Neurose Virtual Fitting Room (VFR) – Technical Overview & System Blueprint

1. Project Vision

The Neurose Virtual Fitting Room (VFR) enables users to realistically visualize garments on themselves or chosen models using a hybrid AI pipeline that combines **geometry-aware 3D simulation**, **AI diffusion rendering**, and **photo-realistic garment synthesis**. The system aims to achieve **Kling AI-level realism** while maintaining scalability and cost-efficiency.

2. Core Objectives

1. Deliver ultra-realistic garment try-on experiences without CAD models.
2. Integrate Kling AI as a premium rendering backend for ultimate realism.
3. Maintain a full local fallback pipeline achieving comparable quality.
4. Support variable pricing through pay-as-you-go and tiered quality levels.
5. Provide API and SDK access for retailers and e-commerce platforms.

3. Key Functional Modules

3.1 User Interface (Front-End)

- Web and mobile app (React + Flutter)
- User photo upload & privacy consent
- Garment selection from retailer catalog
- Real-time preview and rendering queue
- User account management, subscription, and history

3.2 Backend API (FastAPI / Node.js)

- User management & authentication (Firebase Auth)
- Image upload & preprocessing service
- Rendering job queue (Redis + Celery)
- Result delivery & caching layer (PostgreSQL + S3)
- REST/GraphQL endpoints for partners

3.3 AI Rendering Pipeline

Hybrid Architecture: Local + Kling AI integration.

Stage	Model	Purpose	Cost
Person Parsing	SCHP / CIHP / LIP	Segment body & clothes	Free
Pose Extraction	OpenPose / YOLOv8-Pose / ControlNet	Extract joints	Free
Garment Warping	StableVITON / GP-VTON / ReclothVITON	Align garment to body	GPU cost only
Geometry Fitting	SMPL-X + Cloth Simulation	Physically accurate drape	GPU cost only
Final Rendering	Kling AI API	Photo-realistic result	Pay-per-image
Local Fallback	Flux / SDXL + IP-Adapter + InstantID	Kling-grade rendering	GPU cost only

4. Advanced Stack (Tier-S Hybrid System)

4.1 Person Canonicalization

- SMPL-X or GHUM model fit from user photo(s)
- Neural texture extraction for user identity
- Segmentation-based hair/skin protection masks

4.2 Garment Assetization

- Multi-photo garment reconstruction (front, side, detail)
- UV unwrapping and neural texture mapping
- Fabric classification for draping presets

4.3 Differentiable Draping

- Coarse cloth simulation with stiffness presets
- Collision & contact correction refinement
- Real-time pose-based garment fitting

4.4 Photoreal Finisher

- Gaussian-splat rendering for lighting & shading
- SDXL/Flux img2img polish (denoise 0.10–0.22)
- Multi-ControlNet (Pose + Depth + Normal + Seg + Edge)
- IP-Adapter (Garment) + InstantID/FaceID-Plus for identity and fabric detail

4.5 Post-Processing

- SDXL Refiner pass

- Real-ESRGAN upscale → downsample
- CodeFormer (face restoration)
- BGMv2 matting + relight blending

4.6 Quality Assurance & Retry Logic

- CLIP & LPIPS garment similarity
- Identity retention score
- Auto-retry with modified parameters
- Kling escalation on double-failure

5. Infrastructure Requirements

5.1 Hardware

Component	Minimum	Recommended
GPU	RTX 3090 / A6000	RTX 4090 / A100
VRAM	24 GB	48 GB+
CPU	i9 / Ryzen 9	Threadripper / Xeon
RAM	64 GB	128 GB
Storage	2 TB NVMe	4 TB NVMe SSD
Cloud Option	GCP / Vast.ai	AWS EC2 G6e / Lambda Labs

5.2 Software Stack

- OS: Ubuntu 22.04 LTS
- Backend: FastAPI, Celery, Redis, PostgreSQL
- AI: PyTorch 2.2+, Diffusers, ControlNet, IP-Adapter, InstantID, SMPL-X
- Cloud Integration: Firebase, S3, Kling AI API
- Frontend: React.js / Flutter
- Deployment: Docker Compose / Kubernetes (multi-GPU scaling)

6. Data Flow Overview

1. User uploads image(s) and selects garment.
2. Preprocessing: segmentation, pose, depth, and garment parsing.
3. If Kling quota available → send to Kling API.
4. If not → run local 2.5D/3D hybrid fallback pipeline.
5. Post-process output → store in cache → deliver to user.
6. Training data (Kling pairs) periodically used for LoRA distillation.

7. Functional Highlights

- Variable pricing & quality tiers (Basic, Pro, Ultra)
- Real-time rendering queue monitoring
- Retailer dashboard for SKU management
- Cache reuse (hash(user, garment, pose))
- Multi-user batch rendering optimization
- Model auto-updates via modular config registry

8. Future Enhancements

1. **Full 3D garment digital twin** generation from 2–3 photos.
2. **Virtual fitting avatars** with real-time animation for AR/VR use.
3. **Voice & gesture interface** integration using Neurose VLA stack.
4. **Personalized body-shape estimation** for size recommendation.
5. **Decentralized rendering mesh** (distributed GPU network for scale).

9. Summary

The Neurose Virtual Fitting Room fuses physics-based realism with neural rendering and modular scalability. By integrating Kling AI selectively and maintaining an advanced fallback stack, it achieves premium realism at optimized cost.

This document serves as the **technical and functional reference** for all development, infrastructure setup, and partnership discussions.

10. Model Registry & Pipeline Specs (Authoritative)

This section is the **single source of truth** for all AI models used in VFR, including purpose, inputs/outputs, and where they sit in the pipeline. Teams must keep this table and the configs below in sync with deployments.

10.1 End-to-End Pipeline (Top-Level)

1) **Pre-ingest** → file checks, EXIF strip, PII guard. 2) **User Canonicalization** → SMPL-X fit + neural texture + UV masks. 3) **Garment Assetization** → multi-photo mesh + UV + fabric tag. 4) **Draping** → cloth sim (coarse) + differentiable refinement, collisions. 5) **Photoreal Finisher** → (A) Gaussian-splat render → SDXL/Flux img2img polish; or (B) SDS UV texture bake (per SKU pose family). 6) **Post-processing** → Refiner, upscaler, relight, matting, seam clean. 7) **QA & Retry** → metrics checks; retry once; escalate to Kling if fail. 8) **Cache & Deliver** → CDN/S3, dedupe on (user, garment, pose) hash.

10.2 Model Registry (Table)

Module	Preferred Models	Framework	Inputs	Outputs	Notes
Parsing (person/clothes/hair/skin)	SCHP (CIHP/LIP alt)	PyTorch	RGB image	Segmentation masks (person/garment/skin/hair)	Use FP16; export to ONNX optional
Pose	YOLOv8-Pose (OpenPose alt)	PyTorch	RGB image	2D keypoints (17/25 set)	Fast and robust; cache per user image
Depth	ZoeDepth (MiDaS alt)	PyTorch	RGB image	Depth map	For ControlNet-Depth & silhouette carving
Normals (optional)	NormalBae / ControlNet-Normal	Diffusers	RGB + depth	Normal map	Boosts seam realism in finisher
Edge/Seams	HED / Lineart	PyTorch	RGB	Edge map	Guides collars/hem lines
Identity	InstantID **and** IP-Adapter FaceID-PlusV2	Diffusers	Face crop + reference	Face embedding/adapter features	Use both for stability + fidelity
Garment Style Adapter	IP-Adapter (Image-Plus)	Diffusers	Garment photo	Style/texture features	Drives fabric micro-detail
SMPL-X Fitting	SMPL-X + PIXIE/SMPLify-X	PyTorch	User image(s), keypoints	Body mesh, pose, UV	Persist per user/session
Garment Mesh from Photos	Multi-view recon: silhouette carving + ZoeDepth priors	PyTorch	2–3 garment photos	Coarse mesh + UV	No CAD required
UV Inpainting	LaMa (on UV)	PyTorch	Partial UV	Completed UV texture	Fill occlusion gaps
Fabric Classifier	Lightweight CNN (custom)	PyTorch	Garment crop	{denim, knit, satin, leather, printed}	Selects drape + BRDF preset

Module	Preferred Models	Framework	Inputs	Outputs	Notes
Draping (Coarse)	Fast cloth sim (Taichi/ARCSim-lite or NVIDIA Flex alt)	CUDA	Body mesh + garment mesh + pose	Draped mesh	10–30 ms target
Draping (Refine)	Differentiable refinement (projective, contact losses)	PyTorch/ CUDA	Draped mesh + masks	Collision-free, thickness-aware mesh	Corrects sleeve twist/collar lift
Gaussian-Splat Renderer	3D Gaussian Splatting	CUDA	Mesh + textures	Soft render (RGB+A)	Fast lighting/ shadows
Finisher (img2img)	**SDXL** or **Flux-1** + ControlNets	Diffusers	Base render + controls + adapters	Photo-real image	Denoise 0.10–0.24, steps 34–42
ControlNets	Pose, Depth, Seg, Normals, Edge, (Tile optional)	Diffusers	Above maps	Guidance features	Weights: Pose . 55, Depth .65, Seg .75, Normal . 45, Edge .35
Refiner	SDXL-Refiner	Diffusers	Finisher output	Polished image	Denoise 0.10–0.18
Upscaler	Real-ESRGAN x4 / SwinIR	PyTorch	Image	Upscaled image	Downsample to target after
Face Restore (if needed)	CodeFormer	PyTorch	Image	Face-enhanced image	Weight 0.5–0.7
Matting	BGMv2	PyTorch	Image	Alpha matte	Clean edges for composite
Relight	LUT/Light estimation (custom)	PyTorch	Image + bg	Relit image	Match original scene
Seam Cleanup	Poisson/Seamless clone	OpenCV	Image + mask	Artifact-free seams	Hem/collar fix
QA Metrics	CLIPScore, LPIPS, Aesthetic predictor	PyTorch	Image + refs	Scores	Thresholds below
Premium Backend	**Kling AI**	API	User + garment inputs	Ultra-real image	Use when escalated or for hero shots

10.3 Finisher (Img2Img) Baseline Config

```
IMG2IMG_DENOISE=0.20
STEPS=38
CFG=6.2
RES_LONG=1344
CTRL_POSE=0.55
CTRL_DEPTH=0.65
CTRL_NORMAL=0.45
CTRL_SEG=0.75
CTRL_EDGE=0.35
ADAPT_GARMENT=1.00
ADAPT_FACEID=0.85
REFINER_DENOISE=0.14
TILEPASS_DENOISE=0.12
TILEPASS_STEPS=16
```

10.4 Inputs/Outputs (I/O Contracts)

- **User Canonicalization**
- In: RGB image(s) (min 1024 px long side)
- Out: SMPL-X mesh (OBJ/NPZ), UV texture (2048^2 PNG), masks (PNG), keypoints (JSON)
- **Garment Assetization**
- In: 2–3 product photos ($\geq$ 1024 px), fabric tag (string)
- Out: Garment mesh (OBJ), UV (PNG), texture (2048^2 PNG)
- **Draping**
- In: Body mesh + garment mesh + pose
- Out: Draped garment mesh, collision map (NPZ)
- **Finisher**
- In: Soft render (PNG), control maps (PNG), adapters (features)
- Out: Final photo-real PNG/JPEG

10.5 QA Thresholds & Retry

- **Garment fidelity (CLIP/LPIPS):** ≥ 0.28 / ≤ 0.32
- **Aesthetics:** ≥ 0.58
- **Identity (face score):** ≥ 0.75
- **Leak/overpaint checks:** garment vs skin IoU ≥ 0.9
- **Retry policy:** at most 1 retry; adjust seed, +5 steps, +0.05 garment adapter, −0.02 denoise. If still failing → **route to Kling**.

10.6 Resource Profiles (Single 24 GB GPU)

- Controls: 0.3–0.6 s total (parallel)
- VTON expert: 0.9–1.8 s
- Img2img + Refiner: 6–9 s @ 1344 px
- Post: 0.6–1.0 s

- **E2E:** 8–12 s per image (batch 2–4). Cache user/garment assets.

10.7 Failure Modes & Fallbacks

- **Face drift** → raise FaceID weight; reduce denoise; apply CodeFormer.
- **Logo/print smear** → enable Tile Control pass; ensure UV baked texture exists.
- **Hem/collar melt** → increase Edge/Seg weights; Poisson seam fix.
- **Depth/pose mismatch** → recompute controls; prefer YOLOv8-Pose.
- **Lighting mismatch** → relight LUT; re-composite with matting.

10.8 Ownership & Responsibilities

- **ML-Perception Team:** parsing, pose, depth, normals, identity modules.
- **Geometry Team:** SMPL-X fitting, garment assetization, draping.
- **GenAI Team:** finisher configs, adapters, refiner, upscaling, post.
- **Backend Team:** job queue, caching, API, CDN, observability.
- **Integrations:** Kling API, retailer SDKs, admin dashboards.
- **QA/Ops:** metric thresholds, retries, dataset logging, LoRA distillation cadence.

10.9 Config Management

- All hyperparams stored in **YAML** (`/configs/pipeline.yaml`) with env overrides.
- Version every model weight/artifact in **Model Registry** (Postgres table + S3 path), with SHA256 and semantic version.
- Canary deploy via feature flags; roll back on trigger metrics.