet, rather than claiming perfect measures through a single benchmark, so we will end up having to stitch together a number of tests which will makes it more difficult to do comparisons. Style is subjective - two totally different outfits can both be "good," even if a metric says only one is good. Offline wins also do not always seem to behave so quickly on a real phone, in patchy networks, so device checks are needed. Context also decays over time (weather changes, new occurrences, etc.) and so what was the best outfit yesterday, is maybe not going to be the best today, and potential testing should consider time (context) dependant scenarios. Long-tails of styles and sizes also easily go unnoticed unless looked at coverage and not just averages. Privacy guidelines restrict how much user data can be logged for training and evaluations, so we build small, courteous tests to help signal signals. Finally, explanations can look nice, but be unfaithful. We still need to

consider whether the stated reasons aligns with the scoring signals, if it doesn't, then your trust will waver even if all the other numbers make you feel good (Hou et al., 2019).

2.12 Chapter Summary

In this chapter, we described a simple and practical plan for assessing Lufyco, a contextualization relevant to actual mobile usage. We measure what users notice: agreeable visual matches that are prominent, reasonable full ensembles for their scenarios, simple to understand labels that now create trust in both filters and explanations, quick responses that seem seamless, and fewer choices to build the confidence. We refer to simple metrics: Accuracy, Precision, Recall, F1, and AUC-ROC for labels; Recall@K, MRR, Precision@K, NDCG for ranking; latency, memory, battery for performance; and SUS, time-to-decision, edit rate, save/share, and return rate for user value. We make fair offline comparisons in parallel conditions and with equal resources, then we validate improvements seen in offline comparisons with an on-device test to ensure that what seems better on paper also performs better in hand. We identified the tough parts as well: no all-in-one dataset, subjective taste, differences between lab and real-life context, changing context, long-tail coverage, privacy limits, and faithfulness to explanation. These are all typical reasons that fashion apps feel slow or unhelpful. Lufyco tackles these by treating speed, context, wardrobe reasoning and honest explanations as first-class goals, by using benchmarks that reflect real decisions a phone user might have to make, and by validating results first in controlled tests and then in day-to-day use.

3 CHAPTER 03: METHODOLOGY

3.1 Chapter Overview

This chapter demonstrates how Lufyco works and the rationale behind the design choices. It converts ideas discussed in Chapters 1 and 2 into a straightforward plan that operates on a phone with a backend. First, we describe the system design: a mobile app, APIs, and ML services for image retrieval and outfit suggestions. Then we describe the end-to-end flow that starts with user actions: uploading a picture or choosing mood and occasion, then ends with the output barrier showing similar items and full outfit and why each recommendation was made. Next we describe system planning and the key elements of the plan: the image retrieval pipeline encoder, vector index, freshness; the AI Stylist - mood, occasion, weather, wardrobe; the digital wardrobe - auto-tagging, storage, controls; and explainability layer - why this messages. Then we describe data pipeline - where to source data, clean it, label it, and split train / valid / test and augmentations. We describe model design and training, regarding baselines, hyperparameters to use, etc., and ablation studies. We describe inference and performance; latency performance, caching, quantization, mobile limits, etc. Next we outline the evaluation plan as described in Chapter 2, so that offline scores and on device tests, tell the same story. We also identify security, privacy and ethics steps, including minimal data, encryption, consent, safety, and bias checks. Finally, we provide a listing of risks and mitigations, including cold start, long tail styles, drift, and graceful fallbacks. At the end, the reader has a stepwise blueprint to build, test, and ship Lufyco's mobile experience.

3.2 Research Methodology (Saunders' Research Onion in Table Form)

3.2.1 Overview

In this section, we outline the overarching research design for Lufyco using Saunders' Research Onion. We illustrate our customized onion (philosophy → approach → methodologies → time horizons → techniques/procedures) in Figure 1, Table 6 indicates the options we made for each layer and the rationale for the options chosen. When we come to discuss our results later, we will reference the layers of the onion in Figure 1 and the summarized decisions in Table 6.

- Philosophy

  The research is located in Pragmatism meaning it is a combination of positivist measurements (e.g., accuracy and latency of the model) with interpretivist

observations (e.g., usability and trust), to ultimately because Lufyco needs both hard metrics and human feedback (Saunders et al., 2019)

- Approach
  We adopted a Deductive + Inductive approach testing hypotheses about speed/relevance whilst also learning from interviews/observations with the purpose of revising features (Saunders et al., 2019)

- Methodological choice
  We chose Mixed Methods appropriate for collecting quantitative performance and qualitative UX evidence in one coherent design (Saunders et al., 2019).

- Strategies
  We conduct experiments (offline model comparison and in-app A/B testing), surveys and semi-structured interviews, an initial case-study/pilot, plus some archival/literature review and observation/analytics to assist in triangulating our findings (Kohavi et al., 2009; Saunders et al., 2019; Kiapour et al., 2015).

- Time horizons
  We triangulate cross-sectional short-term snapshots (first-run usability, offline metrics) with a short time horizon longitudinal window (2-4 weeks) to monitor retention and the impact of changes to context (Saunders et al., 2019).

- Techniques & procedures
  For ML we will use additional datasets + pilot catalog + also user-like photos (Liu et al., 2016); for UX we collect consented in-app data (about mood/occasion, wardrobe uploads) with cryptographic storage and processes for easy deletion (Saunders et al., 2019). Evaluation metrics (Recall@K, NDCG, F1, AUC-ROC, latency, SUS, time-to-decision) are also depicted in Table 3.1 (Kiapour et al., 2015; Liu et al., 2016; Kohavi et al., 2009).

Figure 1: Onion Diagram

Table 6:Research Methodology

Layer	Choice (what we use)	Justification (why we use it)
Research Philosophy	Pragmatism (uses both positivism and interpretivism)	Lufyco needs numbers (model accuracy, speed) and people's views (usability, trust). Pragmatism lets us use both without forcing one viewpoint. We measure hard facts and listen to users. (Saunders et al., 2019)
Research Approach	Deductive + Inductive	Deductive: test clear hypotheses (e.g., "context-aware ranking is faster than no context"). Inductive: learn from interviews/observations what users need and understand. Using both helps us prove what works and discover improvements. (Saunders et al., 2019)
Methodological Choice	Mixed Methods (quantitative + qualitative)	We must show that the app is accurate and fast (quant) and easy and trustworthy (qual). Mixed methods capture both sides, giving a complete picture. (Saunders et al., 2019)
Strategy	Experiments (offline + A/B in-app)	Offline: compare encoders, indexes, thresholds under the same settings. Online: A/B test context-aware vs. baseline in the app. Experiments show clear, fair comparisons. (Kohavi et al., 2009)

Strategy	Surveys / Questionnaires (closed + open questions)	Closed questions give scores (e.g., SUS). Open questions tell us why users liked or disliked something. Short, in-app surveys are easy to run (Saunders et al., 2019).
Strategy	Semi-structured Interviews	One-to-one talks reveal how users read outfits and explanations, what confuses them, and how they decide. Semi-structured = a simple guide + freedom to probe (Saunders et al., 2019).
Strategy	Case Study / Pilot	A small real-world pilot (limited catalog + users) checks the end-to-end flow on real phones and networks, not just in the lab (Saunders et al., 2019).
Strategy	Archival / Literature Review	We learn best practices in fashion retrieval, outfit compatibility, explainable AI, and mobile UX. This sets solid baselines and fair comparisons (Kiapour et al., 2015).
Strategy	Observation & In-App Analytics	With consent, we track time-to-decision, edits, saves/shares, and simple click paths. This shows friction points users may not say out loud (Kohavi et al., 2009).
Time Horizon	Cross-sectional + short Longitudinal	Cross-sectional: snapshot tests for models and first-run UX. Short longitudinal (2–4 weeks): see retention and repeated use, and how weather/context changes affect results. (Saunders et al., 2019)

Techniques / Procedures Data Collection (ML)	Secondary datasets (public fashion image/outfit sets) + pilot catalog + user-like photos	Public data pretrains models. Pilot catalog and user-like photos fine-tune to real mobile photos (messy lighting, poses). This is practical and diverse.(Liu et al., 2016)
Techniques / Procedures Data Collection (App/UX)	Primary, consented app data (inspiration photos, wardrobe uploads, mood/occasion; simple weather)	We only collect what we need for "find similar" and outfit suggestions. Data is opt-in, encrypted, and easy to delete/export. (Saunders et al., 2019)
Requirement Elicitation	Literature review, competitive scan, semi-structured interviews, structured questionnaires	We learn the space (review/scan), then gather needs and language (interviews), then confirm priorities at small scale (questionnaires). (Saunders et al., 2019)
Evaluation Quantitative	Retrieval/Ranking: Recall@K, Precision@K, MRR, NDCG, mAP; Attributes: Accuracy, Precision, Recall, F1, AUC-ROC; Performance: p50/p95 latency, model size, memory/battery	These show: do we find good matches early, order outfits well, label items correctly, and keep the app fast and light? (Kiapour et al., 2015; Liu et al., 2016)
Evaluation Qualitative	SUS, time-to-decision, edit rate, save/share rate, short post-task interviews	These show: does the app actually feel easier, faster, and more trustworthy? Interviews explain the scores. (Saunders et al., 2019; Kohavi et al., 2009)
Data Analysis	Simple statistics (t-test/Mann-Whitney), effect sizes, confidence intervals; Thematic analysis for interviews	Stats make A/B conclusions reliable. Thematic coding finds repeating UX themes that guide design and explanation wording. (Saunders et al., 2019)

Hypotheses (examples)	H1: Context-aware re-ranking reduces time-to-decision. H2: Explanations lower edit rate and raise save/share. H3: ViT+ANN improves Recall@10 vs. keyword search. H4: p95 latency ≤ 1s raises SUS and task success. (Johnson et al., 2017	Each hypothesis maps directly to a user benefit (speed, trust, relevance, smooth feel) and is testable with our metrics.
Sampling & Participants	Purposive (interviews: varied styles/sizes/occasions); convenience (pilot users); minimum n per A/B arm from a quick power check (Saunders et al., 2019)	Ensures we cover common and edge cases, keep the study practical, and reduce the chance of false results.
Ethics, Privacy, Safety	Consent, data minimization, encryption (in transit/at rest), easy opt-out, bias checks (Saunders et al., 2019)	Wardrobe photos and preferences are sensitive. We collect the least we need, secure it, allow deletion/export, and check rankings for unfair bias.
Quality & Validity	Triangulation, pilot runs, pre-registered metrics (Saunders et al., 2019; Kohavi et al., 2009)	Multiple methods pointing to the same answer increase confidence. Pilots de-risk. Pre-declared metrics avoid "moving the goalposts."

3.3 Development Methodology

Lufyco is created in short, repeatable cycles. The process is simple: we plan a small change, design it, build it, test it on real phones, collect results, and improve it. We combine lean UX (quick sketches → quick tests) with agile sprints (1–2 week sprints) and a light ML loop (collect/clean the data → train → evaluate → deploy). Every cycle ends with a working build, a small usability check, and model numbers that are aligned with the benchmarks from Chapter 2. This principle keeps design, engineering and evaluation close together, culminating in every definition becoming a testable feature or metric (Schwaber & Sutherland, 2020).

3.4 Requirement Elicitation Methodology

3.4.1 Interviews

We conducted straightforward semi-structured interviews with a diverse assortment of shoppers whose ages, sizes, styles, and climates varied. We asked them to reflect on how they search when they don't know fashion terminology, how they select clothing for an event, how "mood" and "occasion" resonated with them, and for their impression of a digital wardrobe. We also inquired about privacy and trust: what data would they be okay sharing, and what would lead them to feel comfortable with any explanations offered. We recorded with permission, then transcribed and summarized the conversations. We categorized similar sentiments into themes. Those themes contributed to user personas, pain points, and concrete user stories (Saunders et al, 2019) (for example, some kind of automatic indication of rain-friendly shoes).

3.4.2 Surveys

We used short, in-app surveys and longer online surveys. Closed questions (such as 1-5 ratings) produced numbers for satisfaction with image search, suggestions for outfits, and explanations. Open-ended questions allowed users to write their own language around their style and needs. These findings helped us to prioritize features and set sensible targets (Saunders et al., 2019; Kohavi et al., 2009) (like time-to-decision), while also having a consistent measuring system across versions of the product by ask those user enjoy the same questions across all the variants.

3.4.3 Document Analysis

We examined competitor apps, app store reviews, help materials, and other helpful articles. This highlighted for us what is common (searching, filters, “complete the look”) and what people complain about (weak personalization, slow outputs, and confusing sizing). We noted areas where they misses the mark for a mobile-only app, such as shaky image searching with personal photos, and no suggestions that considered weather conditions. This led us to characterize early functional requirements (e.g., upload a photo to find similar) and non-functional requirements (Saunders et al., 2019) (quick responses, explicit privacy, short explanations).

3.4.4 Brainstorming

We did quick workshops with the team and some users. We used simple methods (How-Might-We prompts, quick sketching, dot-voting). We discussed possible options for the AI Stylist, mood/occasion inputs, and explaining styles. Weaimed to connect to user pains and docs metrics from Chapter 2 (when possible), and then we used MoSCoW (Must/Should/Could/Won't) just to select what to build first. When we experienced trade-offs (more control vs. easier screen), we planned small A/B tests so data would guide us (Kohavi et al., 2009; Schwaber & Sutherland, 2020).

3.4.5 Observation

We observed people completing basic tasks on paper prototypes and early clickable demos. Tasks included: upload a photo, find similar items, set mood and occasion, and accept or edit an outfit. We timed each step and noted areas where users hesitated, as well as errors or obstacles. This "watch and listen" step uncovered practical usability issues with the prototypes unclear icons/design, long filter panels, and confusing wording that may not have surfaced in an interview only context. We came out of each observation with a list of small UI or copy fixes for the next sprint (Saunders et al., 2019).

3.4.6 Prototyping

We progressed from quick sketches, to low-fidelity wireframe and finally to high-fidelity interactive prototypes. At the beginning, we were testing layout and wording with lightweight "fake AI" before we had the models ready for usability testing. Later we connected to actual models to measure response time to mockup interactions, as well as ranking on device (Schwaber & Sutherland, 2020). Each round of prototypes ended with a quick usability check we always documented. Changes for usability fixes resulted from each round of prototypes, which all fed directly into the next sprint to ensure continuous application improvement.

3.4.7 Self-Evaluation

The team used Lufyco every day, on lots of their own photos and wardrobes. We kept a pretty simple shared log, with a date, scenario, problem, and suggested fix. Every week we reviewed the log together and turned notes into requirements or bug tickets. This "use it ourselves" step

kept quality high between formal research studies and ensured it stayed practical for everyday use (Schwaber & Sutherland, 2020).

Together, these methods provided us with a complete, cross-checked set of requirements. Interviews and observation exposed real needs. Surveys described the size of those needs. Document analysis established reasonable baselines. Brainstorming provided options based on insights. Prototyping and self-evaluation tested what worked on real phones. Each source of input mapped to a feature, a metric, or a simple rule we could test, so that Lufyco maintained a strong focus on solving real user problems.

3.5 Design Methodology

Method: Our central methodology is OOADM (Object-Oriented Analysis and Design) with a light touch of SSADM (Structured Systems Analysis and Design) where it helps clarify data flows and data storage (Saunders et al., 2019).

Why OOADM: Lufyco as a mobile app is composed of clear things (users, things in your wardrobe, outfits, context, image searches, explanations, etc). OOADM captures this nicely. It allows us to define what each thing does, how things communicate with each other, and how screens and APIs fit together.

Where SSADM Helps: For parts that pass data (image → embedding → vector search → results) and for data storage (collections/tables and their connections), the use of SSADM-style data flow diagrams and simple ER/collection sketches makes all of this easy to visually digest quickly.

3.5.1 OO Analysis - What the App Must Do

The application is designed to assist users in an intelligent and intuitive way, as they curate their fashion and style journey. Its primary use cases allow users to easily sign up, log in, and manage their profile. Users can upload a photo to find visually similar items with the help of image search capabilities powered by artificial intelligence (AI). A digital wardrobe lets users add clothing photos, edit or delete them, and they all have auto-generated tags on attributes identified during the photo uploads. The "Plan My Look" feature allows users to set a mood or occasion, in conjunction with automatically detected weather conditions, to enable them to plan

for relevant wardrobe items. Furthermore, users can essentially suss through several of the suggested outfits, swap items, and play and adjust the colors of the outfits as they see fit. From there, users can save the style cards they create or share with others their favorites, and all their favored outfits can go on a wishlist, too. The catalog can also be browsed and filtered by category, size, color, or season.

The key domain objects include either a User (with preferences, privacy settings, a wardrobe, and style card), a WardrobeItem (an owned clothing item that has an image and tags about it), a CatalogItem (an item in a store with associated attributes), an ImageQuery and SearchResult (for all image-query based searches), a Context (mood, occasion, weather, and time), an Outfit (a set of items, and explanations), and a Recommendation (ranked outfit lists for a given context).

3.5.2 OO Design - How the App Will Do It

The application follows a mobile app structure utilizing React Native. The front end comprises Views/Screens displaying the UI, ViewModels which manage state and actions, and Models representing the core domain objects. The backend service layer specifies clear interfaces for various services. The ImageEmbeddingService translates the uploaded images into a vector representation, then the VectorSearchService does a very fast nearest-neighbor search. The ContextService is responsible for mood and occasion parsing with weather integration. The OutfitService builds and ranks an entire outfit. The ExplanationService generates a short but candid explanation for each outfit suggestion. Supporting services such as WardrobeService, CatalogService, and AuthService serve basic CRUD and authentication services. In terms of design patterns, this architecture will ensure the scalability and maintenance of the application. The Strategy pattern provides an easy way to add more flexible outfit scoring approaches (rule-based, learned, or hybrid) to the overall architecture. The Adapter pattern is applied as a wrapper around external APIs, as in weather services. The Repository pattern is in place to handle clean access to data, while Facade provides a common entry point to the machine learning services. Observer/Event updates the vector index when new items are added, and Cache saves earlier searches and popular contexts. Key flows are Find Similar, in which the app uploads an image, the image is converted to an embedding, ANN search performs, results are re-ranked if needed, and back comes short explanation text; Plan My Look, in which the user defines a mood and occasion, checkes the weather, generates outfits, and presents the

explanations; and Save Style Card, in which favorite outfit combinations are saved and/or shared. (Johnson et al., 2017; Google Research, 2020–2024; He & Hu, 2018; Wang et al., 2019; Hou et al., 2019)

3.5.3 Data & Storage (SSADM-Style Clarity)

Data is organized in simple collections and tables. The users collection contains user profile information, preference settings, and privacy settings. **wardrobe_items** contains items of clothing owned by the user, including a reference image and automatically generated tags. **catalog_items** are equivalent to store products and hold relevant attributes, as well as availability status. outfits is a table of combined outfits that are generated by the app as well as context and explanation. **style_cards** hold user saved looks and embeddings are used for various vector references for fast retrieval. **event_logs** records user interactions with the app as well as app performance metrics such as time to decision and saves.
The primary data flow occurs through two pipelines: for image search, the flow is as follows—Image → Normalize → Embed → Index/Search → Re-rank → Return. And for look planning, the flow is as follows—Plan → Build Context → Generate Outfits → Explain → Return. This is the streamlined flow from user input to allowable meaningful visual output. (Liu et al., 2016; Ge et al., 2019; Johnson et al., 2017)

3.5.4 API Design (Clean and Small)

The backend of the app offers light-weight and privacy-conscious APIs.

- POST /image-search will upload a picture and return the top similar items, with short explanations. (Johnson et al., 2017; Hou et al., 2019)
- POST /plan-look will take the mood and occasion, add the weather context automatically, and return outfits recommended with alternatives and explanations.
- POST /wardrobe takes images of wardrobe items, auto-tags them, and saves them to the user account.
- GET /itemsfilters - a user can browse the catalog and filter by parameters (color, size, season...).

- POST /style-cards will save or share selected style combinations.

All responses return only the data that is appropriate to response while maintaining privacy and also that explanations represent the real logic of decisions used in the system.

3.5.5 Non-Functional Design (Built-in)

The design includes performance and user trust. The goal is to achieve p95 latency below one second for visible results. The vector index permits micro-updating, so that new items from a catalog will appear without delay. Privacy is ensured, but limiting data is critical, encryption must be used, and users must control delete/export requests. If embedding, weather, or something fails, the product will gracefully default to a simpler search mode (e.g. attribute-only or context-free outfits). The ML models are small and quantized to minimize computation time, and caching is used to limit redundant network requests. (Johnson et al., 2017; Google Research, 2020–2024)

3.5.6 Traceability and Validation

A clear requirements-to-design matrix will ensure that each feature has an identifiable line back to the user needs and testable outcomes. You will build prototypes first with early wireframes and stub APIs so that you have something to validate flow and wording before working through the full implementation. Sprint demonstrations will occur every one to two weeks. During these demonstrations, usable slices of the app's functionality will be shown. This iterative process will allow you to identify usability or performance problems in the app earlier and ensure the app reflects user needs and project requirements. (Schwaber & Sutherland, 2020; Saunders et al., 2019)

3.6 Programming Paradigm

Lufyco employs a variety of programming paradigms so that each component is written in the style that makes the most sense. The primary paradigm is Object-Oriented Programming (OOP) for the mobile app and backend services, because the system inherently includes "things" such

as Users, WardrobeItems, Outfits, Context, and Recommendations. We also employ a Structured/Procedural programming style for very clear, step-by-step scripts (data prep, index refreshes, migrations). Further, we use a Functional style for data transforms, ranking/re-ranking, and React UI state updates, because pure, composable functions are much easier to test and reason about. The combination of all three makes for a simple, fast, and maintainable codebase. (Schwaber & Sutherland, 2020).

3.6.1 Object-Oriented Programming (OOP) - primary paradigm

OOP is applied in mobile domain models and view models, backend service classes, and ML model wrappers. It facilitates how we organized Lufyco into modular, reusable components that resemble real-world entities, e.g., users, wardrobe items, and outfits. The modular design simplifies testing and the ability to replace or expand/testing without affecting one-shoe other. The small, well-defined interfaces are also helpful as they facilitate collaboration and teamwork as well as maintenance of the code. Some common design patterns we've use include the Strategy pattern for different methods of scoring outfits (rule-based versus learned versus hybrid), Adapter to connect to external weather providers, DAO/Repository to help with data access, as well as Facade to act as a simple entry to a complex ML service. (Schwaber & Sutherland, 2020)

3.6.2 Structured/ procedural programming - scripts and pipelines.

Procedural programming is appropriate for clearly structured tasks that use stepwise flows, such as data preparation, index management, and maintenance utilities. (Johnson et al., 2017) The scripts will follow a simple procedure: read in input files or data, validate and clean that data, then transform it in some way such as by resizing or augmenting, and finally save and log the result. This adds clarity and trustworthiness for any processing task that in batch mode needs to run on a schedule. Thus, scripts are great for nightly vector index updates or other maintenance tasks that should be predictable and easy to rerun in necessary.

3.6.3 Functional Programming - transforms, ranking and UI states

Functional programming is a technique used in Lufyco, especially when dealing with pure, side-effect free computations. It is mostly found in feature extraction, similarity scoring, context-dependent ranking, and React UI state updates. The functional programming style is mainly about using pure functions (inputs always yield the same result) to enhance

predictability and testability. Using operators like map, filter, and reduce allows one to build complex logic out of small, composable functions. In React, using functional components and hooks (useMemo, useCallback) leads to clean, reactive UI while minimizing bugs and maximizing consistent results. (Kohavi et al., 2009)

3.6.4 Programming Languages and Fit

Lufyco employs TypeScript and Python because each language provides good support for all three paradigms. TypeScript is used primarily in mobile and backend development, providing fairly thorough OOP support through interface and class support, as well as functional utility methods for managing UI logic or data lists. Python is used for ML and data pipelines, allowing procedural scripting and functional transformations using libraries like NumPy and Pandas. Their combination creates a good balance between structure and readability, and performance across all parts of the system. (Johnson et al., 2017)

3.6.5 Summary checklist

Lufyco utilizes OOP for its primary app architecture, and backend services, for the sake of clear responsibilities and testability. The team favors procedural programming for batch jobs and data pipelines to keep it simple and reliable. Functional programming was used for handling transformations, ranking logic, and UI state to ensure predictability and composability. By applying the right paradigm for each respective task, Lufyco remains simple to comprehend, quick to execute, and extensible as the system grows.

3.7 Testing Methodology

3.7.1 Model testing

Prior to deploying these models in the application, each machine learning model is validated independently using data sets (train/validation/test splits) that have been rigorously prepared or when working with small data sets, using k-fold cross-validation. The evaluation criteria will be task-specific and are as follows: Recall@K, Precision@K, MRR, NDCG, mAP (for image retrieval), Accuracy, Precision, Recall, F1, AUC-ROC (for tagging), NDCG, Diversity, and Coverage (VUS) for outfit ranking. We are also evaluating models beyond accuracy to include performance calibration, explanation fidelity, performance under noisy conditions (blur, crop, light), and fairness across conditions. Metrics related to the system would include Inference time, model model size and memory. In addition, ablation studies will help determine which components are most important,

hyperparameters are only tuned on validation data with early stopping, and the final models will be reported on the held-out test set with reproducible settings (Kiapour et al., 2015; Liu et al., 2016; Ge et al., 2019; Gao et al., 2020).

3.7.2 Prototype Testing

After verifying standalone models, we run end-to-end tests of the complete Lufyco prototype on real phones for valuable user journeys like photo upload, look planning, wardrobe edits, and style card sharing. Integration and regression already complete and we will verify shared communication between the app, APIs, and ML services. We measure startup time and latency, as well as memory and battery utilization levels on a range of devices and networks. We will ensure the app functions with resiliency through poor connectivity and gracefully fallbacks occur. Usability tests track task success, timings, and SUS scores, while smaller A/B tests understand ranking and explanation features. Accessibility, privacy, and security are covered, as is QA that combines manual, automated, and smoke testing. During pilot releases we monitor performance, push out updates or fixes safely, and ultimately approval of pilot releases will require completion of an acceptance checklist regarding quality, usability, latency, and privacy. Feedback from the pilot will enhance the next product build sprint (Kohavi et al., 2009; Schwaber & Sutherland, 2020).

3.7.3 Unit Testing

We add small and fast tests for isolated pieces of code to catch bugs early. For the app, we unit test our React Native components and view models (props/state updates, error states). For services, we unit test functions such as input validators, color/size normalization, and explanation templates. For ML code, we test data loaders, preprocessing steps, and utility functions (e.g., cosine similarity, PQ encode/decode) with tiny fixtures. Finally, we mock external calls (storage, weather API) to ensure tests are deterministic. We establish a reasonable coverage target and run these tests on every commit in CI to avoid regressions.

3.7.4 System Integration Testing

In this step, we validate the components acting together in a cohesive way. We start the back end service, the vector index, and the model servers in a test environment, and execute a series of workflow tests such as: Upload Photo → Embed → ANN search → Re-rank → Results with explanation and Plan My Look → Context build (mood/occasion/weather) → Outfit generation → Save style card. We perform contract tests for API schemas, error codes, and paginators; simulate third-party outages (weather API) to verify graceful fallback solutions; and check for other non-functional checks such as p95 latencies, payload sizes, and freshness on the index after an update to the catalog. These tests are executed nightly and prior to releases.

3.7.5 User Acceptance Testing (UAT)

Prior to sign-off, we conduct a UAT with a small group of target users who perform scripted tasks that mirror real-world goals, such as: finding similar items from a selfie, planning an outfit for a rainy evening, or editing and saving a style card, or adding items to the digital wardrobe. Each task has clear acceptance criteria that we track. We document not only whether each task was successful, but also the time it took for users to make their decisions, SUS, and some light comments. Critical issues are then fixed and re-tested prior to release. The UAT verifies that the aspects that passed in the lab also work for at least some people using real phone feeds in the real world.

3.8 Solution Methodology

3.8.1 Data collection process

We leverage three sources for images: public fashion datasets (e.g., DeepFashion/Polyvore) for variation only, small pilot catalog (for actual product images and attributes), and small, consented set of user-like photos (selfies, screenshots, images of wardrobes). We also collect simple context (mood, occasion), local weather using the API (Liu et al., 2016; Ge et al., 2019; Kiapour et al., 2015), and timestamps. In effect we provide the system with a mix of clean studio cases, and messy real images.

3.8.2 Data Processing

We fix image rotation, standardize the size, normalize pixels, and apply simple light augmentations (e.g. crop, flip, blur, brightness / contrast) to increase robustness. We remove near-duplicates to lessen leakage. For text / attributes, we standardize color names, categories, fabrics, and patterns, and then create simple tokens or small embeddings (Ge et al., 2019; Gao et al., 2020). Clean, consistent inputs help the models learn the right signals.

3.8.3 Selecting and engineering features

The primary features are visual embeddings generated from a CNN/ViT. We add attribute features (category, color, fabric, pattern, season), context features (mood, occasion, weather, time horizon), wardrobe features (owned items, palettes, gaps), and query features (base similarity, attribute match) to these visual embeddings. This allows us to find similar items, build complete outfits, and provide explanations for the choices.

3.8.4 Modeling selection

For visual search, we fine-tune a ResNet/ViT with contrastive/triplet loss and serve the results via a FAISS/ScaNN index for quick top-K results. For auto-tagging, we use a smaller CNN/ViT-tiny multi-label model. For outfits, we check a two-stage approach where first simple rules are used to build valid candidates and then a lightweight re-ranker (LambdaMART/XGBoost or small MLP) is used to combine visual, context, and wardrobe signals for ranking. For explanations, we rely on templates that are filled with the same signals used for ranking.

3.8.5 Training the Model

We split the data into train/val/test (stratified), keeping look-alikes together in a split, and applying the appropriate losses (contrastive for retrieval, cross-entropy/focal for tags, pairwise/listwise for ranking). We only tune on valid ation, use early stopping, and track seeds/configs for reproducibility. Mixed precision (Liu et al., 2016; Ge et al., 2019) and checkpointing ensure training remains efficient.

3.8.6 Testing the Model

We evaluate retrieval using Recall@K, Precision@K, NDCG, mAP, and time-to-first-result; tagging with Accuracy, Precision, Recall, F1, AUC-ROC; and outfit ranking using NDCG, diversity, and coverage. We also consider calibration (how well confidence maps to reality), explanation fidelity, robustness (blurry photos/crop/lighting), fairness (simple slices), and performance (latency p50/p95, size, memory).

3.8.7 Feedback loop

With your permission, we will log your clicks, saves, swaps, hides, and time-to-decision. Weekly we refresh data and make small, low-risk updates. We limit popularity effects, control for bias, and human spot-check (Kohavi et al., 2009) explanations and edge cases. This ensures the system is able to improve while keeping the overall system stable and fair.

3.9 Project Management Methodology

3.9.1 Approach

We employ an Agile/Scrum methodology that allows us to build, test and iterate Lufyco in small, clear steps of progress. We plan work in quick two-week sprints. Every sprint begins with a manageable goal (piece of functionality), ends with a working demo, and includes one review to determine the next piece of functionality to improve. We maintain a single, prioritized backlog, and update it as we learn. Design, APIs, and docs are kept light and versioned for safety and tracking.

3.9.2 Team roles

A product lead/research lead or owns the scope, priorities, and acceptance criteria for the functionality. Mobile developers create the React Native app. Backend/ML engineers create the APIs, embeddings, vector search, and recommender. A designer/UX lead develops flows, wireframes, and usability assessments. QA plans tests, runs regression tests, and verifies behavior on real devices (multiple people can share this role in a small team).

3.9.3 Rhythm of work

At the beginning of every sprint, we plan the work. We check-in for a short daily meeting to remove any blockers. Every week we do a quick sync for any risks. At the end of the sprint, we run a review to demo progress and a retrospective to agree on improvements. "Done" means code reviewed, tests pass, performance targets are met, privacy checks completed, and basic documentation updated.

3.9.4 Tools

We use Jira/Trello/Notion for planning, Figma for design, and GitHub/GitLab with CI for code and builds. We guard quality with unit, integration, and UI tests, plus linting and static analysis. We also watch crash reports and simple dashboards and monitors for latency and error rates so we can react quickly.

3.9.5 Risk and change control

We also keep a visible risk log (for example: model latency, cold start, data quality) with an owner and mitigation for every risk. Any change which affects scope, timeline, or quality goes through a short change request: what is changing, why, impact, and final decision. This helps us all stay consistent and transparent.

3.10 Scope

3.10.1 In-scope

We will develop the Lufyco mobile app (Android/iOS) that can be accessed by an end-user, developed in React Native. The app will provide account management and user profile with minimal preferences and privacy controls. Users can either upload an image or take a photo to initiate the image-based discovery, which recognizes visually similar items by employing deep visual embeddings and a fast vector search. Users will be able to build a digital wardrobe by uploading a photo of their own clothing items. The items will be auto-tagged (category, color, season, fabric) but can be modified or deleted. The context-aware AI Stylist will allow the user to set a mood and occasion and will autonomously detect the weather; it will also suggest full outfits that the user will be able to edit, using the items in the wardrobe first, and fill in any

gaps from a small pilot catalog. Each suggestion will come with a brief explanation of why it was an acceptable suggestion (e.g. fit the color palette, weather, fits with the wardrobe). The application has simple browsing and filtering functions, a wishlist, and saving or sharing style cards; on the server side, we have clean APIs for the embedding service, vector index, outfit generator, and explanation, all with minimal and secure payloads. We intend to have low latency (about a second to the first useful result), encrypt data in transit and at rest, provide opt-in telemetry, easy data delete/export, and provide basic accessibility (font scaling, color contrast, TalkBack/VoiceOver). We will sample offline on models (Recall@K, NDCG, F1), validate on real devices (latency, memory, battery), and run a small pilot to solicit feedback. Liu et al., 2016; Ge et al., 2019; Johnson et al., 2017; Google Research, 2020–2024; Hou et al., 2019).

3.10.2 Out-scope

We will not create dashboards for merchants and shops for inventory management, pricing, promotions, or analytics. We will not perform the full suite of marketplace operations such as multi-seller onboarding, payouts, or dispute resolution. Production payments are not included. We will not deal with shipping or returns either. A website version is excluded from this phase; we are only providing work on the mobile application. Additionally, AR/VR try-on, 3D avatars/body scan sizing are excluded from this phase. A "global" rollout with complex cross-border compliance, or a 24/7 production service level agreement are also excluded from this phase. Finally, we will not integrate large 3rd party personalization datasets, nor will we develop advanced content moderation pipelines beyond basic image plausibility checks. Some of these phases may be illustrated with simple stubs, or mocked flows, for usability testing, but they will not be delivered as live product features in this phase. (project scoping best practice; Saunders et al., 2019).

3.11 Schedule

3.11.1 Gantt Chart (key milestones and phases)

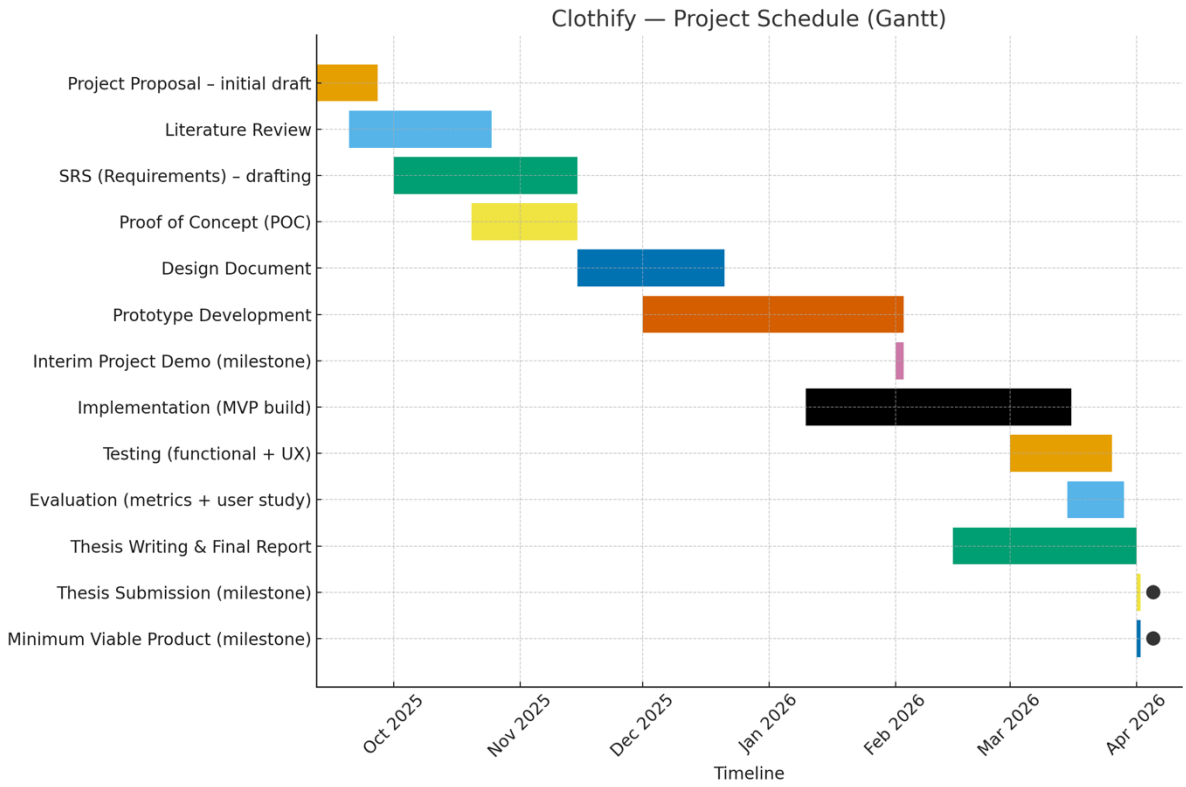


Figure 2 : Gantt Chart

3.11.2 Deliverables (milestones & hand-ins)

Table 7:Deliverables

Deliverable	Planned Date
Project Proposal – initial draft	26 Sep 2025
Literature Review – complete set of sources	24 Oct 2025
SRS (Software Requirements Specification) – final	14 Nov 2025
Project Proposal & SRS – final pack	14 Nov 2025
Proof of Concept (image search working on small dataset)	14 Nov 2025
Design Document (system + data + model design)	20 Dec 2025
Prototype (end-to-end mobile flow)	02 Feb 2026
Interim Project Demo	02 Feb 2026
Implementation (MVP build) complete	15 Mar 2026

Testing (device + usability + model metrics)	25 Mar 2026
Evaluation (results + discussion)	28 Mar 2026
Thesis / Final Report	31 Mar 2026
Thesis Submission (final)	01 Apr 2026
Minimum Viable Product tag	01 Apr 2026

3.12 Resource Requirements

3.12.1 Hardware Requirements

Table 8: Hardware Requirements

Requirement	Justification
Laptop with Intel i7/Ryzen 7 (or better)	Enough compute for local training, builds, and emulator runs.
16–32 GB RAM	Smooth model training, indexing, and multi-tool workflows.
512 GB+ SSD	Store datasets, embeddings, checkpoints, and builds.
Optional NVIDIA GPU (e.g., RTX 3060/T4)	Faster fine-tuning of vision models and re-ranking experiments.
Cloud VM (4–8 vCPU, 16–32 GB RAM) + object storage	Host inference services, vector indices, and images.
Test devices: 1 low-end + 1 mid-range + 1 high-end Android/iOS	Real-world performance, battery, and UX checks on phones.
Stable broadband connection	Dataset pulls, model pushes, continuous integration.

3.12.2 Software Requirements

Table 9: Software Requirements

Software	Usage
React Native (Expo optional)	Mobile app for Android/iOS.
Python (FastAPI/Flask) or Node.js (Express/Nest)	Backend APIs for embeddings, search, outfits, and explanations.

PyTorch / TensorFlow	Train/fine-tune visual encoder, auto-tagger, and outfit re-ranker.
FAISS / ScaNN	Fast vector search for image-based “find similar”.
MongoDB / PostgreSQL	Profiles, wardrobe items, and configuration data.
Redis	Caching for low-latency responses.
ONNX / TorchScript	Efficient model serving.
Firebase Auth and S3/Firebase Storage	Secure sign-in; store images and indices.
Figma	UI flows, components, and hand-off.
VS Code + GitHub/GitLab + CI	Dev environment, version control, builds, and tests.
Postman/Insomnia	API testing.
Docker	Reproducible environments for serving.
Google Colab/Notebooks	Quick experiments and EDA.
Zotero/Notion	Reference and note management.

3.12.3 Data Requirements

Table 10: Data Requirements

Data	Source / Notes
Public fashion image datasets	DeepFashion / DeepFashion2, Polyvore Outfits, Street2Shop—broad coverage of categories, poses, lighting.
Pilot product catalog	Small real or simulated catalog with titles, images, category/color/fabric/season.
Consented user-like images	Mirror selfies, screenshots, and wardrobe photos captured on phones to reflect real noise.
Context signals	Weather API (city/date), user-provided mood/occasion, and timestamps for time-aware styling.
Annotation & taxonomy	Standardized color vocabulary, category tree, and pattern/fabric lists for clean labels.

3.12.4 Skill Requirements

Table 11: Skill Requirements

Skill	Why it is needed
Mobile engineering (React Native)	Build a stable, smooth app with native-like performance.
Backend & API design	Secure, low-latency endpoints and data pipelines.
Computer vision & recommender systems	Train image embeddings, auto-tagger, and outfit re-ranking.
Data engineering & MLOps	Preprocessing, indexing, serving, monitoring, and reproducibility.
UX/design & accessibility	Clear flows, explainable outputs, and inclusive UI.
Testing (unit/integration/device)	Reliability on real phones and networks.
Privacy & security basics	Safe data handling, encryption, and user controls.
Academic writing & presentation	Reports, thesis, and demos.

3.13 Risks and Mitigation

Table 12:Risks and Mitigation

Risk	Impact	Likelihood	Exposure	Mitigation
Insufficient diverse images for training	4	3	12	Add more public datasets; collect consented user-like photos; strong augmentations.
Inconsistent catalog labels	3	4	12	Enforce a shared taxonomy; mapping rules; auto-tagger to fill gaps.
Visual search accuracy too low on user photos	5	3	15	Fine-tune encoder with contrastive loss; hard negatives; evaluate with real mobile photos.
Outfit recommendation latency high	5	3	15	Two-stage (rules + re-rank), FAISS tuning (IVF/HNSW), caching, quantization.

Mobile battery/performance issues	4	3	12	Compress images, memoize calls, background prefetch, test on low-end devices.
Overfitting to studio images	4	3	12	More user-like data; augmentations (blur/crops/lighting); early stopping.
Fairness/bias across sizes/tones/styles	4	2	8	Slice metrics; rebalance; human review; adjust ranking weights.
Privacy/security non-compliance	5	1	5	PII minimization; encryption in transit/at rest; access controls; opt-in and easy delete/export.
External API outage (weather)	3	3	9	Cached last forecast; retry with backoff; fallback to wardrobe-only planning.
Schedule slippage	3	3	9	Smaller sprints; clear acceptance criteria; early demos; buffer before deadlines.
Tooling/CI breakages	2	3	6	Pin versions; Docker images; smoke tests on each build.

3.14 Chapter Summary

This section is complete with deadlines for each deliverable by Lufyco, what resources will be necessary, and what risks of slowing progress exist. The timeline leverages the deliverables of proposal, SRS, proof-of-concept, prototype, demo, and testing, evaluation and delivery. It is also based upon a practical mobile-and-ML stack, including React Native on the client-side, a python /node API, PyTorch or TensorFlow using FAISS for search, and cloud storage for the images and the index. Data needs for the project include public datasets for fashion, a small pilot catalog, consents for user-like photos, along with a simple contextual layer (mood,

occasion, weather). The risk table identifies the biggest threats, including accuracy, latency, data quality, privacy and mitigations that can be taken. With these plans and contingencies in place, the project is ready to confidently move forward into build, testing, and evaluation.

Chapter 04: Software Requirement Specification (SRS)

4.1 Chapter Overview

The Software Requirement Specification (SRS) chapter contains a comprehensive examination of the requirements for the system, including how they were elicited, the stakeholders, their perspectives, and what the functional and non-functional system requirements are. It includes diagrams such as the Rich Picture Diagram, Stakeholder Onion Model, Context Diagram, and Use Case Diagram, for the purpose of understanding the context of the system and how the system will interact with the different entities in the context. The MoSCoW principle is used to prioritize the requirements, to focus on what is essential for the system's success.

4.2 Rich Picture Diagram

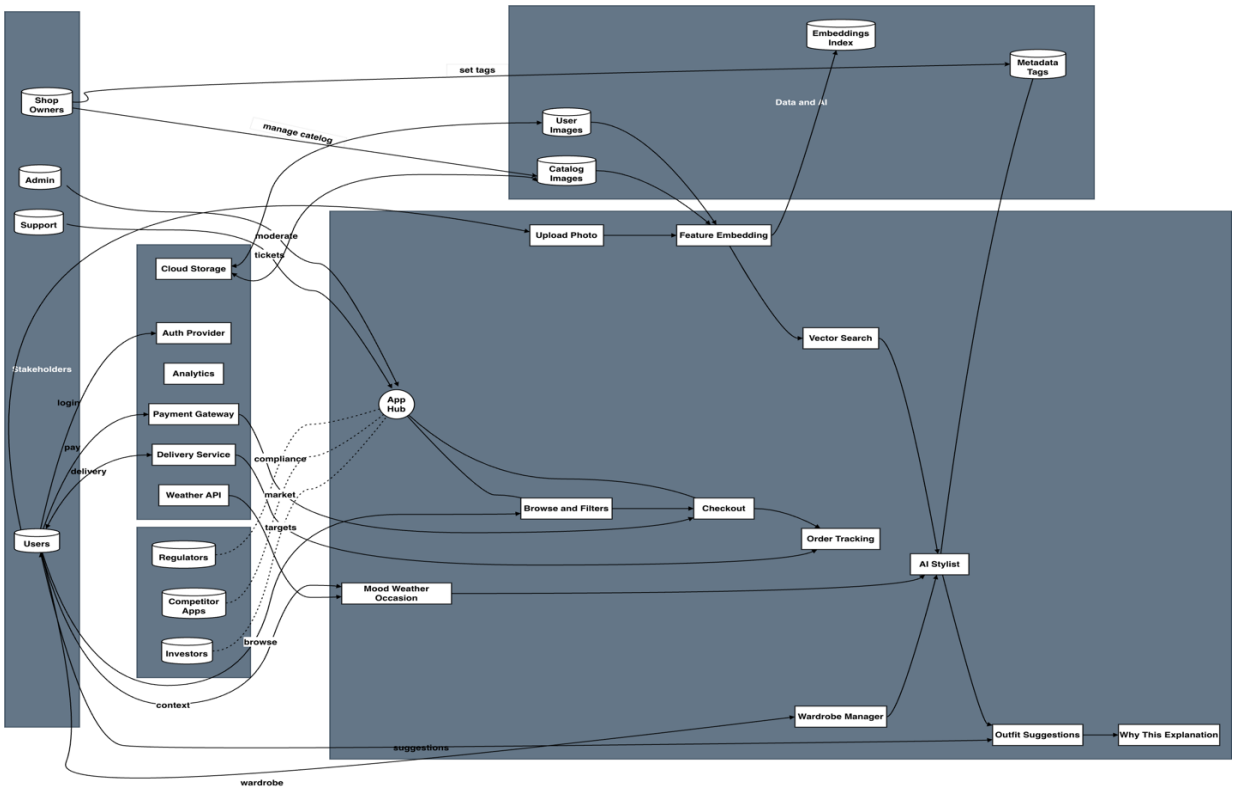


Figure 3: Rich Diagram

(https://drive.google.com/file/d/1ZH7NN1KO94Kzb8QYNzWNkjt1yxmORNjt/view?usp=drive_link)

4.3 Stakeholder Analysis

Lufyco's customer base includes fashion shoppers and small retailers, benefiting from the combination of image-based search, an AI Stylist, and a digital wardrobe. Participants include direct users (shoppers, sellers), and enabling participants (payments, cloud, logistics, compliance). When mapped out, it clarifies who is responsible for influencing requirements, operating the system, or controlling it.

4.3.1 Stakeholder Onion Model

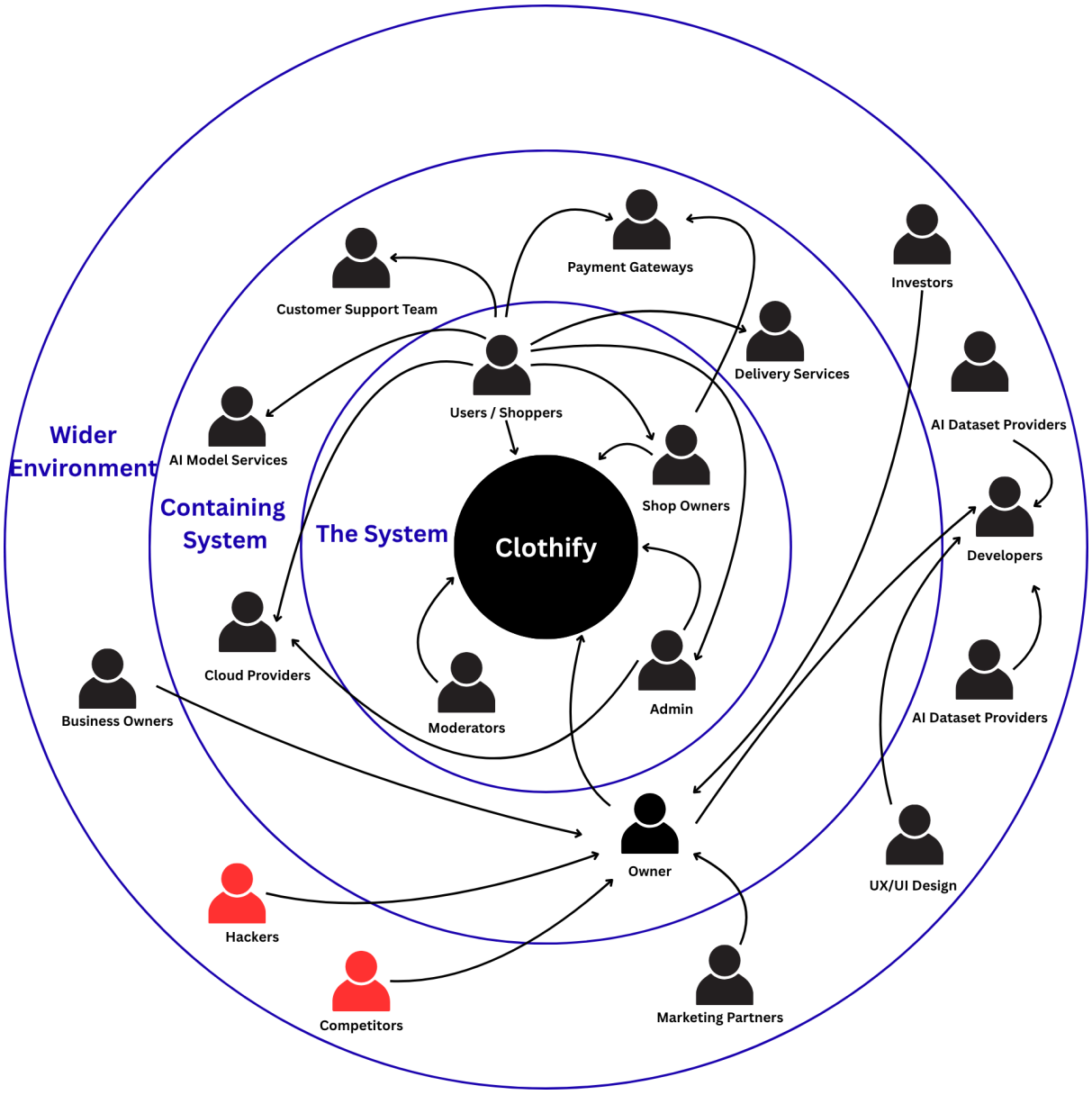


Figure 4:Stakeholder Onion Model

4.3.2 Stakeholder Viewpoints

Table 13: Stakeholder Viewpoints

Stakeholder	Role in/around system	How they interact with Lufyco	Primary needs / benefits
Users / Shoppers	Primary end-users	Upload photos, search/browse catalog, manage digital wardrobe, use AI Stylist, purchase, review	Relevant results, clear explanations ("why this"), fast & secure checkout, privacy controls, order tracking
Shop Owners	Supply-side partners	Onboard products, manage inventory/prices, view demand signals, fulfill orders/returns	Easy listing tools, accurate tagging, exposure via recommendations, sales analytics
Admin	System administration & policy	Configure catalogs, manage users/roles, monitor health/abuse, handle takedowns	Audit trails, dashboards, RBAC, policy enforcement, SLA monitoring
Moderators	Trust & safety	Review flags, handle IP/privacy complaints, curate or remove inappropriate items	Efficient review queue, evidence logs, escalation workflows
Customer Support Team	Help & issue resolution	Answer tickets/chats about orders, payments, returns, app issues	Access to customer/order history, SLAs, canned solutions, refund/replace tools
Owner / Product Lead	Business & product direction	Prioritize roadmap, set KPIs, approve releases and experiments	Reliable metrics (SUS, Recall@K,

			conversion), risk reporting, A/B results
Developers	Delivery team	Build mobile app, APIs, ML services, CI/CD	Clear requirements, test/stage envs, logs/observability, performance budgets
UX/UI Designers	Experience design	Research, prototypes, usability tests, accessibility reviews	User analytics, test panels, design system, accessibility tooling
Marketing Partners	Growth & campaigns	Run promos, creatives, influencer tie-ins, attribution	Campaign hooks, deep links, promo codes, privacy-safe analytics
Payment Gateways	External integration	Tokenize cards, authorize/capture/refund, dispute handling	Stable API contracts, idempotency, PCI compliance, low latency
Delivery Services	Fulfillment partners	Rate quotes, label generation, pickup & tracking webhooks	Accurate order data, address validation, status callbacks
Cloud Providers	Hosting & storage	Compute, storage (images/embeddings), CDN, monitoring	Predictable workloads, security baselines, cost controls
AI Model Services	ML infra & tooling	Embedding generation, re-ranking, model registry/serving	Clean data contracts, versioning, latency/throughput targets
AI Dataset Providers	Data sources/licensing	Provide licensed fashion images/attributes for training/fine-tuning	Proper licensing, usage reports, data security, acknowledgements

Investors	Governance & funding	Review milestones, growth, unit economics	Transparent KPIs, risk log, compliance posture
Business Owners (Brands)	Potential partners	Provide official catalogs/feeds, co-marketing	Brand safety, correct attribution, sales lift
App Stores (Apple/Google)	Distribution platform	App review, policies, crash analytics, billing rules	Policy compliance, privacy labels, stable builds
Regulatory / Data-Protection Authorities	Compliance environment	Oversee privacy/consumer protections (GDPR-like), e-commerce rules	Consent, data minimization, deletion/export, auditability
Competitors	Market forces	Compete for users/catalogs	— (tracked for benchmarking and differentiation)
Hackers / Bad actors	Threat environment	Attempt fraud, scraping, abuse, account takeover	— (mitigated via monitoring, rate limits, anomaly/fraud detection)

4.4 Selection of Requirement Elicitation Methodologies

In this segment, we outline our methods for requirement-gathering for Lufyco. Due to the various user groups supported by the application and the different requirements, both technical and usability requirements, we adopt different approaches to capture accurate, actionable, inputs. Each method is chosen based on what it shows, how much effort it takes to conduct, and how well it clearly relates to FR/NFRs. Together, they will assist us in converting real user needs into testable, actionable requirements that are ready for building.

Table 14:Requirement Elicitation Methodologies

Literature Review
The author performed a literature review of fashion e-commerce and image-based retrieval and outfit recommendation. The review of current apps and research helped identify limitations such as context awareness, explainability, and mobile speed. These findings informed requirements that were based on the research and aligned with those of industry stakeholders regarding Lufyco. The review also pointed to constraints (data quality, fairness, latency) and areas for future enhancement.
Surveys
We conducted short in-app/web surveys that turn opinions into quantitative data. The ratings included satisfaction rating of image searching, explanations, and speed. The open-ended responses provided us with the types of wording that feel natural, and could be included in the UI. The recommendations included acceptance measures (e.g. SUS score, time to decision), and prioritized features.
Brainstorming Sessions
Mixed teams (Product, UX, engineers, stylist) sketched fast flows. We reviewed possibilities for Upload→Similar, Plan→Outfit, Save/Share. Ideas were further filtered by what is doable now & can be tested next. Outputs were a more focused backlog with MoSCoW priorities.

4.5 Discussion of Findings

For each method employed in the study, I have outlined the key findings below.

4.5.1 Literature Review Findings

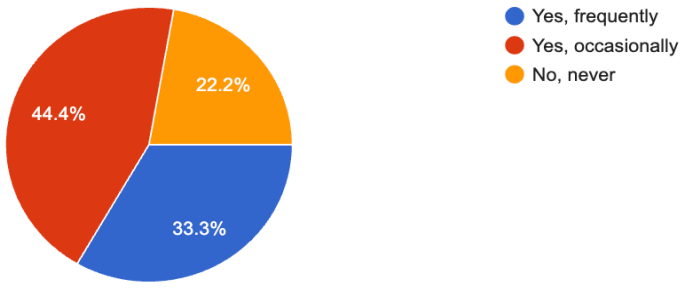
Table 15:Literature Review Findings

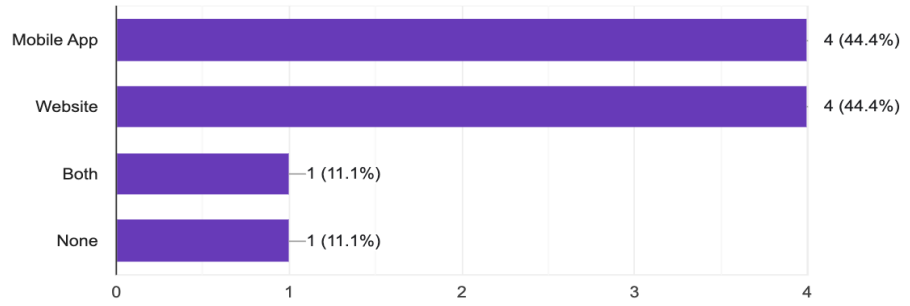
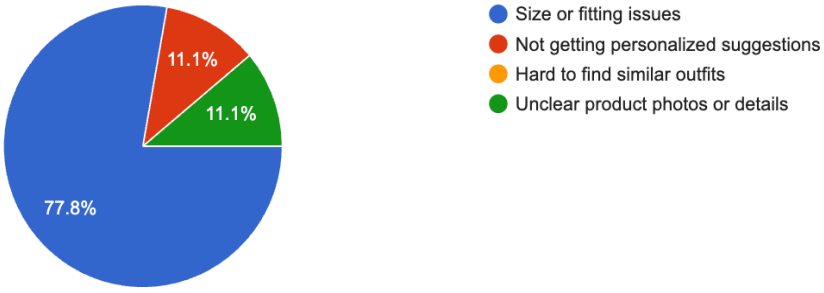
Citation	Finding
(Kiapour et al., 2015)	Street-to-shop retrieval highlights domain shift between user photos and studio images; robust visual embeddings and augmentation are required to keep Recall@K high.
(Liu et al., 2016)	Large, labeled fashion datasets improve attribute tagging and filtering; a standardized taxonomy reduces label mismatch across catalogs.

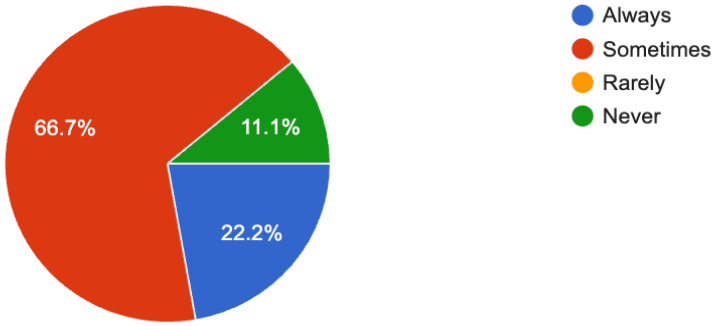
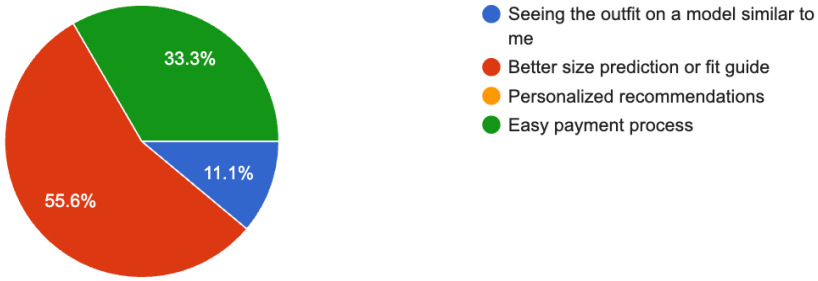
(Johnson et al., 2017)	Triplet/contrastive objectives outperform plain classification baselines for image retrieval, improving early-rank metrics such as Recall@10**.**
(Ge et al., 2019)	Modeling outfit compatibility (graph/listwise approaches) improves recommendation quality, but most systems lack transparent explanations**,** motivating a lightweight "why this" layer.
(Kohavi et al., 2009; Brooke, 1996)	Online A/B testing with user-centered metrics is essential to validate ranking changes and measure real UX impact.
(Saunders et al., 2019)	Mixed-methods evaluations (quant + qual) and pragmatic designs strengthen validity; triangulating results reduces bias and supports actionable FR/NFR derivation.

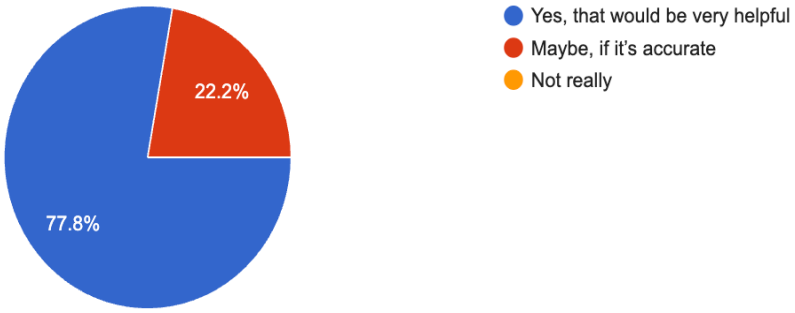
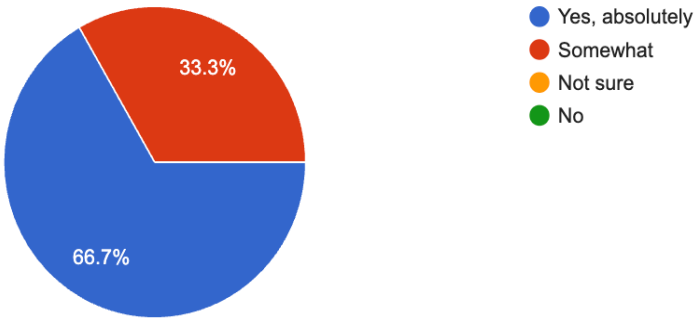
4.5.2 Survey Findings

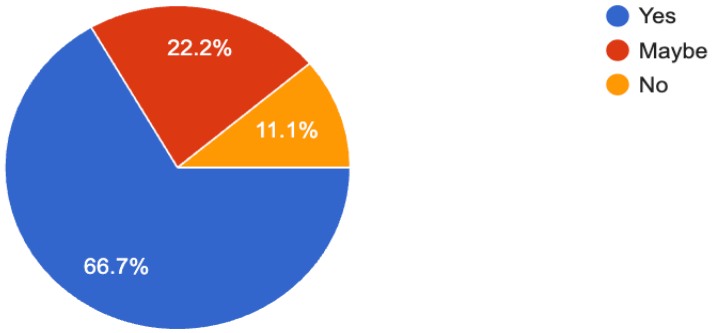
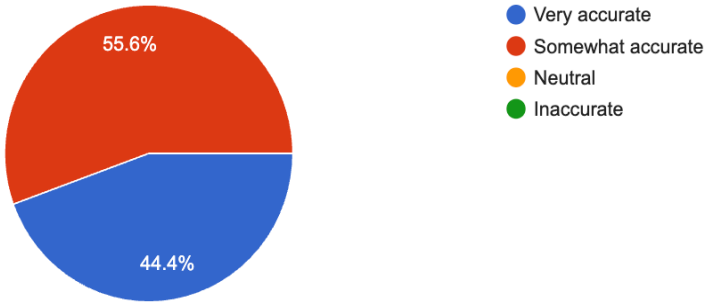
Table 16:Survey Findings

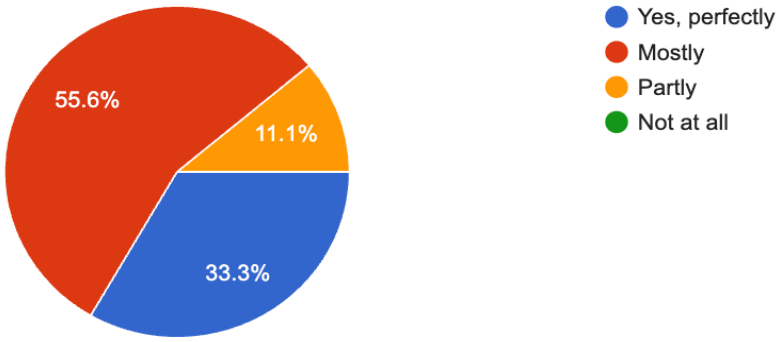
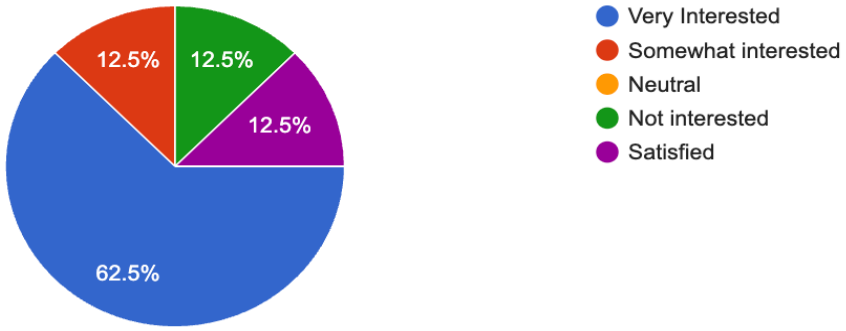
Have you used any online fashion or apparel shopping apps before?	
Aim	To gauge respondents' prior experience using online fashion/apparel shopping apps.
Results	
Findings & Conclusion	77.8% have used them (33.3% frequently, 44.4% occasionally); 22.2% never. Most are familiar—add simple onboarding for newcomers.
Which platform do you usually use for online shopping?	
Aim	Identify users' preferred platform for online shopping.

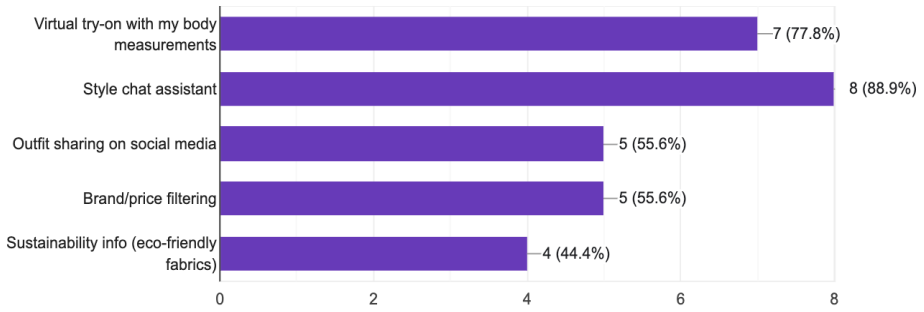
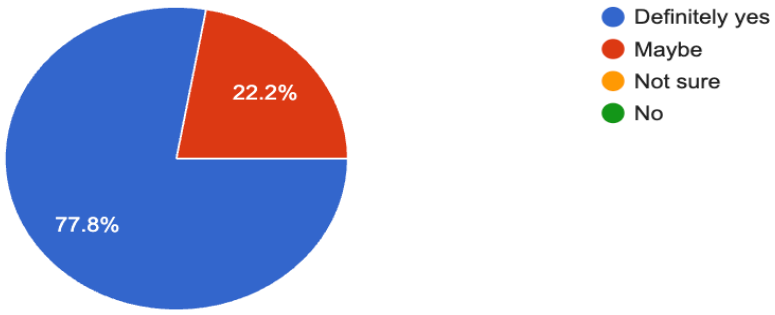
Results	
Findings & Conclusion	Usage splits evenly: 44.4% mobile app, 44.4% website, 11.1% both, 11.1% none. Build mobile-first with a strong responsive web, keep experiences consistent across both.
What difficulties do you face when buying clothes online?	
Aim	Find the biggest problems in online clothes shopping.
Results	
Findings & Conclusion	77.8% struggle with size/fitting; 11.1% want personalized suggestions; 11.1% note unclear photos/details. Focus first on fit guidance, then personalization and better images.
How often do you leave items in the cart without purchasing?	
Aim	Understand how often users abandon carts without buying.

Results	
Findings & Conclusion	66.7% abandon sometimes**,** 22.2% always**,** 11.1% never. Cart abandonment is common (88.9% at least sometimes). Prioritize nudges: fit/size confidence, price/shipping clarity, and reminder prompts to improve checkout.
What makes you more confident to complete a purchase?	
Aim	Identify which features most increase purchase confidence.
Results	
Findings & Conclusion	Better size prediction/fit guide (55.6%) matters most, then easy payment (33.3%)**,** then seeing outfits on similar models (11.1%)**.** Personalized recommendations: 0%**.** Focus on sizing first, checkout second.
Would you like an app that lets you upload a photo to find similar outfits automatically?	
Aim	Measure interest in a photo-upload “find similar outfit” feature.

Results	
Findings & Conclusion	77.8% said Yes; 22.2% said Maybe, if accurate; 0% Not really. Clear demand—build the feature, with priority on high matching accuracy to convert the "maybe" group.
Do you think AI-based outfit suggestions (based on mood, weather, or occasion) would improve your experience?	
Aim	Check if AI-based outfit suggestions (mood, weather, occasion) would improve shopping.
Results	
Findings & Conclusion	66.7% said Yes, absolutely; 33.3% said Somewhat; 0% Not sure/No. Strong support—implement AI stylist. Focus on quality/context accuracy to convert "somewhat" users.
Would you use a digital wardrobe feature to manage your owned clothes and get mix-and-match suggestions?	
Aim	Assess interest in a digital wardrobe for managing clothes and mix-and-match suggestions.

Results	
Findings & Conclusion	Yes 66.7%**,** Maybe 22.2%, No 11.1%**.** Strong interest; prioritize building the digital wardrobe, with onboarding/tutorials to convert "maybe" users.
Was the image search accurate in finding similar clothing items?	
Aim	Measure users' perceived accuracy of the image-search feature.
Results	
Findings & Conclusion	Very accurate 44.4%; Somewhat accurate 55.6%**.** Accuracy is generally good; focus on boosting precision (better embeddings/re-ranking) to shift "somewhat" to "very."
Did the AI Stylist's outfit suggestions match your preferences or occasion?	
Aim	Assess how well the AI Stylist's outfit suggestions match user preferences/occasions.

Results	
Findings & Conclusion	Mostly 55.6% (5/9)**;** Yes, perfectly 33.3% (3/9)**;** Partly 11.1% (1/9); Not at all 0%.Good fit for most users (89% mostly/perfect). Improve by refining preference capture and context signals to convert "mostly" → "perfect."
How interested would you be in using an app like Lufyco for online clothes shopping?	
Aim	Measure interest in using an app like Lufyco for online clothes shopping.
Results	
Findings & Conclusion	Very interested 62.5% (5/8); Somewhat interested 12.5% (1/8); Not interested 12.5% (1/8); Satisfied with current solutions 12.5% (1/8); Neutral 0%.Strong demand indicated.
Which additional features would you like to see?	
Aim	Identify which extra features users want most.

Results	
Findings & Conclusion	Prioritize Style Chat Assistant and Virtual Try-On for the MVP; add Filtering and Sharing next; Sustainability info can follow as a nice-to-have.
Would you recommend Lufyco to others if it becomes available?	
Aim	Measure willingness to recommend Lufyco.
Results	
Findings & Conclusion	Very strong referral potential; no negative sentiment. Convert "Maybe" users with accuracy demos, trust signals, and early incentives.

4.5.3 Brainstorming Findings

Table 17:Brainstorming Findings

Aspect	Finding
Photo-first discovery	Users prefer starting with Upload/Take Photo → Similar Items instead of typing; this should be the home action.
Explainability	A one-line "why this" under each recommendation increases trust and speeds decisions.
Latency & resilience	Slow networks caused hesitation; enforce p95 ≤ 1s, add caching, and provide offline/attribute-only fallbacks.

Taxonomy & sizes	Ambiguous color/size terms confused users; adopt a shared taxonomy and a size-mapping helper in filters.
Outfit editing	Users want quick swap of any item inside a suggestion with 2–3 alternatives, without leaving the card.
Catalog freshness	Shop owners need instant visibility after updates; support micro-updates to the vector index and show update status.

4.6 Summary of Findings

Table 18:Summary of Findings

Finding	Literature Review	Surveys	Brainstorming
Photo-first discovery improves relevance and engagement	✓	✓	✓
Low-latency retrieval is critical for satisfaction	✓	✓	✓
Short “why this” explanations increase trust and reduce edits	✓	✓	
Clean taxonomy and size-mapping reduce search friction	✓	✓	
Offline resilience and graceful fallbacks are expected on mobile			✓
Fresh catalog with micro-updates (near-real-time indexing) is required	✓		✓
Easy swap/alternatives inside outfits improves decision speed		✓	✓
Privacy controls (opt-in, delete/export) boost willingness to share photos	✓	✓	

4.7 Context Diagram

A Lufyco context diagram visualizes the app in the middle, connecting it to entities in its ecosystem.Users provide photos, moods/occasions, and clothing items, while Lufyco connects to a weather/auth service, cloud storage/vector service, payment service, shipping service, and

an AI service.Outputs include similar items, outfits suggestions, and justification of how Lufyco makes recommendations intelligently and confidently.

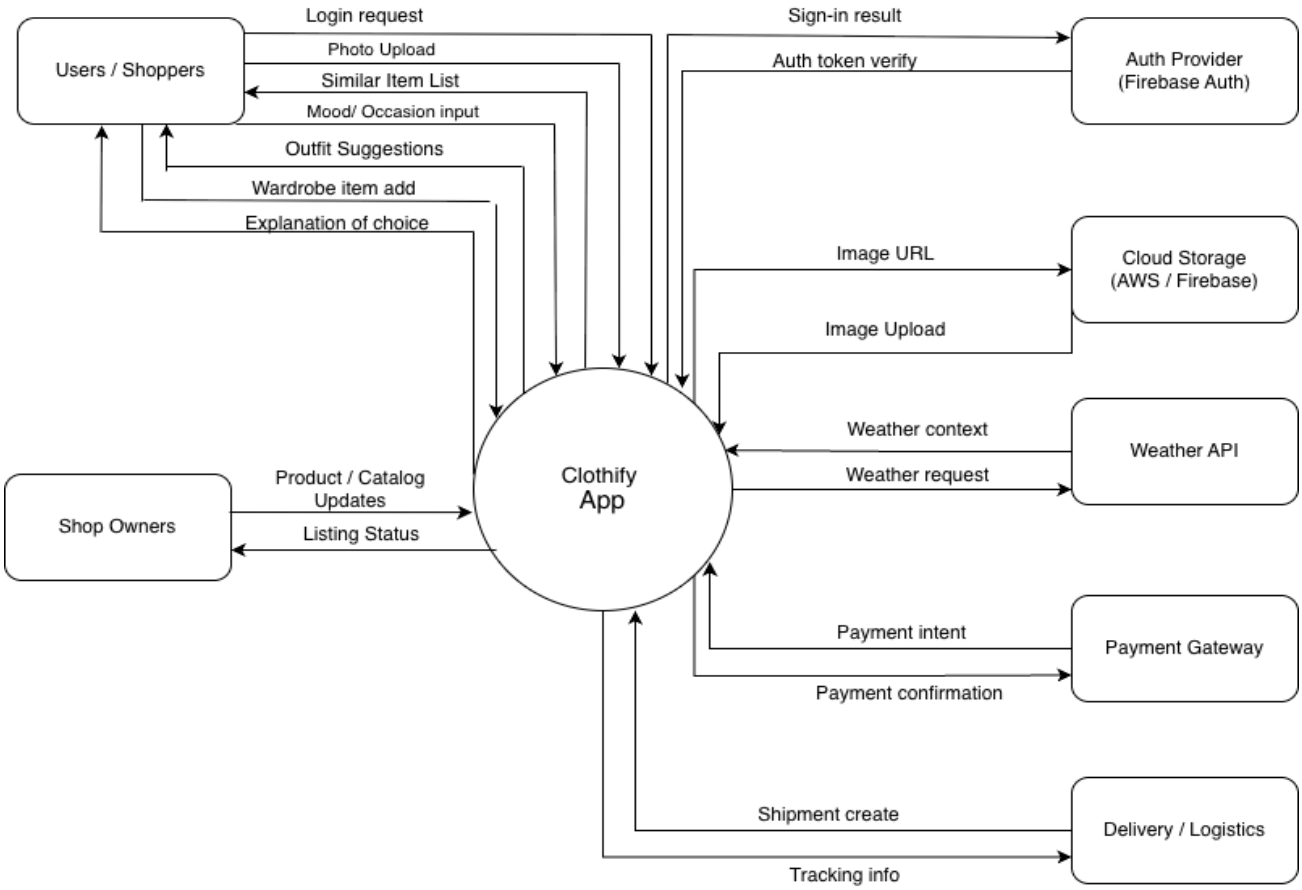


Figure 5:Context Diagram

4.8 Use Case Diagram

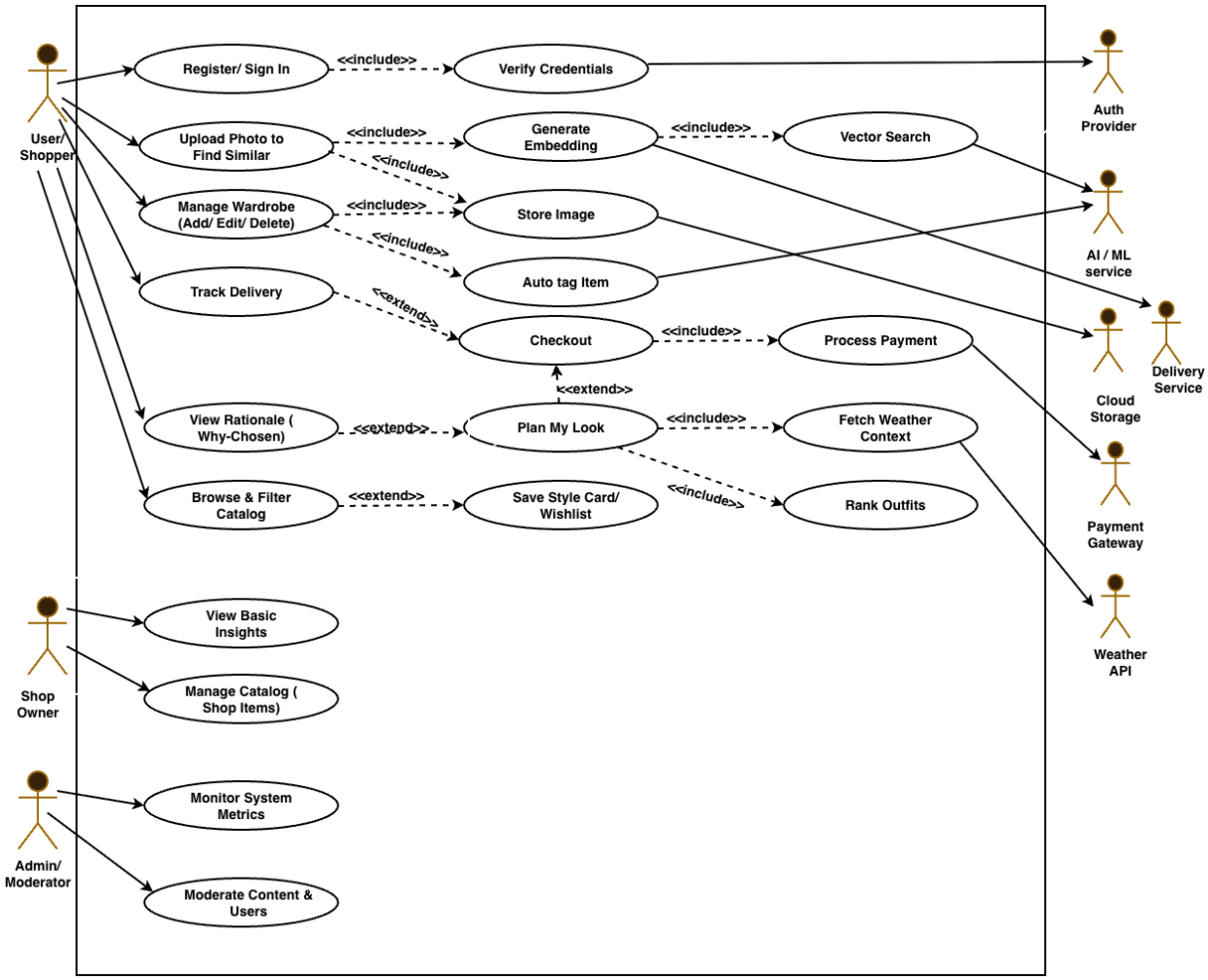


Figure 6:Use Case Diagram

4.9 Use Case Descriptions

The main use cases are described here and other use case descriptions can be found on Appendix A – Use Case Descriptions

Table 19:Use Case Descriptions

Use Case	Register / Sign In
Id	UC1
Description	Authenticates a shopper or shop owner to access Lufyco features.
Primary Actor	User/Shopper (or Shop Owner)
Supporting Actors	Auth Provider
Preconditions	App installed; network available.

Extended Use Cases	None
Included Use Cases	Verify Credentials (UC1a)
Main Success Scenario	1. User enters credentials or uses social login. 2. System verifies via auth provider. 3. User lands on home screen.
Alternative Flow	New user selects “Register” and creates an account.
Exceptional Flows	Invalid credentials; locked account; auth service timeout.
Postconditions	Session token created; profile available in app.
Comment	Foundation for all personalized features.

Use Case	**Upload Photo to Find Similar**
Id	UC2
Description	User uploads a photo; system returns visually similar items.
Primary Actor	User/Shopper
Supporting Actors	AI/ML Services, Cloud Storage
Preconditions	UC1 completed; photo selected or captured.
Extended Use Cases	None
Included Use Cases	Store Image (UC2a), Generate Embedding (UC2b), Vector Search (UC2c), Auto-tag Item (UC2d)
Main Success Scenario	1. User uploads image. 2. System embeds & searches index. 3. Similar items are listed.
Alternative Flow	User crops/retakes image before search.
Exceptional Flows	Unsupported format; size limit; ML service error.
Postconditions	Results view shown; user may save or refine.
Comment	Core entry to discovery.

Use Case	**Plan My Look (Outfit Suggestions)**
Id	UC3
Description	Generates outfit suggestions using mood/occasion, weather, and wardrobe/catalog; presents ranked looks with reasons.

Primary Actor	User/Shopper
Supporting Actors	AI/ML Services, Weather API
Preconditions	Signed in; items available; network on.
Extended Use Cases	None
Included Use Cases	Fetch Weather Context (UC3a), Rank Outfits (UC3b), View Rationale (UC3c)
Main Success Scenario	1. User sets mood/occasion. 2. System fetches context & composes outfits. 3. Ranked suggestions shown with "why-chosen."
Alternative Flow	Defaults used if user skips context.
Exceptional Flows	Context/ranking failures → safe fallback lists.
Postconditions	User can save/share/checkout.
Comment	Increases decision speed & trust.

Use Case	**Manage Wardrobe**
Id	UC4
Description	Users curate personal wardrobe with photos and tags.
Primary Actor	User/Shopper
Supporting Actors	Cloud Storage, AI/ML Services
Preconditions	Signed in.
Extended Use Cases	None
Included Use Cases	Store Image (UC2a), Auto-tag Item (UC2d)
Main Success Scenario	1. Add/edit/delete item. 2. System stores media & tags. 3. Wardrobe updates.
Alternative Flow	Manual tag correction.
Exceptional Flows	Upload failure; validation errors.
Postconditions	Updated wardrobe available to UC3/UC2.
Comment	Personalization anchor.

Use Case	**Browse & Filter Catalog**
Id	UC5

Description	Explore store catalog with filters/sort and details.
Primary Actor	User/Shopper
Supporting Actors	System
Preconditions	Catalog items exist.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Open catalog. 2. Apply filters/sort. 3. View item detail.
Alternative Flow	Switch to “similar items” from detail.
Exceptional Flows	No results; service unavailable.
Postconditions	Items shortlisted or added to cart.
Comment	Works with UC8 checkout.

Use Case	**Save Style Card / Wishlist**
Id	UC6
Description	Save outfits or items to a personal board for later.
Primary Actor	User/Shopper
Supporting Actors	System
Preconditions	Signed in; item/outfit selected.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Tap save. 2. System stores reference. 3. Item visible in Style Cards/Wishlist.
Alternative Flow	Organize into collections.
Exceptional Flows	Storage error.
Postconditions	Persistent bookmark created.
Comment	Feeds re-engagement.

Use Case	**Checkout**
Id	UC7
Description	Completes purchase of selected items securely.
Primary Actor	User/Shopper
Supporting Actors	Payment Gateway, Delivery Service

Preconditions	Items in cart; user address available.
Extended Use Cases	None
Included Use Cases	Process Payment (UC7a)
Main Success Scenario	1. Confirm cart/address/shipping. 2. Pay. 3. Order created & confirmation shown.
Alternative Flow	Apply promo; choose COD if enabled.
Exceptional Flows	Payment failure; out-of-stock item.
Postconditions	Order stored; tracking enabled.
Comment	PCI/DSS compliant via gateway.

Use Case	Track Delivery
Id	UC8
Description	Shows shipment status and ETA.
Primary Actor	User/Shopper
Supporting Actors	Delivery Service
Preconditions	Order shipped with tracking id.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. User opens order. 2. System fetches carrier status. 3. Updates timeline/ETA.
Alternative Flow	Push notifications for status changes.
Exceptional Flows	Carrier API down; invalid tracking.
Postconditions	Latest status visible.
Comment	Improves transparency post-purchase.

Use Case	Manage Catalog (Shop Items)
Id	UC9
Description	Shop owner adds/edits products with images, price, stock.
Primary Actor	Shop Owner
Supporting Actors	Cloud Storage, AI/ML Services
Preconditions	Authenticated as owner.

Extended Use Cases	None
Included Use Cases	Store Image (UC2a), Auto-tag Item (UC2d)
Main Success Scenario	1. Create or edit item. 2. Upload media/tags. 3. Publish to catalog.
Alternative Flow	Bulk upload via CSV.
Exceptional Flows	Validation errors; stock sync failure.
Postconditions	Catalog updated.
Comment	Directly affects discovery/ranking.

Use Case	View Basic Insights
Id	UC10
Description	Owner views sales, views, saves, and top items.
Primary Actor	Shop Owner
Supporting Actors	System
Preconditions	Transactions and events available.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Open analytics. 2. Select range. 3. See charts/tables.
Alternative Flow	Export CSV.
Exceptional Flows	No data for range.
Postconditions	Insight shared/acted on.
Comment	Guides merchandising.

Use Case	Monitor System Metrics
Id	UC11
Description	Admin monitors uptime, latency, error rates, ML health.
Primary Actor	Admin
Supporting Actors	System
Preconditions	Admin role; observability stack running.
Extended Use Cases	None
Included Use Cases	None

Main Success Scenario	1. Open dashboard. 2. Review alerts. 3. Trigger remediation/runbook.
Alternative Flow	Schedule reports.
Exceptional Flows	Missing metrics; alert storm.
Postconditions	Issues tracked and resolved.
Comment	Keeps SLAs/SLOs on target.

Use Case	Moderate Content & Users
Id	UC12
Description	Reviews reports, removes abusive content, restricts users.
Primary Actor	Moderator
Supporting Actors	System
Preconditions	Moderator role; reported items exist.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Open queue. 2. Review evidence. 3. Take action and notify user.
Alternative Flow	Escalate to admin.
Exceptional Flows	Appeal submitted by user.
Postconditions	Community standards enforced.
Comment	Safety & trust.

4.10 Requirements

The MoSCoW principle is used to prioritize requirements into:

- **M**: Must Have (mandatory).
- **S**: Should Have (important).
- **C**: Could Have (desirable).
- **W**: Will Not Have (out of scope).

4.10.1 Functional Requirements

Table 20:Functional Requirements

ID	Requirement	Priority
FR1	Register / Sign in Users can create accounts and log in via email/OAuth; sessions handled with JWT.	M
FR2	Upload Photo to “Find Similar” User uploads an image; system returns top-K visually similar products.	M
FR3	Browse & Filter Catalog Keyword search with filters (size, color, price, brand) and pagination.	M
FR4	Wardrobe Management Add/Edit/Delete personal items with image and auto-tags.	M
FR5	AI Outfit Suggestions Generate outfits using mood/occasion/weather + user wardrobe/catalog.	M
FR6	Why-Chosen Explanation Show short rationale for each suggestion to build trust.	M
FR7	Checkout & Payments Handoff to gateway; receive success/failure callback; store order status.	M
FR8	Order Tracking View order states: placed, confirmed, shipped, delivered.	M
FR9	Save Style Card / Wishlist Save and revisit outfits/items; manage list.	S
FR10	Context Signals Integration Use weather API and user-set occasion to re-rank results.	S
FR11	Shop Owner Catalog Management Shop CRUD for products (title, images, price, stock, tags).	S
FR12	Basic Shop Analytics Views, add-to-cart, conversions over last 30 days.	S
FR13	Delivery Partner Updates Receive shipment status via webhook/manual portal.	S
FR14	Social Share of Style Cards Share saved looks to social apps as image/link.	C
FR15	Virtual Try-On (Preview) Simple overlay preview for supported categories.	C
FR16	Multi-language UI Add one additional locale beyond English.	C
FR17	AR Fitting Room Full 3D/AR try-on experience.	W

FR18	Marketplace Escrow & Disputes Payment holding and dispute workflows.	W

4.10.2 Non-Functional Requirements

Table 21:Non-Functional Requirements

ID	Requirement	Priority
NFR1	Usability - UI must be simple, consistent, and easy to learn.	M
NFR2	Performance Efficiency - Search/results load within acceptable time (incl. model latency).	M
NFR3	Availability & Reliability - Service remains stable with minimal downtime.	M
NFR4	Security - Strong auth, encryption in transit/at rest, secure APIs.	M
NFR5	Privacy & Data Protection - Collect minimally; allow delete/export; follow policy.	M
NFR6	Scalability - Handles growth in users, items, and images without major slowdowns.	S
NFR7	Cross-Platform Compatibility - Works consistently on iOS, Android, and web.	M
NFR8	Maintainability & Modularity - Code structured for quick fixes and updates.	S
NFR9	Observability & Logging - Health metrics and logs for monitoring and troubleshooting.	S
NFR10	Accessibility - Basic support for screen readers and font scaling.	S
NFR11	Battery/Network Efficiency - Optimize image uploads/inference to save data and power.	C
NFR12	Offline Mode (Full) - Comprehensive offline features are not targeted for this release.	W

4.11 Chapter Summary

This chapter outlined the Software Requirement Specification for Lufyco. It described how requirements were elicited (literature review, interviews, surveys, brainstorming/observation) and translated into clear functional and non-functional requirements. Supporting models Rich Picture, Stakeholder Onion, Context, and Use Case diagrams—were used to capture stakeholders, system boundaries, and core interactions. Finally, priorities were assigned using the MoSCoW method so the February build targets Must-Have items first, while later milestones incorporate Should-Have and selected Could-Have enhancements.

References

Ajakan, H., Lample, G., Denoyer, L. & Ralaivola, L. (2014). Domain-Adversarial Training of Neural Networks. Available from https://arxiv.org/abs/1505.07818 [Accessed 12 March 2025].

Cui, Z., Liu, S., Zhang, Z., Cao, X. & Shan, S. (2019). Dressing as a Whole: Outfit Compatibility Learning Based on Node-wise Graph Neural Networks. Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19). Available from https://doi.org/10.1145/3331184.3331268 [Accessed 12 March 2025].

Ge, Y., Zhang, R., Wang, X. et al. (2019). DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images. CVPR 2019. Available from https://arxiv.org/abs/1901.07973 [Accessed 12 March 2025].

Guo, R., Sun, P., Lindgren, E., Gaur, Y., Simcha, D., Chern, F. & Kumar, S. (2020). Accelerating Large-Scale Inference with Anisotropic Vector Quantization (ScaNN). Proceedings of Machine Learning and Systems (MLSys). Available from https://research.google/pubs/scann-efficient-vector-similarity-search/ [Accessed 12 March 2025].

He, R., Chen, Z., Kan, M.-Y. & Chen, X. (2016). Neural Outfit Recommendation. Available from https://arxiv.org/abs/1707.05535 [Accessed 12 March 2025].

Hou, Z., Chang, X., Nie, L. & Snoek, C.G.M. (2019). Explainable Outfit Recommendation with Joint Outfit Matching and Comment Generation. Available from https://arxiv.org/abs/1904.02239 [Accessed 12 March 2025].

Jégou, H., Douze, M., Johnson, J. & Matas, J. (2017). Billion-Scale Similarity Search with GPUs (FAISS). Available from https://arxiv.org/abs/1702.08734 [Accessed 12 March 2025].

Kiapour, M.H., Yamaguchi, K., Berg, A.C. & Berg, T.L. (2015). Where to Buy It: Matching Street Clothing Photos in Online Shops. ICCV *2015.* Available from https://doi.org/10.1109/ICCV.2015.425 [Accessed 12 March 2025].

Liu, Z., Luo, P., Wang, X. & Tang, X. (2016). DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations. CVPR 2016. Available from https://arxiv.org/abs/1608.03088 [Accessed 12 March 2025].

Sun, K., Lu, S., Chen, E. et al. (2020). FashionBERT: Text and Image Matching in the Wild. Available from https://arxiv.org/abs/2005.09801 [Accessed 12 March 2025].

Tan, M. & Le, Q. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ICML 2019. Available from https://arxiv.org/abs/1905.11946 [Accessed 12 March 2025].

Wang, X., Chen, H., Zhang, Z., Shen, C. & Zhang, J. (2021). FashionCLIP: Connecting Language and Images for Fashion Understanding. Available from https://arxiv.org/abs/2108.09116 [Accessed 12 March 2025].

Zhang, R., Tang, S., Ge, Y., Yi, L. & Wang, X. (2020). Context-Aware Outfit Recommendation with Attribute-Aware Explainability. Available from https://arxiv.org/abs/2006.13174 [Accessed 12 March 2025].

APPENDIX A – USE CASE DESCRIPTIONS

Use Case	Verify Credentials
Id	UC1a
Description	Validates username/password or OAuth token.
Primary Actor	System
Supporting Actors	Auth Provider
Preconditions	Credentials provided.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. System sends request to auth provider. 2. Receives verification. 3. Issues session.
Alternative Flow	2FA challenge when required.
Exceptional Flows	Provider unreachable or token invalid.
Postconditions	Auth result returned to UC1.
Comment	Security-critical; logged.

Use Case	Store Image
Id	UC2a
Description	Securely stores the uploaded image and returns a reference.
Primary Actor	System
Supporting Actors	Cloud Storage
Preconditions	Valid image received.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Upload to storage. 2. Get URL/ID. 3. Persist reference.
Alternative Flow	Local cache if offline (queued upload).
Exceptional Flows	Storage error; quota exceeded.
Postconditions	Image reference available to UC2.
Comment	Encrypted at rest.

Use Case	Generate Embedding
Id	UC2b
Description	Produces vector representation from the image.
Primary Actor	System
Supporting Actors	AI/ML Services
Preconditions	Image accessible.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Call model. 2. Receive vector. 3. Hand off to search.
Alternative Flow	Use fallback model if primary unavailable.
Exceptional Flows	Model timeout; invalid output.
Postconditions	Vector ready for UC2c.
Comment	Versioned models tracked.

Use Case	Vector Search
Id	UC2c
Description	Searches ANN index for nearest items.

Primary Actor	System
Supporting Actors	ANN Index
Preconditions	Valid embedding available.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Query ANN. 2. Retrieve top-K. 3. Return to UC2.
Alternative Flow	Expand to category filters.
Exceptional Flows	Index unavailable; empty results.
Postconditions	Candidate list returned.
Comment	Logged for evaluation (Recall@K).

Use Case	Auto-tag Item
Id	UC2d
Description	Predicts attributes (color, sleeve, fit) for images.
Primary Actor	System
Supporting Actors	AI/ML Services
Preconditions	Image or item photo available.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Run attribute model. 2. Save tags. 3. Enrich search/ranking.
Alternative Flow	Manual edit by owner.
Exceptional Flows	Low confidence → flag for review.
Postconditions	Tags persisted.
Comment	Boosts retrieval quality.

Use Case	Fetch Weather Context
Id	UC3a
Description	Retrieves local weather to inform outfit choices.
Primary Actor	System

Supporting Actors	Weather API
Preconditions	Location permission or user city provided.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Call API. 2. Parse temp/precip. 3. Provide to UC3.
Alternative Flow	Use cached forecast if API fails.
Exceptional Flows	Permission denied; API error.
Postconditions	Context attached to session.
Comment	Improves relevance.

Use Case	Rank Outfits
Id	UC3b
Description	Orders outfit candidates using relevance, diversity, and constraints.
Primary Actor	System
Supporting Actors	AI/ML Services
Preconditions	Candidate outfits generated.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Score candidates. 2. Optimize list. 3. Return top-K.
Alternative Flow	Heuristic ranking if model unavailable.
Exceptional Flows	Empty candidates; tie handling.
Postconditions	Ranked list to UC3.
Comment	Tracks NDCG/coverage.

Use Case	View Rationale (Why-Chosen)
Id	UC3c
Description	Shows short explanation for each suggested outfit.
Primary Actor	User/Shopper
Supporting Actors	System
Preconditions	UC3 results available.

Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. User taps outfit. 2. System displays rationale. 3. User accepts or edits.
Alternative Flow	Expanded details on demand.
Exceptional Flows	Explanation generation fails → show basic tags.
Postconditions	User informed; can proceed.
Comment	Builds transparency.

Use Case	Process Payment
Id	UC7a
Description	Handles card/PayPal/etc. via gateway.
Primary Actor	System
Supporting Actors	Payment Gateway
Preconditions	Valid order total & billing info.
Extended Use Cases	None
Included Use Cases	None
Main Success Scenario	1. Tokenize & submit payment. 2. Receive confirmation. 3. Persist transaction id.
Alternative Flow	Retry or alternative method.
Exceptional Flows	Declined, timeout, fraud flag.
Postconditions	Payment status returned to UC7.
Comment	No card data stored by Lufyco.