Closet Canvas: Fit and Style-Aware Personalized Recommendations for Apparel

Ajay D. Sudhir
University of North Carolina at Charlotte
asudhir@charlotte.edu

Bruh Lemma Yadecha
University of North Carolina at Charlotte
byadecha@charlotte.edu

Abishek Arulkumar Padmapriya
University of North Carolina at Charlotte
aarulkum@charlotte.edu

Nitin Chandrasekhar
University of North Carolina at Charlotte
nchandr8@charlotte.edu

Mitchell Nguyen
University of North Carolina at Charlotte
mnguye65@charlotte.edu

Abstract

*Closet Canvas is a body-aware fashion recommendation system that matches users to clothes based on **fit** and **style**. Unlike traditional fashion recommendation systems that rely solely on user behavior data, Closet Canvas combines body profile extraction from video, per-size garment specifications, multimodal CLIP-based style embeddings, fit-based filtering, and online preference learning. This technical report documents the complete system architecture, machine learning models, data pipelines, and infrastructure components that power the recommendation engine.*

1. Introduction

Online fashion retail faces a fundamental challenge: the disconnect between digital garment representations and physical user compatibility. Closet Canvas addresses this by combining computer vision-based body measurement extraction with intelligent recommendation algorithms.

The system integrates five core capabilities: (1) **Body Profile Extraction** from video using pose estimation, (2) **Garment Specifications** with per-size measurement catalogs, (3) **Style Embeddings** using multimodal CLIP-based features, (4) **Fit-Based Filtering** comparing body measurements (b_{spec}) with garment measurements (g_{spec}), and (5) **Preference Learning** through online adaptation to user ratings.

1.1. Recommendation Scoring

The recommendation engine produces three primary scores:

- **Fit Score**: 0–100 based on measurement matching (chest, shoulder, waist, hip)
- **Preference Score**: 0–100 based on CLIP embedding similarity
- **Combined Score**: Weighted average (60% fit, 40% preference by default)

The combined score is calculated as:

$$S_{combined} = w_{fit} \cdot \frac{S_{fit}}{100} + w_{pref} \cdot \frac{S_{pref}}{100} \tag{1}$$

where $w_{fit} = 0.6$ and $w_{pref} = 0.4$ by default.

2. System Architecture

2.1. High-Level Architecture

The Closet Canvas system follows a microservices architecture with the following primary components:

- **Frontend (React)** – User interface for video capture and recommendation display
- **Capture Service (FastAPI)** – REST API for session management and file uploads
- **Ingest Workers** – Background job processors for ML inference
- **Recommendation Engine** – Core logic for fit analysis and preference scoring

The infrastructure layer provides:

- **MinIO** – S3-compatible object storage for videos, masks, and measurements
- **Redis** – Job queue (RQ) and pub/sub for real-time status updates
- **PostgreSQL** – Persistent storage for users, sessions, and metadata
- **GPU/CPU Workers** – Containerized workers for ML inference

2.2. Processing Pipeline Flow

The video processing pipeline consists of four sequential stages:

1. **CPU Worker (video queue)** – Video remuxing and preprocessing
2. **GPU Worker (gating queue)** – Video quality gating and person segmentation using SegFormer and SAM
3. **GPU Worker (smpl queue)** – Body measurement estimation using MediaPipe pose detection
4. **GPU Worker (recommendation queue)** – Fit analysis (b_{spec} vs g_{spec}) and preference scoring

Each stage publishes status updates via Redis pub/sub, which are forwarded to the frontend via WebSocket for real-time progress tracking.

3. Core Components

3.1. Body Capture Module

The body capture module uses YOLOv8s for person detection with bounding box extraction, frame cropping, and confidence-based filtering (threshold: 0.3).

3.2. Video Quality Gating

The quality gating module ensures video frames meet requirements for accurate body measurement estimation. We evaluate five weighted metrics: person visibility (40 pts, $>$10% coverage), body cutoff (15 pts, full body in frame), lighting (15 pts, mean gray $>$60), sharpness (20 pts, Laplacian variance $>$10), and contrast (10 pts, std dev $>$20). Individual frames must score $\geq$60, and $>$70% of frames must pass for video acceptance.

3.3. Catalog Segmentation

The catalog segmentation pipeline processes garment images using two models in sequence:

1. **SegFormer B3** (`sayeed99/segformer_b3_clothes`)
   - Semantic segmentation for clothing categories
   - Labels 1–17 for different garment types
2. **Segment Anything Model (SAM)** (`vit_h`)
   - Mask refinement using bounding box prompts
   - Produces high-quality segmentation masks

The pipeline flow is:

$$\text{Input Image} \xrightarrow{\text{SegFormer}} \text{Coarse Mask} \xrightarrow{\text{SAM}} \text{Refined Mask}$$

3.4. CLIP Embeddings

The style embedding module uses FashionCLIP (ViT-B/32) from `patrickjohncyh/fashion-clip`, generating 512-dimensional L2-normalized embeddings with GPU acceleration.

3.5. Body Measurement Estimation

The measurement estimation module uses MediaPipe Pose (model_complexity=2) to extract five key measurements: height (nose-to-ankle distance scaled to user input), shoulder width (left-right shoulder landmark distance), and chest, waist, and hip circumferences (computed from ellipse perimeter approximations). Circumference is calculated via:

$$P \approx \pi(a+b)\left(1+\frac{3h}{10+\sqrt{4-3h}}\right) \tag{2}$$

where $h = \frac{(a-b)^2}{(a+b)^2}$, and a, b are the semi-major and semi-minor axes. The system performs async multi-image inference (max_concurrency=4), fusing measurements from multiple frames with per-image quality feedback.

3.6. Fit Recommender

The fit recommender compares body measurements (b_{spec}) with garment specifications (g_{spec}). A measurement is acceptable if the difference is 1–3 cm, and overall fit requires all measurements to be acceptable.

Fit Score Calculation:

$$S_{fit} = 25\times \sum_{m\in\{chest,shoulder,waist,hip\}} \mathbf{1}[1 \leq |g_m - b_m| \leq 3] \tag{3}$$

where g_m and b_m are the garment and body measurements respectively, and $\mathbf{1}[\cdot]$ is the indicator function.

3.7. Preference Model

The preference learning module adapts to user ratings using signed weights: rating 1 maps to -1.0 (strong dislike), 2 to -0.25 (mild dislike), 3 to 0.0 (neutral), 4 to $+0.5$ (like), and 5 to $+1.0$ (strong like). The update rule is:

$$\vec{p} \leftarrow \text{normalize}\left(\vec{p} + \eta \cdot w \cdot \text{normalize}(\vec{e})\right) \tag{4}$$

where $\vec{p}$ is the preference vector, η is the learning rate, w is the rating weight, and $\vec{e}$ is the garment embedding.

Scoring: Cosine similarity between preference vector and garment embedding:

$$S_{pref} = \frac{\vec{p}\cdot\vec{e}}{||\vec{p}||\cdot||\vec{e}||} \in [-1,1] \xrightarrow{\text{normalize}} [0,100] \tag{5}$$

4. Data Pipeline

4.1. Video Capture Session

The capture flow follows these steps:

1. **Frontend** initiates capture session via `POST /v1/sessions`
2. **Capture Service** creates session with unique ID and upload URLs
3. **User** records video clips (WebRTC $\rightarrow$ blob $\rightarrow$ PUT to MinIO)
4. **Frontend** signals clip upload complete via `POST /v1/sessions/{id}/clips/{clip_id}/uploaded`

4.2. Video Processing Queue

We use Redis with RQ (Redis Queue) to manage four processing queues: `video` (CPU worker for remuxing), `gating` (GPU worker for quality checks and segmentation), `smpl` (GPU worker for body measurement), and `recommendation` (GPU worker for fit analysis and ranking).

4.3. Storage Architecture

MinIO provides S3-compatible object storage across five buckets: `uploads` (raw video clips), `processed` (preprocessed videos), `masks` (segmentation masks), `smpl-measurements` (body measurements), and `user-preferences` (serialized preference models).

5. Machine Learning Models

5.1. Model Summary

The system integrates five deep learning models for different computer vision tasks. For person detection, we employ YOLOv8s from Ultralytics, which operates on both CPU and GPU. Clothes segmentation uses SegFormer B3 from HuggingFace for semantic garment classification. Mask refinement leverages Meta's Segment Anything Model (SAM ViT-H) to produce high-quality segmentation boundaries. Body pose estimation relies on Google's MediaPipe Pose running on CPU for efficient keypoint detection. Finally, style embeddings are generated using FashionCLIP from HuggingFace, a vision-language model fine-tuned for fashion understanding, executed on GPU for optimal performance.

5.2. Model Weights

- **SAM ViT-H:** `sam_vit_h_4b8939.pth` (∼2.4GB) – Downloaded from HuggingFace
- **SegFormer B3:** Auto-downloaded from HuggingFace Transformers
- **FashionCLIP:** `patrickjohncyh/fashion-clip` from HuggingFace Hub

6. API Reference

6.1. REST API

(The FastAPI service exposes endpoints for session management (`POST /v1/sessions`, `POST /v1/sessions/{id}/clips`, `POST /v1/sessions/{id}/clips/{clip_id}/uploaded`) and user management `POST /v1/users/login`, `POST /v1/users/{id}/measurements`).

6.2. WebSocket Status Updates

The system uses Redis Pub/Sub with WebSocket bridge for real-time status updates. Status progression follows: `idle` → `recording` → `gating` → `smpl` → `finishing` → `recommending` → `complete`.

7. Infrastructure

7.1. Containerized Deployment

The system deploys via Docker Compose with seven services: React frontend (port 3000), FastAPI capture service (port 8000), CPU and GPU workers for processing, MinIO (ports 9000-9001), Redis (port 6379), and PostgreSQL (port 5432).

7.2. GPU Worker Configuration

GPU workers require NVIDIA GPU access with environment variables: `MODEL_DIR` for ML weights, `HF_ACCESS_TOKEN` for model downloads, and `USE_BODY_CAPTURE` to toggle between YOLOv8 and SegFormer pipelines.

8. Technology Stack

8.1. Implementation

The backend is implemented in Python 3.11+ with FastAPI, PyTorch 2.8+, OpenCV, HuggingFace Transformers (4.57.1), open-clip-torch (3.2.0), segment-anything (1.0), MediaPipe (0.10.21+), and Ultralytics (8.3.228+).

The frontend uses React 18+ with TypeScript, Vite for building, Tailwind CSS for styling, and React Hooks for state management.

Infrastructure components include Docker Compose for containerization, MinIO for S3-compatible storage, PostgreSQL 18 for persistence, Redis with RQ for job queuing, and Nginx for serving the frontend.

9. Experimental Validation

9.1. Preference Model Evaluation

We evaluate whether the rating–embedding combination compresses user preference into a single vector by:

1. Initializing a randomized user vector matching the CLIP embedding dimensionality.
2. Warming up on the first N rated samples, adding rating-weighted, L2-normalized embeddings and renormalizing after each update.
3. Performing online evaluation: for each subsequent embedding, predict a rating via cosine similarity, record accuracy (exact and within-1), and update the user vector with the new rating-weighted embedding.
4. Reporting final metrics (exact accuracy, within-1 accuracy, MAE) and saving the resulting user vector.

This procedure demonstrates how sequential rating-weighted embedding updates yield a compressed representation of user preference.

Table 1. Signed-weight evaluation runs

Number	Warmup	Evaluated	Exact acc	Within-1 acc	MAE
1	10	95	0.316	0.768	0.968
2	10	94	0.298	0.691	1.096
3	10	112	0.339	0.652	1.179
4	10	409	0.235	0.633	1.269
5	10	89	0.213	0.629	1.292

Even with only ten warmup interactions, these online accuracies outperform random-chance expectations (exact ≈ 0.20, within-1 ≈ 0.52 for uniform ratings); several runs are significantly above those baselines, showing that the compressed preference vector captures usable signal from limited

feedback.

References

[1] Himan Abdollahpouri and et al. Review of modern fashion recommender systems. *arXiv preprint*, 2024.

[2] Sijie Ge and et al. Openfashionclip: Vision-and-language contrastive learning with fashion-specific knowledge. *arXiv preprint*, 2024.

[3] Ruining He and Julian McAuley. Vbpr: Visual bayesian personalized ranking from implicit feedback. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[4] Menglin Jia and et al. Fashionpedia: Ontology, segmentation, and an attribute localization dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[5] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Yolov8 by ultralytics. *GitHub repository*, 2023.

[6] Muhammed Kocabas, Nikos Athanasiou, and Michael J. Black. Vibe: Video inference for human body pose and shape estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[7] Zhihao Li, Jianzhuang Xu, Yanyi Yu, Yifan Zhang, Ce Li, and Liang Zheng. Cliff: Carrying location information in full frames into human pose and shape estimation. In *European Conference on Computer Vision (ECCV)*, 2022.

[8] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. Smpl: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, 2015.