

INVERSE GARMENT DESIGN WITH VISUAL AGENTS

Nanxiang Jiang¹, Yunuo Chen^{1,5}, Zeng Tao¹, Zizhang Li², Qi Guo³, Jiajun Wu²,
Yin Yang^{4,5}, Chenfanfu Jiang^{1,5}

¹University of California, Los Angeles ²Stanford University

³Independent Researcher ⁴University of Utah ⁵Envora



nxjiang-jnx.github.io/dress-agent

ABSTRACT

We study the problem of inverse garment design, reconstructing simulation-ready 3D outfits with arbitrary layers from a single image, with visual agents. Unlike differentiable optimization that is time-consuming, or feed-forward methods limited by training data, we introduce **DressAgent**, a hierarchical agentic system that recovers garment sewing patterns through planning and closed-loop refinement grounded in physical simulation. To this end, we first present an enhanced representation based on GarmentCode for diverse styles and arbitrary dressing layers, and build the first multi-layer sewing pattern dataset with over 15K outfits for supervision and benchmarking. Inverse design then proceeds coarse to fine: In *layer planning and coarse estimation*, agents infer layer order and relations, peel each layer into its own view, and estimate coarse patterns. In *fine-grained refinement*, an agentic trio of Tailor, Critic, and Orchestrator refines patterns in a closed loop of editing, simulating, and judging against the reference with visual and measured evidence. Carefully designed tools and an evolving memory make this refinement converge on the target geometry. We also design an agentic generation method for high-fidelity textures. DressAgent outperforms feed-forward methods by a large margin, surpasses or matches differentiable optimization in a fraction of the time, and generalizes to complex layering, diverse poses, and body shapes.

A couturier must be an architect for design, a sculptor for shape, a painter for color, a musician for harmony, and a philosopher for temperance.

— Cristóbal Balenciaga

1 INTRODUCTION

Imagine a photo of a beautiful outfit you would love to wear yourself. A skilled couturier can read that photo: how many layers, where the hem falls, how the fabric hangs on the body. From this reading, the couturier drafts the flat patterns, fits a trial on the body, and redraws them until the copy matches the original. We call this problem *inverse garment design*: from a single in-the-wild image, recover the outfit as sewing patterns ready to simulate, animate, and even manufacture (Fig. 1).

Recent work on inverse garment design with sewing patterns mainly follows two paradigms: *Differentiable optimization methods* (Li et al., 2024b; 2025; 2024a) simulate the garment, compare it with the image, and update the pattern by gradient descent. They recover faithful geometry, but are time-consuming and vulnerable to local minima in optimization. *Feed-forward methods* (Liu et al., 2023; Bian et al., 2025; Tao et al., 2026) train a network on large-scale datasets, and predict a pattern in one pass. They are efficient but limited by the data distribution and struggle to generalize to occlusion or complex layering. Vision-language models (VLMs) are recently used to write the pattern program in a single pass (Badalyan et al., 2026; Zhou et al., 2025), leveraging broad semantic priors to generalize beyond the training data and bypass costly optimization. However, this one-shot process is inherently open-loop, and without fine-grained visual grounding, even powerful VLMs struggle to recover subtle details. Moreover, people wear layered outfits in daily life (e.g., a cardigan over a dress), yet few methods support zero-shot recovery of arbitrary layers and their relations.



Figure 1: **We introduce DressAgent for inverse garment design with agentic planning and refinement.** (a) Results on in-the-wild photos with arbitrary layering, diverse poses, and body shapes. (b) The outfits drape and animate under physics. (c) Every garment is a sewing pattern with high-fidelity texture. (d) DressAgent pushes the Pareto frontier of quality and speed.

We propose **DressAgent**, a hierarchical agentic system for inverse garment design. Since garments can be written as code (Korosteleva & Sorkine-Hornung, 2023), our key insight is to solve inverse garment design the way agents already excel at coding: writing code and refining it against execution feedback. We therefore design a closed-loop system with agentic planning and refinement, grounded in physical simulation. To unleash this coding ability on real-world outfits, the agent needs a language expressive enough to write them, so we first introduce GarmentCode-XL (eXtended & Layered), an enhanced representation based on GarmentCode, which adds diverse styles and arbitrary dressing layers with their relations (e.g., tucked into, worn over), and extend the cloth simulator to realize these relations. As no data exists for this new language, we build GarmentCodeData-XL, the first multi-layer sewing pattern dataset with over 15K outfits for supervision and benchmarking.

Building on this foundation, DressAgent recovers the outfit in two stages, from coarse to fine. In *layer planning and coarse estimation*, a Layer Planner decomposes the outfit into ordered layers with carefully designed skills, and peels each layer into its own view. A Visual Reader then generates a style template per layer, decoded into a coarse program. In *fine-grained refinement*, three agents interact in a closed loop: a Tailor that edits and simulates the pattern, a Critic that judges the drape against the reference with visual and measured evidence, and an Orchestrator that manages the iterative refinement. With an evolving memory of past edits and their measured effects, the loop converges progressively. Since sewing patterns capture only geometry, we further design a novel agentic generation method with tool calls to recover high-fidelity textures from the input image.

Experiments show that DressAgent outperforms feed-forward methods by a large margin, surpasses or matches differentiable optimization in a fraction of the time, and achieves state-of-the-art preference (Fig. 1). In summary, our contributions are:

- We introduce DressAgent, a hierarchical agentic system for inverse garment design. It recovers outfits with arbitrary layers and their relations from a single in-the-wild image, through agentic planning and closed-loop refinement grounded in physical simulation.
- To unleash the most potential of coding agents on real-world garments, we propose GarmentCode-XL, an enhanced garment representation and GarmentCodeData-XL, the first multi-layer sewing pattern dataset with 15K outfits, together with an agentic method for high-fidelity textures.
- We show that DressAgent achieves the best overall accuracy and human preference, generalizing far beyond feed-forward methods at a fraction of the cost of differentiable optimization.

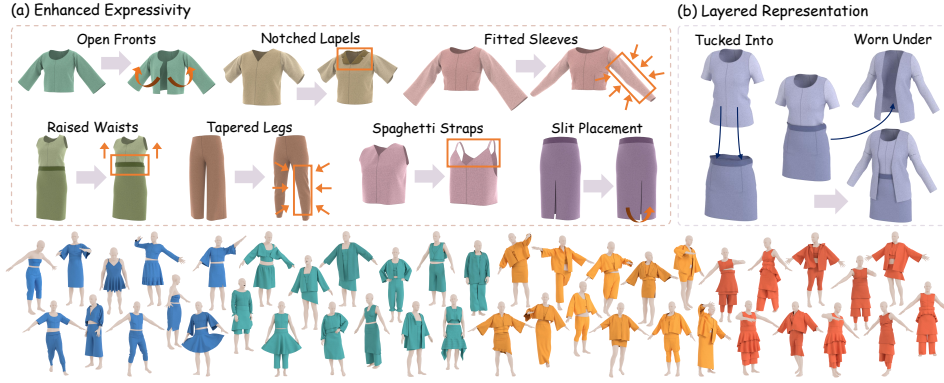


Figure 2: **Enhanced Garment Representation and Dataset.** *Top:* Examples of enhanced expressivity and layered representation in GarmentCode-XL. *Bottom:* Examples from GarmentCodeData-XL. Its 15,205 outfits contain 4,640 one-layer, 9,282 two-layer, 721 three-layer, and 562 four-layer examples. From left to right, colors indicate one to four layers. For more details, see Appendix B.

2 RELATED WORK

Garment Representations. 3D garment representations have been widely explored. *Template-based methods* (Jiang et al., 2020; Patel et al., 2020; Luo et al., 2024) deform a predefined mesh and stay topologically valid, but cannot generalize beyond that template. *Free-form fields* (Corona et al., 2021; De Luigi et al., 2023; Moon et al., 2022) represent arbitrary shapes but lose garment structure, and are hard to rig and simulate. *Sewing patterns* (Korosteleva & Lee, 2022; Li et al., 2024b) retain both: a garment is described by 2D panels and their stitches, just as it is sewn, so it stays editable and simulation-ready. GarmentCode (Korosteleva & Sorkine-Hornung, 2023) further writes the pattern as code, which is a natural fit for coding agents and the basis of our work.

Inverse Garment Design with Sewing Patterns. Single-image garment recovery with sewing patterns mainly follows two paradigms: *Differentiable optimization methods* (Li et al., 2024a; 2025; 2024c) fit each image by gradient descent through a differentiable cloth simulator, recovering faithful geometry at a high cost in time and expertise. *Feed-forward methods* (Liu et al., 2023; He et al., 2024; Weng et al., 2026; Tao et al., 2026) train on paired datasets and predict in one pass, but stay limited by that distribution and lose accuracy on in-the-wild photographs (e.g., occlusion, unusual poses, complex layering). VLMs are now used in both lines (Bian et al., 2025; Nakayama et al., 2025; Zhou et al., 2025) to estimate the pattern from an image, which generalizes better but remains open-loop, so even powerful VLMs cannot verify or refine subtle details.

Agentic Systems for Inverse Graphics. Agents can now solve hard tasks by writing code and refining it against execution feedback (Shinn et al., 2023; Yuksekgonul et al., 2024; Duan et al., 2026), and this loop is reaching inverse graphics: VLM agents refine 6D object poses from rendered feedback (Baik et al., 2026), and reconstruct Blender scenes through execution and verification (Yin et al., 2026). Garments are harder than such rigid scenes, since cloth is deformable with complex structures and relations. To the best of our knowledge, we introduce the first agentic system for inverse garment design, with planning and closed-loop refinement grounded in physical simulation.

3 DRESSAGENT

Overview. We present DressAgent, a hierarchical agentic system for inverse garment design from a single in-the-wild image. We first introduce an enhanced sewing pattern representation and the first multi-layer garment dataset (Sec. 3.1). Inverse design then proceeds coarse to fine: in *layer planning and coarse estimation*, agents infer ordered layers and their relations and estimate sewing pattern programs (Sec. 3.2). In *fine-grained refinement*, agents refine these programs through closed-loop tool calls to match the reference (Sec. 3.3). Finally, we design an agentic generative method for high-fidelity textures, and animate the outfit with physics-based simulation (Sec. 3.4).

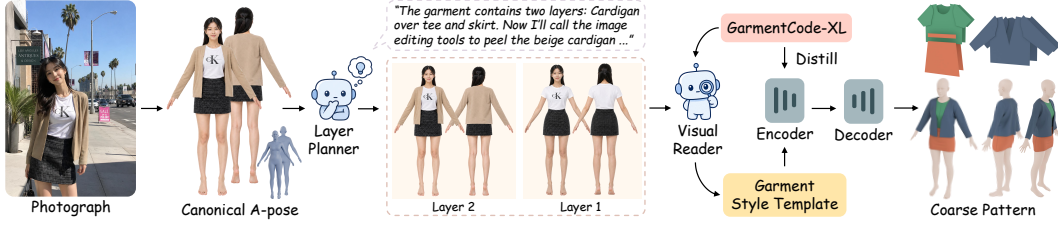


Figure 3: **Layer Planning and Coarse Pattern Estimation.** The Layer Planner reasons ordered layers and peels each into view. The Visual Reader generates a Garment Style Template per layer, then an encoder-decoder with distilled GarmentCode-XL semantics maps it to a coarse program.

3.1 GARMENT REPRESENTATION AND DATASET

In the garment design industry, a garment is made of flat panels sewn together along their edges. GarmentCode (Korosteleva & Sorkine-Hornung, 2023) turns this practice into code programs. A garment program P generates the sewing pattern S from body measurements B :

$$S = P(B) = \bigcup_i f_i(d_i, B), \quad P = \langle \mathcal{F}, \mathcal{D} \rangle, \quad (1)$$

where $\mathcal{F} = \{f_i\}$ are panel constructors and $\mathcal{D} = \{d_i\}$ are their design parameters. Each f_i drafts a component (e.g., a sleeve or bodice) using 2D curves controlled by d_i and body measurements B . For simulation, a Warp-based cloth solver (Macklin, 2022) stitches and drapes S into a mesh $M = \text{Drape}(P, B)$. Our goal is to recover P whose simulated garment matches the input image.

However, the original GarmentCode covers limited styles and no multi-layer outfits, so agents lack a language expressive enough for real-world outfits. We therefore introduce GarmentCode-XL (eXtended & Layered) with two main extensions (Fig. 2). First, we add new panel constructors to $\mathcal{F}$ and widen parameter ranges in $\mathcal{D}$ for diverse real-world styles (e.g., open-front shells, spaghetti straps). Second, we introduce a layered data structure $\mathcal{P} = \langle (P_1, \dots, P_K), \mathcal{R} \rangle$ with K layers ordered from inner to outer, and the relations $\mathcal{R}$ between pieces (e.g., tucked into, worn over). Each layer P_k contains optional upper and lower pieces (e.g., a tee with jeans, then a jacket over them). We simulate the outfit in this inner-to-outer order, using each settled layer as a collider for the next.

With this enhanced representation, we build GarmentCodeData-XL, the first multi-layer sewing pattern dataset (Fig. 2). It contains 15,205 synthetic outfits with their programs, sewing patterns, draped meshes, and rendered views. We use it for subsequent training and benchmarking.

3.2 STAGE 1: LAYER PLANNING AND COARSE ESTIMATION

Correct layer structure is the foundation of inverse garment design. This stage thus explicitly models the layer structure and gives initial estimates per layer. The workflow of Stage 1 is shown in Fig. 3.

Canonical A-pose Views. Garments are hard to read in an in-the-wild photograph due to occlusion, unusual poses, or poor image quality. We therefore canonicalize the photograph I into an A-pose view, where garment structure is easier to read (Korosteleva et al., 2024). We use an image editing model to generate the front view $I^f = g_{\text{canonical}}(I)$, standardizing pose and background while preserving clothes. The model also generates the back view $I^b = g_{\text{back}}(I^f)$, keeping the same body pose and position. From these views, we fit an SMPL-X (Pavlakos et al., 2019) body with NLF (Sárádi & Pons-Moll, 2024) and replace its hands using Hand4Whole++ (Moon, 2026) for finer finger geometry. This fit provides body measurements B for pattern drafting, a body mesh for simulation, and cameras π^f, π^b to align rendered garments with the canonical front and back views.

Layer Planning. To support zero-shot recovery of arbitrary-layer outfits, we need the fully visible views per layer as supervision. The *Layer Planner* agent uses a carefully designed reasoning skill to infer the layer structure and relations $\mathcal{R}$. It writes a description r_k for each layer, then calls image editing tools to generate the views layer by layer:

$$I_K^f = I^f, \quad I_{k-1}^f = g_{\text{peel}}(I_k^f, r_k) \quad (k = K, \dots, 2), \quad I_k^b = g_{\text{back}}(I_k^f), \quad (2)$$

where I_k^f and I_k^b are the front and back views in which layer k is the outermost garment. The tool g_{peel} is an image editor that removes the layer described by r_k .

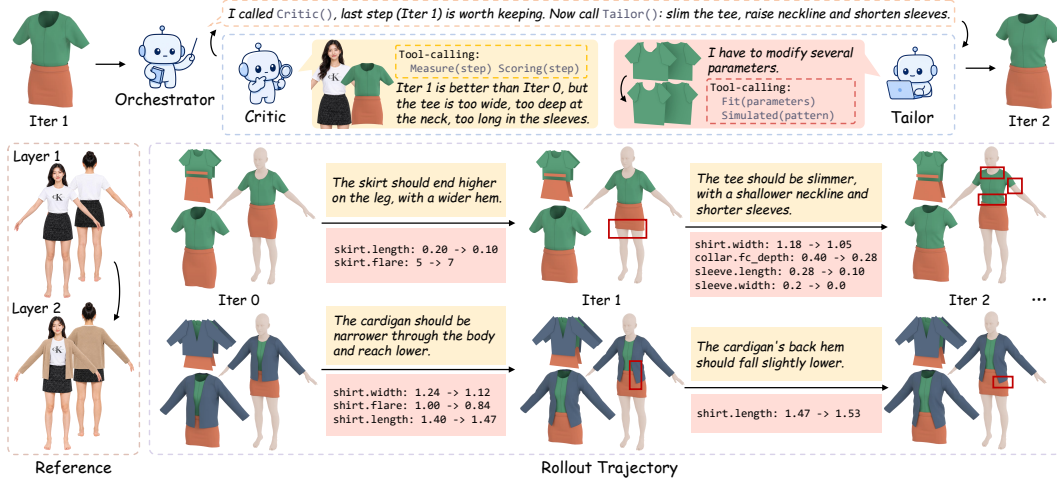


Figure 4: **Fine-Grained Refinement with an Agentic Trio.** *Top:* illustration of one refinement round (from Iter 1 to Iter 2 of the inner layer). The Critic scores each garment and identifies discrepancies, the Orchestrator saves a checkpoint and instructs the Tailor, and the Tailor applies the requested edits and simulates the pattern. *Bottom:* examples of rollout trajectories. For agent implementation, detailed prompts, and the complete record of this example, see Appendix D.

Template Generation. With every layer now visible, we first try prompting a pre-trained VLM to write GarmentCode-XL programs directly. However, the results are poor due to semantic misalignment (Bassamzadeh & Methani, 2024): the model struggles to interpret parameter names (e.g., SkirtCircle, Skirt2) and their bounds. After extensive exploration, we find a surprisingly direct solution: let an LLM read the source code to formulate visual questions for a *Visual Reader* agent to answer. The code reveals the geometric meaning of each parameter, allowing the model to interpret it correctly. A pre-trained LLM g_{ques} generates this questionnaire, with one visual question q_i about the garment property controlled by each parameter d_i :

$$\mathcal{Q} = g_{\text{ques}}(\mathcal{F}, \mathcal{D}) = (q_1, \dots, q_{|\mathcal{D}|}). \quad (3)$$

The Visual Reader answers $\mathcal{Q}$ for each layer using its views I_k^f, I_k^b and description r_k . We find it answers multiple choice questions more reliably than numerical questions. We therefore give each numerical question discrete options such as “quarter length”, “mid length”, or “full length”. Its answers form a *Garment Style Template* for each layer.

Program Decoding. We append an *encoder-decoder* to convert each template into an executable sewing pattern program. We distill GarmentCode-XL semantics into an encoder E through LoRA (Hu et al., 2022), using source code paired with visual descriptions and questions. This allows E to interpret visual answers in terms of the underlying program parameters (Appendix C.6). Then, we encode each question with its selected answer and take the final hidden state, projected through an MLP, as the conditioning vector for a lightweight autoregressive decoder Ψ (Radford et al., 2019). Trained on paired templates and programs, Ψ uses the learned parameter distribution to decode each garment style template into a complete design configuration. This configuration is a fixed sequence of $|\mathcal{D}| = 131$ parameter tokens, quantized by type (Appendix C.6). We decode one program P_k^0 per layer, then assemble these programs in dressing order into the coarse outfit.

3.3 STAGE 2: FINE-GRAINED REFINEMENT

Building on the coarse garment programs from Stage 1, our goal here is to align their draped geometry with the reference views through closed-loop refinement. We therefore design an agentic trio with complementary roles: a *Tailor*, a *Critic*, and an *Orchestrator*. This decomposition separates garment editing, evaluation, and control flow, enabling structured refinement while keeping each agent’s responsibility simple. It also reduces self-evaluation bias: an independent Critic is better positioned to identify errors that an edit-focused Tailor may overlook.

The **Tailor** proposes pattern edits using structured tools. The **Critic** evaluates the simulated garment against the reference for construction, proportion, fit, layering, and physical feasibility, providing

Algorithm 1 Fine-grained refinement of one dressing layer.

Require: coarse pattern P^0 of the current layer, body B , reference views I_k^f, I_k^b, I , budget T

- 1: $M^0 \leftarrow \text{Drape}(P^0, B)$; $(s^0, F^0, \cdot) \leftarrow \Phi_{\text{critic}}(I_k^f, I_k^b, I, O^0)$; $H^0 \leftarrow \{0\}$
- 2: **for** $t = 0, 1, \dots, T-1$ **do**
- 3: $(a^t, c^t, \cdot) \leftarrow \Phi_{\text{orch}}(H^t, s^t, F^t)$ ▷ Chooses Continue, Rollback, or Finish
- 4: **if** $a^t = \text{Finish}$ **then break**
- 5: **if** $a^t = \text{Rollback}$ **to** $j \in H^t$ **then** $(P^t, O^t, s^t, F^t) \leftarrow (P^j, O^j, s^j, F^j)$
- 6: $(u^t, \cdot) \leftarrow \Phi_{\text{tailor}}(c^t, F^t, O^t, H^t)$ ▷ Proposes a construction or size edit
- 7: $P^{t+1} \leftarrow P^t \oplus u^t$; $M^{t+1} \leftarrow \text{Drape}(P^{t+1}, B)$
- 8: $(s^{t+1}, F^{t+1}, \cdot) \leftarrow \Phi_{\text{critic}}(I_k^f, I_k^b, I, O^{t+1}, O^t)$ ▷ Judges with visual and metric evidence
- 9: $H^{t+1} \leftarrow H^t \cup \{t+1\}$ ▷ Context memory update
- 10: **end for**
- 11: **return** the best-scoring checkpoint in H^T

per-garment scores and natural-language feedback. The **Orchestrator** coordinates this interaction, determining when to accept edits, request further refinement, or finish the current layer. To prevent degradation during refinement, it saves pattern checkpoints and reverts to one when the Critic’s scores worsen. Refinement proceeds from inner to outer layers, as illustrated in Fig. 4.

Agent Tools. Agents interact with the garment exclusively through structured tools: *Observation tools* (e.g., *Render*, *Measure*) render the simulated garment beside the reference on a shared body ruler and measure every garment edge as a percentage of body height, providing complementary visual and metric evidence for judgement. *Modification tools* (e.g., *Fit*, *Simulate*) edit the sewing pattern and call the cloth solver to simulate it. *Control tools* (e.g., *Rollback*, *Finish*) manage global control operations. Tool access is role-dependent: the Critic is restricted to observation tools, the Tailor uses observation and modification tools, and the Orchestrator invokes the Tailor and Critic as tools while managing global control. The complete tool interface is provided in Appendix D.2.

Closed-Loop Refinement. Let P^t denote the sewing pattern program of the current layer at iteration t , $M^t = \text{Drape}(P^t, B)$ the simulated result, and O^t its observation through the two cameras together with the measurements. Given the reference views I_k^f, I_k^b and I , the Critic returns

$$(s^t, F^t, r_{\text{critic}}^t) = \Phi_{\text{critic}}(I_k^f, I_k^b, I, O^t, O^{t-1}), \quad (4)$$

where s^t is the per-garment scores, F^t is the list of discrepancies with their measured evidence, and r_{critic}^t is the rationale for the scores, which also states whether this step improves on the previous one. The Orchestrator reads the history H^t of all checkpoints and decides

$$(a^t, c^t, r_{\text{orch}}^t) = \Phi_{\text{orch}}(H^t, s^t, F^t), \quad (5)$$

where a^t is the action (chosen from *Continue*, *Rollback* and *Finish*), c^t is the instruction to the Tailor, which names the discrepancies to address in this round, and r_{orch}^t is the rationale for the action. The Tailor then produces the edit

$$(u^t, r_{\text{tailor}}^t) = \Phi_{\text{tailor}}(c^t, F^t, O^t, H^t), \quad P^{t+1} = P^t \oplus u^t, \quad (6)$$

where u^t is the edit and r_{tailor}^t is the rationale for the edit. The solver then simulates P^{t+1} , and the next iteration begins. We note that the edit takes two forms: a construction edit switches a categorical choice such as the neckline (e.g., crew neck or V-neck) or the garment structure, which resolves the ambiguity the coarse estimate might leave. A size edit tunes the numeric parameters so that the garment matches the reference precisely. The complete procedure is shown in Algorithm 1.

Evolving Memory for Convergent Fitting. To support convergence, we design a persistent, structured memory H^t that retains context across iterations. Each iteration archives sewing patterns, grounded observations, critiques, and all rationales. To bound context growth, views older than two iterations are compacted into summaries. Note that *Rollback* restores only the pattern and leaves the memory intact, so a failed attempt serves as a reference for later edits. As shown in Fig. 4, referencing past pattern edits alongside their visual and measured errors empowers the agents to refine progressively and converge on the target geometry. For instance, to land a cardigan hem on the reference, the Tailor actively contrasts prior length parameters (1.40 vs. 1.47) to deduce the precise value (1.53). Without this memory, each edit degrades into an isolated, single-step guess, and the loop oscillates instead of converging on precise parameters.

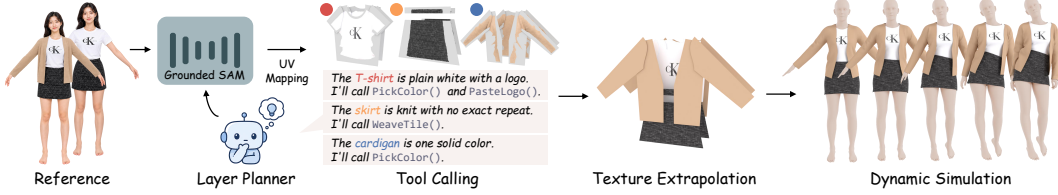


Figure 5: **Texture Generation and Simulation.** We design an agentic generative approach to recover textures for the sewing patterns, and use an XPBD-based cloth solver for dynamic simulation.

3.4 POST-PROCESSING

Texture Generation. To meet the needs of the garment fabrication industry, the final step of our method is to recover high-fidelity texture for every sewing pattern. Unlike direct UV mapping which can introduce boundary artifacts, we design an agentic generative approach (Fig. 5). First, we use Grounded SAM (Ren et al., 2024) to segment each garment with the Layer Planner’s descriptions, and map the pixels to its sewing pattern. We then divide common garment textures into five classes and design a dedicated texture extrapolation tool for each: `PickColor` reads a robust color for solid fabrics. `WeaveTile` turns an observed swatch into seamless knitted or woven texture with FabricDiffusion (Zhang et al., 2024). `RepeatPattern` extends the measured repeat of regular motifs (e.g., plaid, stripes) across the panels. `PasteLogo` pastes the logo back at its exact position, composable with any other tool. `ContinuePrint` handles all remaining prints that only admit reconstruction as a whole, keeping observed regions and generatively continuing them into unobserved cloth. Finally, we reuse the Layer Planner to classify each garment’s texture and call the corresponding tools. See Appendix E for detailed tool implementations.

Dynamic Simulation. We use an XPBD solver (Macklin et al., 2016) on Warp (Macklin, 2022) to stitch the seams and drape each layer under gravity and frictional contact with the kinematic SMPL-X body. To repose, we interpolate the body to the target pose as a moving boundary condition, advect the cloth by its nearest body points under gravity and contact, and reuse each settled inner layer as a moving collider for the next (Fig. 5).

4 EXPERIMENTS

4.1 IMPLEMENTATION DETAILS

Setup. We use GPT-5.6-luna (OpenAI, 2026a) with medium reasoning effort for all agents and g_{ques} . The image tools use GPT-Image-2 (OpenAI, 2026b). The encoder E is a fine-tuned Qwen3.6-27B (Qwen Team, 2026), and the decoder Ψ is trained the official training split of GarmentCodeData and the training split of our GarmentCodeData-XL. Full implementation details are in the Appendix.

Baselines. We compare with state-of-the-art sewing pattern reconstruction methods: Feed-forward methods include SewFormer (Liu et al., 2023), DressCode (He et al., 2024), ChatGarment (Bian et al., 2025), and GarmentGPT (Weng et al., 2026). Differentiable optimization methods include ISP (Li et al., 2024b), GarmentRecovery (Li et al., 2024a), and Dress-1-to-3 (Li et al., 2025).

4.2 RESULTS

Comparison to baselines. We report the *structure accuracy* of the sewing pattern, measured by accuracy for the number of panels, edges, and stitches on the GarmentCodeData (Korosteleva et al., 2024) test split, and the *geometry accuracy* of the draped garment, measured by Chamfer Distance (CD) and Intersection over Union (IoU) on CloSe (Antić et al., 2024) and 4D-Dress (Wang et al., 2024). The results are shown in Table 1. Our method outperforms feed-forward methods by a large margin, and surpasses or matches differentiable optimization in far less time. The gain comes from our enhanced GarmentCode-XL representation, per-layer planning and supervision, and closed-loop refinement by our agentic trio. Fig. 6 shows visual comparisons. DressCode produces coarse textures, GarmentRecovery often interpenetrates the body in simulation, ChatGarment and ISP recover plausible shapes but misjudge the number of layers, since neither models layer planning explicitly. In contrast, our method stays robust across complex layering, diverse poses, and body shapes.



Figure 6: **Qualitative comparison on in-the-wild images.** *Top five:* reconstructing outfits with one to three layers (left to right). *Bottom five:* generalization to diverse poses and body shapes. ChatGarment, ISP, and GarmentRecovery lack texture generation, so we show only their geometry.

Table 1: **Quantitative comparison.** We report the structure accuracy of sewing patterns and the geometry accuracy of draped garments. ISP and GarmentRecovery are excluded from structure accuracy, as they do not produce explicit sewing patterns with panel geometry. Our method achieves the best quality on most metrics, and is significantly faster than differentiable optimization.

METHOD	STRUCTURE ACC. $\uparrow$			GEOMETRY (CLOSe)		GEOMETRY (4D-DRESS)		AVG. TIME $\downarrow$ (MIN)
	PANEL	EDGE	STITCH	CD $\downarrow$	IoU $\uparrow$	CD $\downarrow$	IoU $\uparrow$	
SewFormer	18.67	28.80	7.24	7.40	0.348	7.58	0.496	2.0
ChatGarment	51.84	57.15	70.18	2.39	0.580	1.94	0.763	4.2
GarmentGPT	43.62	38.54	65.43	5.50	0.381	10.46	0.204	0.4
DressCode	19.67	26.47	30.97	2.77	0.542	2.56	0.685	2.2
ISP	-	-	-	1.79	0.687	1.16	0.859	90.0
GarmentRecovery	-	-	-	1.77	0.740	1.51	0.865	25.0
Dress 1-to-3	17.67	31.07	26.14	1.63	0.652	2.05	0.617	176.0
Ours	87.36	76.56	88.90	1.20	0.764	1.21	0.880	8.6

User study. To capture perceptual quality that metrics cannot measure, we conduct a user study on 200 cases, mixing cases from our experiments, in-the-wild photos randomly sampled from Pinterest¹, and photos uploaded in advance by users (Fig. 7). For each case, participants pick the result that best matches the photo in geometry, and rate the simulation plausibility of each garment on the body and the fidelity of its texture from 1 to 5. We hire 65 participants, and Table 2 shows the results. Our method leads by a large margin on all three criteria, indicating strong perceptual quality in geometric reconstruction, simulation plausibility, and texture fidelity.

4.3 ABLATION STUDY

Foundation Models and Iteration Budget. We benchmark different model backbones on 200 cases from our GarmentCodeData-XL test set, as shown in Fig. 8. The coarse estimate varies widely

¹<https://www.pinterest.com>

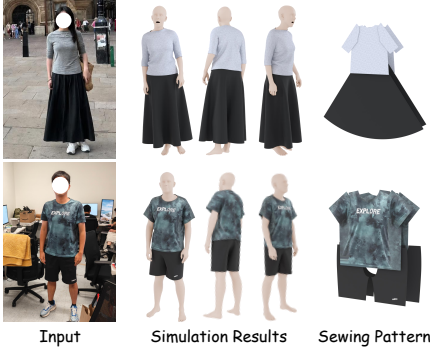


Figure 7: User-uploaded photo results.

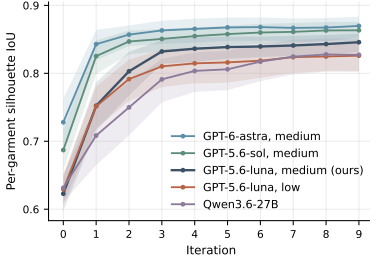


Figure 8: Ablation of foundation models and iteration budget. Iteration 0 is the coarse estimate, and the bands show the 95% bootstrap confidence interval.

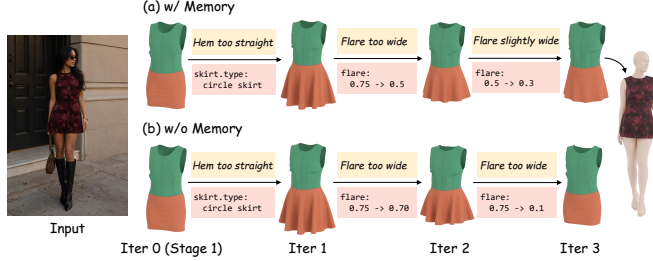


Figure 9: Ablation on evolving memory. We show how the flare of the skirt is adjusted over iterations. With memory, the agents keep the measurements of earlier steps as reference and converge progressively. Without it, the agents suffer from severe context loss and oscillate.

across models, but the refinement loop closes most of the gap and converges after about 6 iterations. To balance quality and cost, we adopt GPT-5.6-luna (medium) with a budget of $T = 6$ as default.

Evolving Context Memory. Fig. 9 demonstrates the necessity of context memory. With the measured effect of every earlier edit in context, the agents align the skirt flare onto the target progressively. Without it, each edit degrades into an isolated guess and the skirt flare oscillates.

Representation and Agentic Trio. We ablate the refinement stage, GarmentCode-XL, the Critic, and the Measure tool on 200 synthetic data from our GarmentCodeData-XL test set and 200 in-the-wild photos from DeepFashion2 (Ge et al., 2019) (Table 3). Without refinement, the coarse estimate is a one-pass predictor trained on our data, and the closed loop lifts it by a large margin on both synthetic and real photos. Beyond refinement, reverting to the original GarmentCode costs the most, as it cannot recover a style outside the design space. Without the Critic, the Tailor falls into self-bias (Xu et al., 2024; Panickssery et al., 2024; Huang et al., 2024), overrating flawed proposals. Without Measure, the agents lack the measured error to reason out a precise edit, so accuracy drops slightly.

Others. For more experiments (e.g., quantitative texture comparison, additional visual results) and ablations (e.g., rollback, texture generation), see Appendix F and Appendix G, respectively.

5 CONCLUSION

We present DressAgent, a hierarchical agentic framework for simulation-ready inverse garment design from a single in-the-wild image. We first introduce GarmentCode-XL, an enhanced representation for diverse styles and arbitrary layers, and GarmentCodeData-XL, the first multi-layer sewing

Table 2: **User study.** We report the percentage of votes as the best match to the photograph for geometry, and the average rating from 1 to 5 for plausibility and texture.

METHOD	GEOMETRY $\uparrow$	PLAUSIBILITY $\uparrow$	TEXTURE $\uparrow$
SewFormer	0.00%	1.68	2.86
ChatGarment	14.50%	3.61	-
GarmentGPT	0.00%	1.24	-
DressCode	9.50%	2.69	2.57
ISP	9.00%	4.43	-
GarmentRecovery	8.50%	3.02	-
Dress 1-to-3	4.50%	3.17	3.42
Ours	54.00%	4.51	4.33

Table 3: **Ablation on representation and the agentic trio.** DeepFashion2 IoU is measured against the annotated clothing masks.

COMPARISON	GARMENTCODEDATA-XL		DEEPPHOTO2
	STRUCT.	ACC. $\uparrow$	IoU $\uparrow$
w/o Refinement	63.25	0.623	0.562
w/o GarmentCode-XL	57.99	0.574	0.650
w/o Critic	69.56	0.822	0.765
w/o Measure	67.21	0.804	0.772
Full	73.59	0.839	0.783

pattern dataset with over 15K outfits. For coarse estimation, a Layer Planner decomposes the outfit into ordered layers and a Visual Reader estimates each layer. For fine refinement, an agentic trio of Tailor, Critic, and Orchestrator edits each pattern in a closed loop with evolving memory. We further generate high-fidelity textures with agentic tools. Experiments demonstrate the best overall accuracy and preference, generalization far beyond feed-forward methods, and far less time than differentiable optimization. We hope this work advances the garment industry and inverse graphics.

REFERENCES

- Dimitrije Antić, Garvita Tiwari, Batuhan Ozcomlekci, Riccardo Marin, and Gerard Pons-Moll. Close: A 3d clothing segmentation dataset and model. In *International Conference on 3D Vision*, 2024.
- Anna Badalyan, Pratheba Selvaraju, Giorgio Becherini, Omid Taheri, Victoria Fernandez Abrevaya, and Michael Black. Ngl: Natural garment language for training-free sewing pattern estimation. *arXiv preprint arXiv:2602.20700*, 2026.
- Sangwon Baik, Gunhee Kim, Mingi Choi, and Hanbyul Joo. Text-guided 6d object pose rearrangement via closed-loop vlm agents. *arXiv preprint arXiv:2604.09781*, 2026.
- Nastaran Bassamzadeh and Chhaya Methani. A comparative study of dsl code generation: Fine-tuning vs. optimized retrieval augmentation. *arXiv preprint arXiv:2407.02742*, 2024.
- Siyuan Bian, Chenghao Xu, Yuliang Xiu, Artur Grigorev, Zhen Liu, Cewu Lu, Michael J Black, and Yao Feng. Chatgarment: Garment estimation, generation and editing via large language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 2924–2934, 2025.
- Black Forest Labs. FLUX.1, 2024. URL <https://github.com/black-forest-labs/flux>.
- Enric Corona, Albert Pumarola, Guillem Alenya, Gerard Pons-Moll, and Francesc Moreno-Noguer. Smplicit: Topology-aware generative model for clothed people. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11875–11885, 2021.
- Luca De Luigi, Ren Li, Benoit Guillard, Mathieu Salzmann, and Pascal Fua. Drapenet: Garment generation and self-supervised draping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1451–1460, 2023.
- Shiying Duan, Pei Ren, Nanxiang Jiang, Zhengping Che, Jian Tang, Zhaoxin Fan, Yifan Sun, and Wenjun Wu. Robopara: Dual-arm robot planning with parallel allocation and recomposition across tasks. *arXiv preprint arXiv:2506.06683*, 2026.
- Yuying Ge, Ruimao Zhang, Lingyun Wu, Xiaogang Wang, Xiaoou Tang, and Ping Luo. A versatile benchmark for detection, pose estimation, segmentation and re-identification of clothing images. *CVPR*, 2019.
- In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, 2024.
- Kai He, Kaixin Yao, Qixuan Zhang, Jingyi Yu, Lingjie Liu, and Lan Xu. Dresscode: Autoregressively sewing and generating garments from text guidance. *ACM Transactions on Graphics (TOG)*, 43(4):1–13, 2024.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2024.
- Boyi Jiang, Juyong Zhang, Yang Hong, Jinhao Luo, Ligang Liu, and Hujun Bao. Bcnet: Learning body and cloth shape from a single image. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*, pp. 18–35. Springer, 2020.
- Maria Korosteleva and Sung-Hee Lee. Neuraltailor: Reconstructing sewing pattern structures from 3d point clouds of garments. *ACM Transactions on Graphics (TOG)*, 41(4):1–16, 2022.

- Maria Korosteleva and Olga Sorkine-Hornung. Garmentcode: Programming parametric sewing patterns. *ACM Transactions on Graphics (TOG)*, 42(6):1–15, 2023.
- Maria Korosteleva, Timur Levent Kesdogan, Fabian Kemper, Stephan Wenninger, Jasmin Koller, Yuhan Zhang, Mario Botsch, and Olga Sorkine-Hornung. Garmentcodedata: A dataset of 3d made-to-measure garments with sewing patterns. In *European Conference on Computer Vision*, pp. 110–127. Springer, 2024.
- Ren Li, Corentin Dumery, Benoît Guillard, and Pascal Fua. Garment recovery with shape and deformation priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1586–1595, 2024a.
- Ren Li, Benoît Guillard, and Pascal Fua. Isp: Multi-layered garment draping with implicit sewing patterns. *Advances in Neural Information Processing Systems*, 36, 2024b.
- Xuan Li, Chang Yu, Wenxin Du, Ying Jiang, Tianyi Xie, Yunuo Chen, Yin Yang, and Chenfanfu Jiang. Dress-1-to-3: Single image to simulation-ready 3d outfit with diffusion prior and differentiable physics. *ACM Transactions on Graphics (TOG)*, 44(4):1–16, 2025.
- Yifei Li, Hsiao-yu Chen, Egor Larionov, Nikolaos Sarafianos, Wojciech Matusik, and Tuur Stuyck. Diffavatar: Simulation-ready garment optimization with differentiable simulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4368–4378, 2024c.
- Lijuan Liu, Xiangyu Xu, Zhijie Lin, Jiabin Liang, and Shuicheng Yan. Towards garment sewing pattern reconstruction from a single image. *ACM Transactions on Graphics (TOG)*, 42(6):1–15, 2023.
- Zhongjin Luo, Haolin Liu, Chenghong Li, Wanghao Du, Zirong Jin, Yinyu Nie, Weikai Chen, and Xiaoguang Han. Garverselod: High-fidelity 3d garment reconstruction from a single in-the-wild image using a dataset with levels of details. In *ACM Transactions on Graphics (TOG)*, 2024.
- Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics, 2022. URL <https://github.com/NVIDIA/warp>. NVIDIA GPU Technology Conference (GTC).
- Miles Macklin, Matthias Müller, and Nuttapong Chentanez. Xpbd: Position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, pp. 49–54, 2016.
- Gyeongsik Moon. Enhancing hands in 3d whole-body pose estimation with conditional hands modulator. *arXiv preprint arXiv:2603.14726*, 2026.
- Gyeongsik Moon, Hyeongjin Nam, Takaaki Shiratori, and Kyoung Mu Lee. 3d clothed human reconstruction in the wild. In *European conference on computer vision*, pp. 184–200. Springer, 2022.
- Kiyohiro Nakayama, Jan Ackermann, Timur Levent Kesdogan, Yang Zheng, Maria Korosteleva, Olga Sorkine-Hornung, Leonidas J Guibas, Guandao Yang, and Gordon Wetzstein. AIpparel: A multimodal foundation model for digital garments. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 8138–8149, 2025.
- OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. <https://openai.com/index/gpt-5-6/>, 2026a.
- OpenAI. Introducing gpt-image-2. <https://developers.openai.com/api/docs/models/gpt-image-2>, 2026b.
- Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.
- Arjun Panickssery, Samuel R. Bowman, and Shi Feng. Llm evaluators recognize and favor their own generations. In *Advances in Neural Information Processing Systems*, 2024.

- Chaitanya Patel, Zhouyingcheng Liao, and Gerard Pons-Moll. Tailornet: Predicting clothing in 3d as a function of human pose, shape and garment style. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7365–7375, 2020.
- Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A A Osman, Dimitrios Tzionas, and Michael J Black. Expressive body capture: 3d hands, face, and body from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10975–10985, 2019.
- Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026. URL <https://qwen.ai/blog?id=qwen3.6-27b>.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019.
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763, 2021.
- Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, Zhaoyang Zeng, Hao Zhang, Feng Li, Jie Yang, Hongyang Li, Qing Jiang, and Lei Zhang. Grounded sam: Assembling open-world models for diverse visual tasks. *arXiv preprint arXiv:2401.14159*, 2024.
- Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover’s distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.
- István Sáránci and Gerard Pons-Moll. Neural localizer fields for continuous 3d human pose and shape estimation. In *Advances in Neural Information Processing Systems*, 2024.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.
- Gaurav Sharma, Wencheng Wu, and Edul N. Dalal. The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30(1):21–30, 2005.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.
- Zeng Tao, Ying Jiang, Yunuo Chen, Tianyi Xie, Huamin Wang, Yingnian Wu, Yin Yang, Abishek Sampath Kumar, Kenji Tashiro, and Chenfanfu Jiang. Dresswild: Feed-forward pose-agnostic garment sewing pattern generation from in-the-wild images. *arXiv preprint arXiv:2602.16502*, 2026.
- Wenbo Wang, Hsuan-I Ho, Chen Guo, Boxiang Rong, Artur Grigorev, Jie Song, Juan Jose Zarate, and Otmar Hilliges. 4d-dress: A 4d dataset of real-world human clothing with semantic annotations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.
- Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004.
- Fangsheng Weng, Junhao Chen, Xiang Li, Jie Qin, Hanzhong Guo, Shaochun Hao, and Xiaoguang Han. Garmentgpt: Compositional garment pattern generation via discrete latent tokenization. In *International Conference on Learning Representations (ICLR)*, 2026.
- Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In *Advances in Neural Information Processing Systems*, 2021.

- Wenda Xu, Guanglei Zhu, Xuandong Zhao, Liangming Pan, Lei Li, and William Yang Wang. Pride and prejudice: Llm amplifies self-bias in self-refinement. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2024.
- Shaofeng Yin, Jiaxin Ge, Zora Zhiruo Wang, Chenyang Wang, Xiuyu Li, Michael J. Black, Trevor Darrell, Angjoo Kanazawa, and Haiwen Feng. Vision-as-inverse-graphics agent via interleaved multimodal reasoning. *arXiv preprint arXiv:2601.11109*, 2026.
- Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.
- Cheng Zhang, Yuanhao Wang, Francisco Vicente Carrasco, Chenglei Wu, Jinlong Yang, Thabo Beeler, and Fernando De la Torre. FabricDiffusion: High-fidelity texture transfer for 3d garments generation from in-the-wild images. In *ACM SIGGRAPH Asia*, 2024.
- Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 586–595, 2018.
- Feng Zhou, Ruiyang Liu, Chen Liu, Gaofeng He, Yong-Lu Li, Xiaogang Jin, and Huamin Wang. Design2garmentcode: Turning design concepts to tangible garments through program synthesis. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 23712–23722, 2025.

CONTENTS

A	Notations	15
B	Garment Representation and Dataset	15
C	Stage 1: Layer Planning and Coarse Estimation	18
C.1	Agent implementation and interaction	18
C.2	Canonical A-pose views	19
C.3	Layer planning	21
C.4	Questionnaire synthesis	23
C.5	Template generation with the Visual Reader	25
C.6	Encoder-decoder from template to program	26
D	Stage 2: Fine-Grained Refinement	27
D.1	Agent implementation and interaction	27
D.2	Tool interface	28
D.3	Program editing with the Tailor	29
D.4	Size fitting with the Fit tool	30
D.5	Drape evaluation with the Critic	30
D.6	Session control with the Orchestrator	31
D.7	An example refinement round	31
E	Texture Generation	32
E.1	Texture routing	33
E.2	Tool implementation	33
F	Additional Experiments	34
F.1	Baseline implementation	34
F.2	Quantitative comparison on texture generation	35
F.3	Quantitative comparison on layer structure	35
F.4	Faithfulness of synthesized views	36
F.5	Stress test: refinement from a plain T-shirt	37
F.6	Additional visual results	38
G	Ablation Study	38
G.1	Ablation on synthesized views	38
G.2	Ablation on agentic trio design	39
G.3	Ablation on single-agent refinement	39
G.4	Ablation on texture generation	41
H	User Study	41
I	Limitations and Future Work	41

A NOTATIONS

Table 4 summarizes the main symbols used throughout this paper.

B GARMENT REPRESENTATION AND DATASET

Since garments can be written as code (Korosteleva & Sorkine-Hornung, 2023), our key insight is to solve inverse garment design the way agents already excel at coding: writing code and refining it against execution feedback, therefore turning the inverse garment design problem into a coding

Table 4: **Notations.** Summary of the main symbols and notations used in the paper.

Symbol	Meaning
<i>Garment Representation</i>	
$S = P(B)$	Sewing pattern of one garment: its flat panels and the stitches that join them, drafted by P on body B .
$P = \langle \mathcal{F}, \mathcal{D} \rangle$	Garment program: panel constructors with their design parameters.
$\mathcal{F} = \{f_i\}$	Panel constructors, one per component such as a sleeve or a bodice.
f_i	The i -th constructor, drafting a component as parametric 2D curves controlled by d_i and B .
$\mathcal{D} = \{d_i\}$	Design parameters, 131 in GarmentCode-XL.
d_i	The i -th design parameter.
B	Body measurements from the SMPL-X fit, used to draft every panel.
$M = \text{Drape}(P, B)$	Garment mesh draped by the cloth solver from P and B .
$\mathcal{P} = \langle (P_1, \dots, P_K), \mathcal{R} \rangle$	Outfit: an ordered stack of layer programs and their relations.
K	Number of layers, indexed from 1 (innermost) to K (outermost).
$\mathcal{R}$	Relations among pieces, such as tucked into or worn over.
P_k	Program of layer k , an optional upper piece with an optional lower piece.
$P_k^0, \mathcal{P}^0$	Coarse program of layer k from Stage 1, and the coarse outfit they assemble.
<i>Canonical Views and Layer Planning</i>	
I	In-the-wild input photograph, with arbitrary pose, background, and quality.
I^f, I^b	Canonical A-pose front and back views of the same person.
$g_{\text{canonical}}, g_{\text{back}}$	Image editing models that canonicalize I into the front view and generate its back view.
g_{peel}	Image editing tool that removes the garment described by r_k and completes what it covered.
r_k	Natural language description of layer k , written by the Layer Planner.
I_k^f, I_k^b	Front and back views in which layer k is the outermost garment.
π^f, π^b	Cameras that align a render of the drape with the two views.
<i>Template Generation</i>	
g_{ques}	Pre-trained LLM that reads the GarmentCode-XL source once and writes $\mathcal{Q}$.
$\mathcal{Q} = (q_1, \dots, q_{ \mathcal{D} })$	Questionnaire, one multiple choice question per design parameter.
q_i	Visual question about the garment property controlled by d_i , with a closed list of options.
τ_i	The option the Visual Reader selects for q_i .
GST_k	Garment Style Template of layer k , the answered questionnaire of that layer.
$\mathcal{V}(\tau_i)$	Normalized interval of parameter values covered by option τ_i .
<i>Program Decoding</i>	
E	Encoder, a VLM with distilled GarmentCode-XL semantics, embedding each question with its answer.
$\mathcal{X}$	Program and description pairs that distill GarmentCode-XL semantics into E .
Ψ	Autoregressive decoder that turns GST_k into the design parameters of P_k^0 .
$\mathbf{Q}$	Type-based quantizer of Eq. 7.
$t_i = \mathbf{Q}(d_i), \mathcal{T}$	Token of the i -th parameter, and the token sequence of one program.
$\text{Norm}(d_i), \text{Enum}(d_i)$	d_i mapped to $[0, 1]$ over its legal range, and the index of a categorical option.
$\lambda = 100$	Scale of the continuous quantizer, one step per percent of the legal range.
<i>Fine-Grained Refinement</i>	
$\Phi_{\text{tailor}}, \Phi_{\text{critic}}, \Phi_{\text{orch}}$	The agentic trio: the Tailor edits, the Critic judges, the Orchestrator controls.
t, T	Iteration index within one layer, and the simulation budget of that layer.
P^t	Program of the current layer at iteration t , starting from the coarse program P^0 .
M^t	Simulated mesh $\text{Drape}(P^t, B)$ at iteration t .
O^t	Observation of M^t : renders through π^f, π^b together with the measurements.
s^t	Per-garment scores on construction, proportion, fit, layering, and physical feasibility.
F^t	Discrepancies against the reference views, each with its measured evidence.
a^t	Action of the Orchestrator: Continue, Rollback, or Finish.
c^t	Instruction to the Tailor, naming the discrepancies to address in this round.
u^t	Edit of the Tailor, applied as $P^{t+1} = P^t \oplus u^t$.
$r_{\text{critic}}^t, r_{\text{orch}}^t, r_{\text{tailor}}^t$	Rationales the three agents attach to their outputs.
H^t	Evolving memory: every checkpoint with its pattern, observation, and critique.

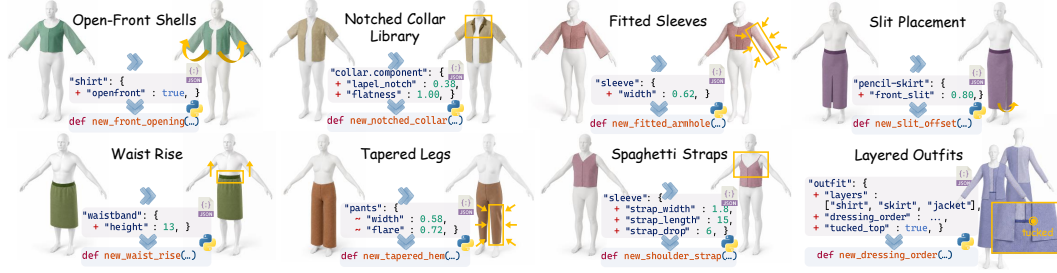


Figure 10: **Examples of Construction details of GarmentCode-XL.** New panel constructors and outfit fields are marked +, and expanded ranges of existing parameters are marked ~. The first seven widen the design space to real-world styles, and the last adds the layered outfit structure with dressing order and relations.

task. The ability of such an agent, however, is bounded by the expressiveness of its programming language. GarmentCode is the representative language, where a compact program compiles to a sewing pattern and its draped mesh, but it covers a limited set of garment styles and cannot represent multi-layer outfits with arbitrary structures and explicit relations. To unleash the potential of coding agents on real-world outfits, we therefore first introduce a more expressive language. GarmentCode-XL extends GarmentCode along two axes: a broader garment design space and a structured representation of layered outfits. Figure 10 shows examples of how we construct both extensions through new panel constructors, expanded parameter ranges, and outfit-level fields.

For layered outfits, GarmentCode-XL represents an outfit as $\mathcal{P} = \langle (P_1, \dots, P_K), \mathcal{R} \rangle$. Each P_k defines one dressing layer and may contain an upper piece, a lower piece, or both. The ordered sequence $(P_1, \dots, P_K)$ runs from the innermost layer to the outermost. The relation set $\mathcal{R}$ records typed relations within and across layers (e.g., tucked into, worn over). This structure also determines the simulation order: we drape the layers from inner to outer, and each settled layer becomes a collider for the next.

We use the outfit in Figure 11 as a concrete example. The listing below gives its GarmentCode-XL program. For clarity, we omit inactive branches, secondary parameters, default values, and schema annotations from the listing.



Figure 11: **Example of a two-layer GarmentCode-XL outfit.** The tee is *tucked into* the skirt within the inner layer L_1 . The cardigan forms the outer layer L_2 , with L_1 *worn under* L_2 .

Abridged GarmentCode-XL outfit program for Figure 11

```
{
  "outfit": {
    "layers": {
      "L1": {
        "program": "P1",
        "garments": ["white_tee", "black_skirt"]
      },
      "L2": {
        "program": "P2",
        "garments": ["beige_cardigan"]
      }
    },
    "relations": [
      {
        "within_layer": "white_tee tucked_into black_skirt",
        "between_layers": "L1 worn_under L2"
      }
    ]
  },
  "programs": {
    "P1": {
      "meta": {
        "upper": "Shirt", "wb": "StraightWB",
        "bottom": "PencilSkirt"
      },
      "shirt": {
        "length": 1.610, "openfront": false
      },
      "collar": {
        "f_collar": "CircleNeckHalf", "fc_depth": 0.402
      },
      "sleeve": {
        "length": 0.101, "width": 0.920
      }
    }
  }
}
```



```

    "waistband": {"width": 0.199},
    "pencil-skirt": {"length": 0.201, "rise": 1.000,
                    "flare": 0.996}
  },
  "P2": {
    "meta": {"upper": "Shirt", "wb": null, "bottom": null},
    "shirt": {"length": 1.610, "openfront": true},
    "collar": {"f_collar": "VNeckHalf", "fc_depth": 1.235},
    "sleeve": {"length": 1.150, "width": 0.600,
              "connecting_width": 0.887}
  }
}

```

Since no data exists for this new language, we build GarmentCodeData-XL, the first multi-layer sewing pattern dataset, for subsequent supervision for Stage 1 and benchmarking for different methods and models. We sample programs from the expanded design space, compose compatible garments into outfits of multiple layers, and discard invalid drapes. Each example contains the outfit program, sewing patterns, body measurements, rendered views, and simulated meshes. The dataset contains 15,205 outfits in total: 4,640 with one layer, 9,282 with two layers, 721 with three layers, and 562 with four layers. It provides both diverse garment designs and valid multi-layer compositions. We hope it serves as a common benchmark for multi-layer garment recovery and supports future research on layered clothing.

Simulation Extensions for Layered Outfits. The layered representation also requires an enhanced simulator. The original GarmentCode solver (Korosteleva & Sorkine-Hornung, 2023) drapes a single garment on a static body and does not model contact across layers. We keep its XPBD core (Macklin et al., 2016) on Warp (Macklin, 2022) and build an outfit-level pipeline on top of it that turns $\mathcal{P}$ into a dressing procedure. The pipeline first partitions the outfit into stitch-connected garments and schedules them by the layer sequence and the relations in $\mathcal{R}$. A worn-over garment is draped after every garment beneath it. A tucked top is draped before its bottom, and its cloth below the waistline is projected onto the body surface so that the bottom can close over it. Once a garment settles, it is frozen and registered as a thin-shell triangle-mesh collider for the layers above it, with a small collision standoff and the same soft frictional contact model as the body.

A layered drape is accepted only when every layer has settled into static equilibrium, clear of the body and free of self-intersection. The same dressing order extends from static draping to dynamic simulation. The body acts as a kinematic boundary condition, each cloth vertex is advected with its nearest body vertex on the same body part, and every settled inner layer is replayed as a moving collider for the layer above it, so the layered structure persists under motion.

C STAGE 1: LAYER PLANNING AND COARSE ESTIMATION

This section provides the implementation details of Stage 1 (Sec. 3.2), including the model of each agent, the prompt and skill design, the agent interaction, the structure of the Garment Style Template, and the encoder E and decoder Ψ .

C.1 AGENT IMPLEMENTATION AND INTERACTION

Agent Implementation. Stage 1 is a multi-agent workflow built on state-of-the-art foundation models, each equipped with carefully designed skills and tools:

- The Layer Planner and the Visual Reader are both GPT-5.6-luna (OpenAI, 2026a), a state-of-the-art multimodal reasoning model, run at medium reasoning effort. The Layer Planner reads the canonical front view I^f with a carefully designed reasoning skill, infers the layer structure $\langle K, \mathcal{R} \rangle$ with a description r_k per layer, and calls g_{peel} to expose each layer in its own view. The Visual Reader reads the two views I_k^f and I_k^b of one layer, answers the questionnaire of that layer, and returns its Garment Style Template GST_k .
- The image tools $g_{\text{canonical}}$, g_{back} , and g_{peel} are GPT-Image-2 (OpenAI, 2026b), a state-of-the-art image generation and editing model, wrapped in a strict-edit interface that pins identity, pose, and framing and returns one 1024×1024 view. $g_{\text{canonical}}$ turns the input photograph I into the

canonical A-pose front view, g_{back} turns any front view into the matching back view, and g_{peel} removes the layer described by r_k from I_k^f to give I_{k-1}^f .

- The questionnaire writer g_{ques} is GPT-5.6-luna at medium reasoning effort, run once offline (Appendix C.4). It reads the source code behind every design parameter and writes the questionnaire Q , which is frozen thereafter.
- The encoder E is Qwen3.6-27B (Qwen Team, 2026), an open-weight VLM of hidden width 5,120, into which we distill GarmentCode-XL semantics with LoRA (Appendix C.6). It reads a question with its selected option and returns the hidden state that conditions the decoder.
- The decoder Ψ is a 51.5M-parameter transformer that we design and train (Appendix C.6). It reads the 131 encoded answers of one layer and decodes the coarse program P_k^0 .

Agent Interaction. Here we summarize the data flow that ties the two agents and the encoder-decoder together. The Layer Planner opens the workflow by writing the outfit structure $\langle K, \mathcal{R} \rangle$ and one description r_k per garment. The relations $\mathcal{R}$ order the peel and r_k names the garment that g_{peel} removes at each step, which leaves every layer in a view where it is the outermost garment. We note that peeling stops at the innermost layer, which is already fully visible, so no view ever exposes the bare body. The same r_k then tells the Visual Reader which garment to read in that view.

The questionnaire Q is written once by g_{ques} , which turns every design parameter d_i into a question q_i with a closed option list, and the garment type and the asymmetry flag from the Layer Planner select the subset of Q that reaches the Visual Reader (Appendix C.5). The Visual Reader answers one option τ_i per question and fills the template GST_k , and E encodes the pair (q_i, τ_i) into the condition vector of the decoder Ψ , so a question and its answer share one embedding space. Ψ turns each GST_k into a program P_k^0 , and $\mathcal{R}$ from the Layer Planner assembles these programs into the coarse outfit $\mathcal{P}^0$ handed to Stage 2.

C.2 CANONICAL A-POSE VIEWS

$g_{\text{canonical}}$ canonicalizes the input photograph I into the A-pose front view I^f , and g_{back} turns that view into the matching back view I^b . Both edits change only the pose, the background, and keep the person and every garment as they are. Both views are locked to a 1024×1024 canvas with the head at the top edge and the feet at the bottom, so an analytic 180° orbit of π^f lands on the back view and no alignment network is needed. We show the prompt of $g_{\text{canonical}}$ below:

Prompt of $g_{\text{canonical}}$

You are given ONE image: an in-the-wild photo of a REAL person wearing an outfit.

Task: output a single clean, photorealistic FULL-BODY photograph of the SAME real person, re-posed into the canonical standing pose described below, standing still.

Keep exactly (the real subject -- do NOT turn them into a mannequin):

- THE PERSON: their true proportions, build and body shape, with natural skin.
- WHAT EACH GARMENT IS: type, color, print, fabric, neckline shape, sleeve length, waistband and hem position, overall length, silhouette, collars, cuffs, slits, pleats and seams.
- HOW IT IS WORN: the layer order and how each garment is fastened -- a zip stays open to the same height, an open front stays open by the same amount, buttons done up stay done up. Never close a garment the photo leaves open.
- WHAT SHOWS THROUGH: every inner garment stays visible wherever the photo shows it, in the same color and with the same neckline. Hiding an inner layer deletes a garment from the outfit.

Canonical standing pose (a strict catalog pose, NOT a relaxed fashion pose). The ARMS matter most and are most often rendered wrong, so read them first:

- Arms: both straight arms raised out to the sides and then lowered about a third of the way toward hanging, like a T-pose whose arms have dropped by a third. Each wrist ends level with the bottom of the ribcage, a hand's width below the chest and a hand's width above the natural waist, far out to the side, with a wide open triangle of white background between each arm and the torso. Both shoulders level, both elbows straight, matching left and right joint heights, neither hand touching the body. A narrow pose with the hands at the hips is wrong; a horizontal T-pose is also wrong, but if unsure, raise rather than lower.
- Upright torso facing the camera straight-on; legs straight and vertical, feet flat on the ground and slightly apart, not a wide fashion stance.
- Head upright and level: no tilt, no lean, no turn, eyes into the camera.
- Perfectly bilaterally symmetric, centered, and filling the frame head-to-toe.

- Studio: the background is PURE WHITE, exactly RGB 255,255,255 -- no tint, no gradient, no floor line, no shadow. Soft even light. No text, props or reflections.

Settle the cloth under gravity to a STATIC rest state: no wind, no motion, no billowing fabric. If the photo shows an action pose, settle every garment to how it would hang while the person stands still in this pose.

Remove every occluder so the garment is FULLY visible:

- Hair: keep the person's own length and style. Only if it falls over the neckline, shoulders, chest or straps, tie it back behind the shoulders.
- No bags, held objects, jewelry, scarves over the garment, or arms crossing the torso.
- If a garment is cropped or occluded, complete it conservatively as an opaque everyday garment consistent with the visible fabric, color and placement; do not invent extra layers or expose more of the body than the original outfit.
- Remove all footwear and redraw the feet as bare skin, so leg length is unambiguous.

Before you finish, check the arms once more: wrists level with the bottom of the ribcage and far out from the body, a wide white triangle under each armpit, elbows straight, matching joint heights. Check the feet too: redraw any remaining shoe or sock as bare feet.

Output image size: exactly 1024x1024 pixels -- a SQUARE 1:1 image. The person stands centered and spans the full height of the square, head near the top edge and feet near the bottom edge.

g_{back} then generates the back view from the front view, as if the camera orbits 180° around the person while the pose and framing stay fixed. The prompt asks for the same framing as the front view, so that the two views align and the back camera follows from the analytic orbit of π^f , and infers the back of each garment from its front. We show the prompt of g_{back} below:

Prompt of g_{back}

You are given ONE image: a clean FRONT catalog photo of a real person wearing an outfit, standing in the canonical pose with both straight arms raised out to the sides and lowered a third of the way toward hanging.

Task: output a single photorealistic FULL-BODY photograph of the SAME person, in the SAME pose, seen from DIRECTLY BEHIND -- the person has turned 180 degrees about their own vertical axis, and nothing else has changed.

FRAMING MUST BE IDENTICAL TO THE INPUT (this is critical):

- Exactly the same scale and position in frame: the top of the head and the soles of the feet sit at the SAME image height as in the input; the body is centered the same way and fills the frame identically.
- Preserve the pose exactly: shoulder, elbow, wrist and fingertip rows and their horizontal distances from the image center match the input. Both arms stay straight and symmetric on the same diagonals, wrists level with the bottom of the ribcage, the same wide white triangle beside the torso. Legs straight and close; standing still and barefoot.
- Studio: PURE WHITE background, exactly RGB 255,255,255, soft even light, no floor line or shadow, no text, props or reflections. Cloth settled to a static rest state under gravity.

BACK APPEARANCE (infer it from the front):

- The input defines the COMPLETE and EXACT garment set. Keep those garments and only those: identical types, layer order, colours, prints, fabrics, sleeve length, hem and waistband heights, overall silhouette.
- Render the BACK of each garment plausibly: back neckline, back closures (zips, buttons, ties), yokes, pleats, vents, slits, hood, and how the hem and sleeves read from behind.
- Where the back is genuinely ambiguous, render the simplest closed, opaque, symmetric back consistent with the front.
- HAIR: keep this person's own hair -- same length, colour and style. Only if it is long enough to fall over the back of the neck, gather it into a tight bun so that the back of the neck, the back neckline of every garment and both shoulders are bare of hair.

Output one photorealistic full-body REAR-VIEW image of this person in the same pose, on a plain studio background, framed identically to the input.

Output image size: exactly 1024x1024 pixels -- a SQUARE 1:1 image. The person stands centered and spans the full height of the square, head near the top edge and feet near the bottom edge.

C.3 LAYER PLANNING

The Layer Planner reads I^f with a carefully designed reasoning skill, infers the layer structure $\langle K, \mathcal{R} \rangle$, and writes a description r_k for each garment. We show the full skill below:

Reasoning skill of the Layer Planner

```
# Garment Layer Planner

You are given one canonical FRONT A-pose image. Identify every visible garment,
its coarse structure, and its inner-to-outer order. Use this image only. Do not
infer fine GarmentCode parameters. A later visual reader answers those questions
from the canonical front and back views.

## Garments

- One record per distinct top, skirt, pants, dress, or jumpsuit. Ignore shoes
  and accessories.
- A dress or jumpsuit is one garment. Call it separates only on visible
  evidence: a top hem, a waistband, a change of fabric at the waist.
- Report a garment when some part of it is visible, and only then. Inner layers
  show at the openings -- a collar above an outer garment's neckline, fabric
  between the edges of an open front, a hem or cuff emerging below or beyond
  the garment above. Read those places before concluding that an outer garment
  is worn against bare skin.
- 'name' is short, visible, and unique, such as 'brown bomber jacket'.
- 'type' is 'top | skirt | pants | dress | jumpsuit'. An outer coat, jacket,
  cardigan, or vest is still 'top'.

## Structure

For every garment report:

- 'closure': 'open' when its front edges visibly separate and expose the
  garment below, otherwise 'closed'.
- 'neck': 'covered' only for a turtleneck, mock neck, cowl, or raised hood.
  Use 'bare' for ordinary necklines.
- 'sleeves': 'none', 'short' above the elbow, or 'long' at or below the elbow.
  Use 'none' for lower-body garments.
- 'tucked_into': exact name of the lower garment whose waistband visibly
  covers this top's hem, so the top is tucked in. Null when the hem hangs
  over the waistband, and for every lower-body garment.
- 'asymmetric': true only when the two sides differ in the garment's overall cut:
  one sleeve present and the other absent, a one-shoulder neckline, a hem that
  sits high on one side and low on the other. Detail on one side is not a
  different cut: a single chest pocket, an off-centre print, a rip on one
  trouser leg.
- 'occluded': a short factual note about parts hidden by another garment.

## Focus

Every garment also carries a 'focus', alongside 'name', 'type' and 'structure'
rather than inside 'structure'. It is one or two sentences addressed to the
reader who will inspect that garment in the view where its layer is
outermost: every outer layer has been removed, while the garments of its own
layer and of the layers below stay on. Name the garment, say which of the
remaining garments are still in that view and must be ignored, and name the
parts of it the reader should judge. Write it from the image, not from a
template: a garment with nothing left beside it needs no warning about
distractors.

## Layer order

Write 'order.layers' from inner to outer. Each entry is one layer. A
layer is about covering the same part of the body, never about which edge
lies over which at the waist.

- Layer 1 holds every garment worn directly on the body: nothing under it
  covers the same part of the body. A top and a bottom worn together are both
  layer 1, whether the top's hem hangs over the waistband or is tucked into
  it. A dress or jumpsuit worn alone is layer 1.
- A garment belongs to an outer layer only when a garment beneath it covers
  the same part of the body and shows at its openings: a jacket, cardigan,
  vest or second top over a top or dress (torso over torso); a skirt over
  pants, or a dress over a top or pants (the inner garment continues under the
  outer one's fabric). A garment with nothing under it over that region goes
  to layer 1, however loose, long or coat-like it is.
- Where a top meets a bottom the two always overlap by a few centimetres,
```

- either hem over waistband or hem tucked under it. That overlap is recorded in 'tucked_into', and the two garments stay in the same layer. Putting the top in an outer layer because its hem covers the waistband is an error: the reader of the pants would then be shown a view without the top, which is a view of a bare torso that was never in the photograph.
- So every layer holds at most one upper-body garment and at most one lower-body garment, or one dress or jumpsuit, and every layer above 1 sits on a garment from a lower layer that covers the same region.

Examples: a sweatshirt hanging over trousers is `'[["sweatshirt", "trousers"]]'`. A tee tucked into a skirt under an open cardigan is `'[["tee", "skirt"], ["cardigan"]]'`. A turtleneck and leggings under a pinafore dress is `'[["turtleneck", "leggings"], ["pinafore dress"]]'`. A shirt and jeans with a skirt over the jeans and a coat over the shirt is `'[["shirt", "jeans"], ["skirt", "coat"]]'`. Every garment must appear exactly once.

The reader of each garment works on the view where that garment's layer is outermost, so 'focus' must describe exactly what is still on the body there: the garments in its own layer and every garment below it. Do not request a separate image-editing step.

Output

Reply with one 'layers.json' object following the format below. This is only a format example. Do not copy its names, colours, structures, or layer order. Fill every value from the input image. Return raw JSON without Markdown fences.

```
```json
{
 "garments": [
 {
 "name": "white tee",
 "type": "top",
 "structure": {
 "closure": "closed",
 "neck": "bare",
 "sleeves": "short",
 "tucked_into": "black skirt",
 "asymmetric": false,
 "occluded": "sides covered by cardigan"
 },
 "focus": "Read the white tee. The black skirt is the only other garment left in this view; ignore it and judge the tee's neckline, sleeve length and how far its hem reaches."
 },
 {
 "name": "black skirt",
 "type": "skirt",
 "structure": {
 "closure": "closed",
 "neck": "bare",
 "sleeves": "none",
 "tucked_into": null,
 "asymmetric": false,
 "occluded": "waist partly covered by cardigan"
 },
 "focus": "Read the black skirt. The white tee above it is the only other garment left; ignore it and judge the skirt's waist height, hem length and flare."
 },
 {
 "name": "beige cardigan",
 "type": "top",
 "structure": {
 "closure": "open",
 "neck": "bare",
 "sleeves": "long",
 "tucked_into": null,
 "asymmetric": false,
 "occluded": ""
 },
 "focus": "Read the beige cardigan. The white tee and black skirt stay visible underneath through its open front; ignore both and judge the cardigan's body length, sleeves, front opening and collar."
 }
],
 "order": {
 "direction": "inner_to_outer",
 "layers": [["white tee", "black skirt"], ["beige cardigan"]]
 }
}
```



```

 },
 "summary": "A white tee tucked into a black skirt under an open beige cardigan."
 }
 ...

```

**Schema validation.** The reply of the Layer Planner is checked against a schema before downstream reads it: the names are unique, the garment type and structural states come from their legal sets, every garment carries a focus note, and the layers hold every garment exactly once. A reply that fails any check goes back to the Layer Planner with the offending fields named, and the agent answers again. The Visual Reader is checked in the same way on its own answer sheet.

**Peel tool.**  $g_{\text{peel}}$  is an image editing tool that the Layer Planner calls once per layer from the outside in. One call takes the current front view and the garment to remove, and returns the same photograph with that garment taken off and the garments beneath it completed, so the layer below becomes the outermost. The peel stops at the innermost layer, so no view exposes the bare body. The call is a strict edit that preserves identity, pose, and framing, so  $I_{k-1}^f$  stays aligned with  $I_k^f$ . We show the tool call below, with the fields that the Layer Planner fills at call time marked in [this colour](#).

#### Tool call of $g_{\text{peel}}$

You are given ONE image: a photorealistic full-body photo of a REAL person in the canonical pose seen from the direct FRONT view on a plain studio background, wearing this outfit: [{TARGET GARMENT}](#), [{REMAINING GARMENTS}](#).

Task: produce a STRICT EDIT of this exact image showing the SAME fully clothed person wearing only the inner part of the same outfit: [{REMAINING GARMENTS}](#). In the edited photograph the [{TARGET GARMENT}](#) is simply not being worn, the way a catalogue shows the same look with the outer piece off. Everything else is pixel-identical. Edit the given pixels -- do NOT redraw the scene.

Keep EXACTLY -- do NOT alter:

- The person: same identity, face, skin tone, hair, body shape and proportions, pose, arm angle, position, camera framing and scale.
- The scene: PURE WHITE background, exactly RGB 255,255,255, soft even light, no floor line or shadow, no text, props or reflections.
- Every garment that stays on ([{REMAINING GARMENTS}](#)): keep their colour, length, neckline, sleeves, hem and silhouette exactly as they appear. If a top is tucked into a bottom, keep it tucked.

What changes ([{TARGET GARMENT}](#)):

- It is absent from the edited photograph, together with its sleeves, collars, lapels and waistbands.
- The garments it used to cover are now fully visible and continue normally across the region it occupied: [{WHAT THE TARGET GARMENT COVERS}](#)
- The person remains fully covered by those garments. No skin is uncovered anywhere the outfit did not already leave bare in the input.
- Invent nothing: no undershirt, no accessory, no shoes, socks or jewelry. Only what the list above names stays on.

Output a single photorealistic full-body image: the SAME real person in the SAME pose, camera viewpoint, framing, and studio scene, wearing [{REMAINING GARMENTS}](#).

Output image size: exactly 1024x1024 pixels -- a SQUARE 1:1 image. The person stands centered and spans the full height of the square, head near the top edge and feet near the bottom edge.

## C.4 QUESTIONNAIRE SYNTHESIS

Here, the biggest problem we met at first is semantic misalignment: a VLM prompted to write GarmentCode-XL programs directly cannot tell the meaning and bounds of a parameter from its name alone, and its programs drift far from the photograph. After many attempts, we found a surprisingly effective solution: let an LLM  $g_{\text{ques}}$  read the source code in advance and write one visual question per parameter for the agent to answer. For every design parameter,  $g_{\text{ques}}$  receives the source of the panel constructor that reads the parameter, the measurements of a mean body, and the fixed list of choice identifiers and numeric bins. Since the code reveals the geometric meaning of each parameter, we ask it to reason from the program but describe only what is visible: trace how the value shapes a panel, then state the result on the wearer in plain words, anchored to body landmarks such as the waist, hip, and knee. The resulting questionnaire  $Q$  is written once and frozen thereafter. We show the system prompt below.

**System prompt of  $g_{\text{ques}}$  for questionnaire synthesis**

You are the GarmentCode-XL questionnaire writer. GarmentCode-XL is a sewing-pattern DSL: each design parameter is read by a Python program that drafts panels for one body. For every parameter you receive, write one visual multiple-choice question that lets a vision-language model recover the parameter from a front and a back photograph of a person wearing the garment. The reader never sees code.

Reason from the program, write for the eye. Read how the program consumes the value: which panel edge or length it scales, what formula turns it into centimetres on the body, which other parameters gate it. Then describe only what that produces on the wearer, as visible geometry anchored to body landmarks (neck base, shoulder point, collarbone, bust line, underbust, natural waist, hip line, crotch, knee, ankle, elbow, wrist). When a choice covers a numeric bin, convert the bin through the program with the supplied body measurements, then state the visible result: where a hem lands, how far a seam stands off the body, how much wider one edge is than another, how deep an opening reaches. A reader with a photograph and no ruler must be able to pick the choice.

Write like a tailor describing the finished garment on a person. Never mention the program, drafting, panels, control points, coefficients, scales, internal lengths, variable names, percentages of internal quantities, or angles in degrees. Centimetres are allowed only for a distance a viewer can relate to a body part. A value with no visible effect of its own is described by its visible consequence.  
Bad: "The quadratic control point sits at 0.23 to 0.46 of the curve scale."  
Good: "The neckline bows below the straight shoulder-to-centre line by about a fifth of that line's length, a softly rounded V."

Rules:

- The answer space is fixed. Use every choice id exactly as given, in the given order, and nothing else. Numeric bins are already calibrated; describe them, never move them.
- `applies_if` states the branch of the program under which this parameter is read, in plain language naming the parent selection, style or flag.
- For booleans say what true and what false make the program build.
- For select parameters every option names a program construct; describe the visible geometry it drafts. Copy the label and description of every token in the glossary.
- Labels are short visual names. Never put a raw identifier into a label or description.
- Never answer the questions. Return JSON only: `{"questions":[{"id", "text", "visible_feature", "applies_if", "choices":[{"id","label","description"}]}]}`.

**Consistent visual semantics.** GarmentCode-XL exposes 30 categorical program tokens such as `SkirtCircle` and `Skirt2`, and 17 of them recur under multiple parameters, so questions written independently may describe the same token differently. We therefore first ask  $g_{\text{ques}}$  to assign every token one visual label and description, and reuse this glossary wherever the token appears.

**Question construction and validation.** With this glossary in place,  $g_{\text{ques}}$  writes one question for each of the 131 design parameters. It uses the complete parameter hierarchy to determine when each question applies. Each record contains a question, its visible feature, the condition under which it applies, and a closed set of choices. A discrete parameter covers every legal value exactly once, while a numeric parameter uses 3 to 6 ordered visual states instead of raw values. We validate identifiers, ordering, coverage, and visual labels against the schema, and regenerate any failing record. Together, these records form a fixed questionnaire  $Q$  whose choices stay consistent across parameters and refer only to observable garment geometry. Table 5 summarizes this answer space, synthesized once and frozen, so no language model writes questions at inference time.

Table 5: **The frozen answer space.** Here, every design parameter of GarmentCode-XL becomes exactly one multiple choice question.

Parameter type	Count	Options	Answer format
Length (float)	83	308	3 to 6 ordered visual states
Count (int)	15	55	ordered states, or one option per admissible count
Boolean	14	28	the two geometric states of the flag
Categorical	11	56	one option per legal value
Categorical with absence	8	34	one option per legal value, plus absence
Total	131	481	

**Mapping visual choices to parameter intervals.** Since the Visual Reader answers multiple choice questions more reliably than numerical ones, every numeric question offers discrete visual choices, and each choice  $\tau_i$  maps to an interval  $\mathcal{V}(\tau_i) \subseteq [0, 1]$  of the normalized parameter range. The

intervals of one question partition  $[0, 1]$ , so every parameter value belongs to exactly one choice, and their boundaries sit at body landmarks rather than uniform splits of the legal range. The choice read from the image and the interval the decoder is trained on therefore refer to the same garment geometry. The shirt-length question below shows how six choices cover the full range.

#### Example: shirt-length choices and parameter intervals

```
id: design.shirt.length (float, range [0.5, 3.5], default 1.2)
text: How long is the visible shirt, judged by where its hem ends on the
 body (above the waist, at the waist, on the hip, on the thigh, at the
 knee, or below the knee)?
visible feature: shirt length
applies if: when the shirt is enabled

::cropped [0.00, 0.14) "Cropped"
 The hem stops above the natural waist and leaves the midriff bare, anywhere from
 a crop just above the waistband to a very short crop up at the ribcage.
::waist [0.14, 0.28) "Waist length"
 The hem ends at the natural waist or just onto the high hip, about where a
 trouser waistband sits, like a typical waist-length top.
::hip [0.28, 0.40) "Hip length"
 The hem covers the hip and seat completely, ending at or just past the hip
 crease, like a standard T-shirt.
::thigh [0.40, 0.60) "Mid-thigh"
 The hem falls onto the thigh, below the seat but clearly above the knee, a tunic
 or short-dress length.
::knee [0.60, 0.78) "Knee length"
 The hem ends at the knee, just above or just below the kneecap.
::calf [0.78, 1.00] "Below the knee"
 The hem falls past the knee to mid-calf or as far as the ankle, a long dress
 length.
```

## C.5 TEMPLATE GENERATION WITH THE VISUAL READER

The Visual Reader answers the questionnaire for each layer from its views  $I_k^f$  and  $I_k^b$  and description  $r_k$ , and its answers form the Garment Style Template  $GST_k$ . Rather than asking all 131 questions for every garment, we use its type and asymmetry state to select only the compatible branches of  $\mathcal{Q}$ . Garments in the same layer share one multimodal call but keep separate answer sheets. The reader resolves construction choices first, and uses their activation conditions to skip questions about geometry the garment does not have. We show the prompt below:

#### Prompt of the Visual Reader

```
You are the Visual Reader. Image 1 is the FRONT view and Image 2 is the BACK view.
The current layer contains these target garments: {TARGET GARMENT NAMES}.
Read every target in this request, but keep a separate answer sheet for each. Use each
target's focus text and questionnaire. Other garments may provide wearing context, but
never use one garment as visual evidence for another.

Question ids are template paths named for the program that constructs the garment, not
for its everyday category. For example, design.shirt.length refers to the body length
of the current upper garment, including a cardigan, jacket, or vest.

Resolve construction questions first, then follow each applies_if condition. For every
active question, choose exactly one listed choice id. Match its geometric description
against the two images and body landmarks. When a region is occluded, infer its full
extent only from the target's construction and visible cues. Use "not applicable" when
the condition is false and "not visible" when an active feature cannot be judged.

Return JSON only:
{"garments": [{ "index": ..., "name": ..., "answers": [
 { "id": ..., "answer": ..., "confidence": "high|medium|low", "evidence": ... }
]}]}.
Keep garment identity, garment order, question ids, and question order unchanged.
Emit every requested row, including inactive rows as "not applicable".

{ONE FOCUS NOTE AND PRUNED QUESTIONNAIRE PER TARGET GARMENT}
```

**Garment Style Template formation.** The selected options form  $GST_k$ . Questions that are inactive or not visible are masked, so the decoder  $\Psi$  receives only supported answers. Table 6 lists the number of questions retained for each garment type and symmetry state.

Table 6: **Layer-specific questionnaire size.** Number of questions retained from  $\mathcal{Q}$  for each garment type and symmetry state.

Garment type	Symmetric		Asymmetric	
	Questions	Numeric	Questions	Numeric
Top	48	34	79	56
Skirt	42	33	42	33
Pants	15	12	15	12
Dress	90	67	121	89
Jumpsuit	63	46	94	68

## C.6 ENCODER-DECODER FROM TEMPLATE TO PROGRAM

The encoder-decoder maps each Garment Style Template back to an executable program. The encoder  $E$ , with distilled GarmentCode-XL semantics, embeds every answered question, and the decoder  $\Psi$  turns these embeddings into the design parameters of  $P_k^0$ .

**Encoder  $E$ .** The encoder  $E$  reads each question with its selected option and returns a hidden state that conditions the decoder. To interpret an answer in terms of the program parameters,  $E$  needs the semantics of every GarmentCode-XL program, which requires fine-tuning. We therefore build  $E$  on the open-weight Qwen3.6-27B (Qwen Team, 2026) and distill the program semantics into it from  $g_{\text{ques}}$ , which has already read the source code. Here  $g_{\text{ques}}$  takes a second pass over the source with the opposite goal: the questionnaire turns each parameter into visible geometry for the Visual Reader, while the descriptions explain each program unit for the encoder. Pairing every description with its source program gives the training set  $\mathcal{X}$  of 417 units at two levels of the program hierarchy: 122 component classes and top-level programs that specify complete panel construction and assembly, and 295 methods that implement local drafting operations such as constructing curves, transforming edges, placing darts, and joining sewing interfaces. Together they cover all 131 design parameters and connect each parameter to the geometry it controls. We show the prompt below.

### Prompt of $g_{\text{ques}}$ for program description synthesis

You are preparing Program Learning data for GarmentCode-XL. Read the given implementation and explain it as detailed natural-language pattern-drafting instructions at the implementation’s own level of abstraction. For garment components, explain panels, geometry, interfaces, stitches, assembly, and the visual effect and direction of every design parameter used. For geometric primitives, explain the exact drafting operation and how later components use its result. Resolve abstract component names from their actual geometry and assembly logic. Use only facts supported by the code. Do not reproduce code, invent missing behavior, or merely paraphrase identifiers. Write at least 60 words so that the description-to-program pair contains enough semantic context.

User: Comment `{module}.{symbol}`. Return JSON only as {"instruction": "..."}.

```
```python
{source code of the program unit}
```
```

**Distillation targets.** We fine-tune  $E$  on two kinds of pairs. The first maps each description to its source program, so  $E$  learns how language translates into drafting code. The second maps the program excerpt that reads each design parameter to the question written for it, so  $E$  learns the geometry behind each question.  $E$  then encodes a question with its selected answer into a vector reflecting the program parameters.

**LoRA setting.** We apply LoRA with rank 16 and  $\alpha = 32$  to every linear projection in the attention and feed-forward blocks, and keep the vision encoder frozen. We use sequence length 8,192, per-device batch size 1, and gradient accumulation 8. The first two epochs use a cosine schedule with a peak learning rate of  $5 \times 10^{-4}$  and 3% warmup, then a tenfold lower peak rate with evaluation every quarter epoch. The full training takes within 1 hour on two NVIDIA RTX PRO 6000 GPUs.

**Decoder  $\Psi$ .** Each numeric question offers visual ranges rather than raw values, and the template records the selected range. The decoder  $\Psi$  maps it back to a precise value that the program can execute: it keeps the 33 discrete choices fixed by the template and autoregressively decodes the 98 numeric parameters. A projection MLP maps the 5,120-dimensional condition vectors of  $E$  to the decoder width of 512, and a causal Transformer decodes the parameter tokens (Fig. 12).

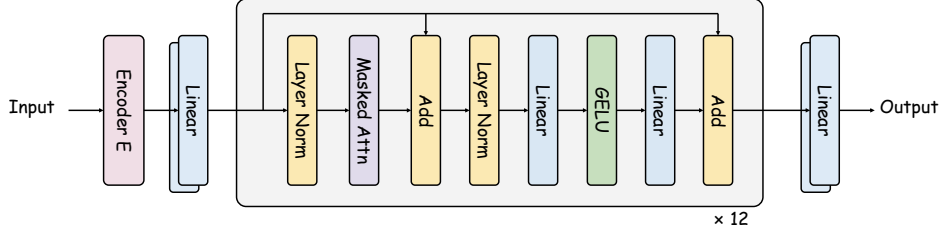


Figure 12: **Architecture of the decoder  $\Psi$ .** The frozen encoder  $E$  embeds each question with its selected answer. A projection MLP maps the embeddings to the decoder width, and a causal Transformer with 12 blocks decodes the numeric program parameters.

**Parameter tokenization.** We represent the  $|\mathcal{D}| = 131$  (total number of design parameters in GarmentCode-XL) design parameters as an ordered token sequence  $\mathcal{T} = (t_1, \dots, t_{|\mathcal{D}|})$ , where the quantizer  $\mathbf{Q}$  respects the type of each parameter:

$$t_i = \mathbf{Q}(d_i) = \begin{cases} 0 \text{ or } 1, & \text{if } d_i \text{ is a boolean, such as a strapless top,} \\ d_i, & \text{if } d_i \text{ is a count, such as the number of panels,} \\ \text{round}(\lambda \cdot \text{Norm}(d_i)), & \text{if } d_i \text{ is a length, normalized to } [0, 1] \text{ over its range,} \\ \text{Enum}(d_i), & \text{if } d_i \text{ is a categorical choice, selected among its options.} \end{cases} \quad (7)$$

We use  $\lambda = 100$ , which keeps centimeter-level precision for every length.

**Training details.** We train  $\Psi$  on 232,500 garment programs from the official training split of GarmentCodeData-v2 (205,487) and the training split of GarmentCodeData-XL (27,013), and validate on the official validation split of GarmentCodeData-v2 (13,195). The GarmentCodeData-XL is split at the outfit level, so no outfit appears on both sides.  $\Psi$  has 51.5M trainable parameters and is trained for 12 epochs with batch size 256 using AdamW, a peak learning rate of  $5 \times 10^{-4}$ , weight decay 0.05, and cosine decay with 5% warmup. We use dropout 0.08, gradient clipping at 1.0, and cross-entropy over the 98 numeric tokens with teacher forcing. The training takes within 2 hours on two NVIDIA RTX PRO 6000 GPUs.

## D STAGE 2: FINE-GRAINED REFINEMENT

This section provides the implementation details of Stage 2 (Sec. 3.3), including the implementation of the agentic trio, the complete tool interface, the design of observations and measurements, the `Fit` tool for size fitting, and the full prompt of each agent.

### D.1 AGENT IMPLEMENTATION AND INTERACTION

**Agent Implementation.** All three agents are GPT-5.6-luna (OpenAI, 2026a) at medium reasoning effort. Each agent has its own system prompt, its own tool set, and a persistent memory that spans the whole session of one layer. As in Sec. 3.3, the agents interact with the garment exclusively through structured tools, which are backed by a shared session that drafts and simulates programs, keeps every simulated step as a checkpoint together with its critique, tracks the simulation budget, and composes the observations of Appendix D.2. Each layer opens with a budget of  $T$  simulations (we set  $T=6$  by default).

**Agent Interaction.** The session opens by draping the coarse program  $P^0$  of Stage 1 as step 0. After every simulation, the Orchestrator invokes the Critic as a tool, and stores its scores  $s^t$  and discrepancies  $F^t$  with the program and observation of that step as a checkpoint in the history  $H^t$ . The Orchestrator then reads the history of all steps and takes one of its three actions: it continues

with an instruction to the Tailor, rolls the program back to an earlier checkpoint, or finishes the layer. Within its round, the Tailor edits freely with the observation and modification tools and ends the round with one `Simulate` call, or returns with `Done` when no remaining discrepancy is worth a simulation.

**Session Memory.** Each agent keeps its own message history across rounds. The views of the two most recent rounds stay inline, older views are compacted into text notes, and every round restates the history of steps, so the agents keep the whole trajectory in context at a bounded size. `Rollback` restores the program only and leaves the history and each agent’s memory intact for reference, which realizes the evolving memory of Sec. 3.3.

## D.2 TOOL INTERFACE

Table 7 lists the complete tool interface of Stage 2. Tool access is role-dependent: the Tailor has access to observation and modification tools, the Critic is restricted to observation tools, and the Orchestrator invokes the Tailor and Critic as tools while managing global control operations such as rollback. We note that only `Simulate` spends the budget, and every other tool is free, so the Tailor can explore the pattern space extensively between two simulations.

Table 7: **The complete tool interface of Stage 2.** Access: T = Tailor, C = Critic, O = Orchestrator.

| Tool                | Access | Arguments                               | Description                                                                                                                                                               |
|---------------------|--------|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Observation</i>  |        |                                         |                                                                                                                                                                           |
| Render              | T      | pictures                                | Views of any step: the photograph, the target and drape pair of either view, the before/after strip, per-garment mask diffs, and the flat sewing pattern.                 |
| Measure             | T      | step                                    | Measurement tables of one simulated step: every garment edge on the drape against the target, and the same sizes read off the drafted pattern, in percent of body height. |
| <i>Modification</i> |        |                                         |                                                                                                                                                                           |
| Draft               | T      | edits                                   | Drafts the current program with the edits and returns the garment sizes of the flat pattern, with the change against the current step.                                    |
| Fit                 | T      | part, channel, lever, target            | Value of one numeric lever that lands one measured size on its target, found through drafting probes (Appendix D.4). Costs no simulation.                                 |
| Simulate            | T      | edits, note                             | Drapes the current program with the edits applied, one call per round; ends the round.                                                                                    |
| <i>Control</i>      |        |                                         |                                                                                                                                                                           |
| Scoring             | C      | scores, discrepancies, comparison, done | The Critic’s structured judgement of one step (Appendix D.5).                                                                                                             |
| Done                | T      | note                                    | Returns the round to the Orchestrator when no remaining discrepancy is worth a simulation.                                                                                |
| Critic              | O      | step                                    | Invokes the Critic on a simulated step; called after every simulation.                                                                                                    |
| Tailor              | O      | instruction                             | Continues from the current step; the Tailor answers with one simulation.                                                                                                  |
| Rollback            | O      | step, instruction                       | Rolls the program back to an earlier checkpoint and keeps every memory.                                                                                                   |
| Finish              | O      | step, reason                            | Ends the layer on one step; a step with cloth through the body is refused if a valid step exists.                                                                         |



**Observation Design.** Every simulated step is observed in three complementary forms. The paired views place our render beside the canonical target on one canvas, cropped identically and overlaid with the same body landmark lines, so a hem height is read on the same ruler in both images. A before/after strip adds the parent step, and per-garment mask diffs mark in green the target cloth our drape lacks and in red our cloth beyond the target. The tables measure eight channels per garment on both views in percent of body height: four positions (top edge, bottom edge, sleeve end, neck depth) and four extents (width, hem width, sleeve width, neck width), each reported as ours, target and gap, beside the same sizes read off the drafted pattern. Finally, the solver’s report is attributed per garment (e.g., cloth through the body, self-intersecting edges) together with the edits of the step that touched that garment, so each physical issue is traced to the parameters that caused it.

### D.3 PROGRAM EDITING WITH THE TAILOR

In each round, the Tailor receives comprehensive context for its edit. This includes the Orchestrator’s instruction  $c^t$  and the critique  $F^t$  of the current step, the solver’s report, the measurement tables, the history  $H^t$  with the measured effect of every past edit including those rolled back, and the list of active parameters. This list gives every parameter its legal range and the visual question that  $g_{\text{ques}}$  wrote for it in Stage 1, with the visual meaning of every numeric interval and categorical option, so the same semantics that ground the Visual Reader also define the action space of the Tailor. As in Sec. 3.3, the Tailor makes two kinds of edits: a construction edit switches a categorical choice such as the neckline or the garment family, and a size edit tunes the numeric parameters to match the reference. We show the prompt of the Tailor below.

#### Prompt of the Tailor $\Phi_{\text{tailor}}$

You are the Tailor in a fitting session. A GarmentCode sewing program drafts the panels of the garments in this layer, a cloth simulator drapes them on the recovered body, a Critic compares the drape with the photograph and says what differs, and an Orchestrator tells you what to work on and keeps every simulated step as a checkpoint. You change the program.

```
This layer
{LAYER CONTEXT}
```

## How the session runs

Each round you receive the Orchestrator’s instruction, the Critic’s critique of the current step, the measurements of the current step, and pictures. You end the round with one ‘Simulate’ call, or with ‘Done’ when no remaining discrepancy is worth a simulation. Simulations are the scarce resource and drafting is free: use ‘Draft’ and ‘Fit’ as often as you like before you simulate.

There are two kinds of edit. Construction is categorical: the garment family, the neckline, the sleeves, the collar, the closure. It decides what the garment is, so settle construction first. Size is numeric: where a hem falls, how wide a sleeve is, how deep the neck opens. Reach sizes with ‘Fit’, which reads the drafted pattern and returns the parameter value that lands the size on its target. The answer also names the interval its value lands in; a fit that crosses into another interval changes the shape, so take it only when the photograph shows that shape.

The solver’s report names the garment it objects to. Cloth through the body cannot be delivered, so clearing it comes before any other improvement, through that garment’s own parameters.

Measurements are evidence, not verdicts: they are read off masks of the segmented photograph and the rendered drape on one body ruler, and a reading can be wrong where an edge is hidden or the back view is generated. When a number and the pictures disagree, say which you trust and why.

Judge your last step before acting on the next: the before/after picture and the Critic’s comparison say whether it helped. Do not repeat an edit that did not move what it was meant to move.

```
GarmentCode parameters
Each entry gives the path, its legal values, and the visual question that defines its meaning. Only these parameters are active in this layer. Switching the garment family swaps which parameters exist from the next round.

{ACTIVE PARAMETERS WITH THEIR VISUAL QUESTIONS}
```

#### D.4 SIZE FITTING WITH THE FIT TOOL

A key design of our refinement is that the agents reason with measurements: the Critic reports each discrepancy as a measured gap in percent of body height, and the Tailor turns that gap into a parameter value. The second step is not obvious, since a size edit asks for a hem a few percent lower or a sleeve a third wider, while GarmentCode-XL parameters are fractions of construction quantities. We therefore design the `Fit` tool, which infers the parameter value from the history and the measurements of the layer.

Given a target size on the drape, the tool first estimates the relation between drafted and draped sizes from the simulated steps in the history. It then maps the target into draft space, and searches for the parameter value with a few drafting probes until the drafted size lands on the goal. In this way, numeric measurements give each edit a precise target, while the visual intervals and the rendered views keep the shape faithful to the photograph, and the agents judge with both.

#### D.5 DRAPE EVALUATION WITH THE CRITIC

The Critic returns its judgement through the `Scoring` tool in a fixed schema. The scores  $s^t$  give each garment of the layer five integers from 0 to 10, one per dimension (construction, proportion, fit, layering, and physical feasibility). The discrepancies  $F^t$  are listed largest first, at most six, and each names the garment, the difference against the target, the place on the body, the measured channel when the table has one, and a severity. The measured value of every named channel is then attached, so a critique reaches the Tailor with its metric evidence in place. The rationale  $r_{\text{critic}}^t$  closes with a comparison against the previous step (better, worse, or similar) and a done flag that ends the layer when no remaining difference would visibly improve with another simulation. The construction vocabulary given to the Critic reuses the categorical questions of Stage 1, and the current program is summarized in the same words, so the Critic compares the photograph and our construction in one vocabulary. We show the prompt of the Critic below.

##### Prompt of the Critic $\Phi_{\text{critic}}$

You are the Critic in a fitting session. A Tailor edits a GarmentCode sewing program, a cloth simulator drapes it on the recovered body, and you compare the drape with the photograph. You never edit. Your judgement decides what the Tailor works on next and whether a step is kept.

## This layer  
{LAYER CONTEXT}

## What you see  
The photograph; our render of the current step beside the canonical target view, with the same landmark lines and body ruler on both; a before/after strip against the previous step; per-garment mask diffs (green is target cloth ours lacks, red is our cloth beyond the target); and a table of measurements in percent of body height. Compare geometry: what each garment is, where its edges fall, how much room it has, how it hangs. Ignore colour, print, texture, lighting and the seam lines the simulator draws between panels.

The numbers are evidence with known failure modes: the masks come from a segmenter, the back view is generated, and an edge hidden under another garment is not an edge of that garment. Read them beside the pictures and say when they disagree.

## What you report  
Score each garment of this layer on five dimensions from 0 to 10, each on its own: construction (the same kind of garment, neckline, sleeves, collar and closure), proportion (where its edges fall on the body), fit (how much room it has and the shape it makes), layering (how it sits with the garments underneath), and physical feasibility (whether the cloth hangs as the garment would; cloth through the body is a failure). Then list the discrepancies worth a simulation, largest first, each naming the garment, what the target has against what ours has, and where you see it, with the measured channel when the table has one. Say whether this step is better or worse than the previous one, and whether the fit is done: done means no remaining difference would visibly improve with another simulation.

A construction difference counts only when the vocabulary below can build the target's construction. Finishing details (ribbing, topstitching, buttons, pockets, prints) are not construction.

## Construction vocabulary in this layer  
{CONSTRUCTION QUESTIONS AND OPTIONS}

## D.6 SESSION CONTROL WITH THE ORCHESTRATOR

The Orchestrator controls the refinement flow of a layer. After each critique, it reads the history  $H^t$  and the critique  $(s^t, F^t)$ , and takes one of three actions: *Continue* with an instruction  $c^t$  that tells the Tailor which discrepancies to address, *Rollback* to an earlier checkpoint when the current step is judged worse, or *Finish* on the best valid checkpoint. We show the prompt of the Orchestrator.

### Prompt of the Orchestrator $\Phi_{\text{Orch}}$

You are the Orchestrator of a fitting session. A Tailor edits a sewing program, a simulator drapes it, and a Critic scores the drape against the photograph. After each critique you decide whether the session continues from the current step, returns to an earlier checkpoint, or ends, and you tell the Tailor what to work on.

```
This layer
{LAYER CONTEXT}
```

Every simulated step is a checkpoint with its critique. A step with cloth through the body cannot be delivered: clearing it comes first in your instruction, and such a checkpoint is not a place to finish. Returning to a checkpoint costs nothing and keeps the Tailor's memory of what was tried, so a step the Critic judged worse is rolled back rather than built on.

Simulations are the budget: a round that would not change what the Critic sees is not worth one, but while the best checkpoint still carries a major discrepancy, the session continues from that checkpoint with a more careful instruction rather than finishing early. End the session when the Critic finds the fit done, when the remaining differences are beyond the program's vocabulary, or when the budget is out, and finish on the step whose garments together score best among those without cloth through the body.

Your instruction to the Tailor is short: which discrepancies to address this round, in what order, and what to leave alone.

## D.7 AN EXAMPLE REFINEMENT ROUND

We show the complete record of one refinement round, from Iter 1 to Iter 2 of the inner layer in Fig. 4. Fig. 13 shows the observation  $O^t$  given to the agents before and after the round: the rendered drape beside the reference view, with the same landmark lines and body ruler. At step 1, the tee is still too wide, too deep at the neck, and too long in the sleeve. The Critic returns the critique below.

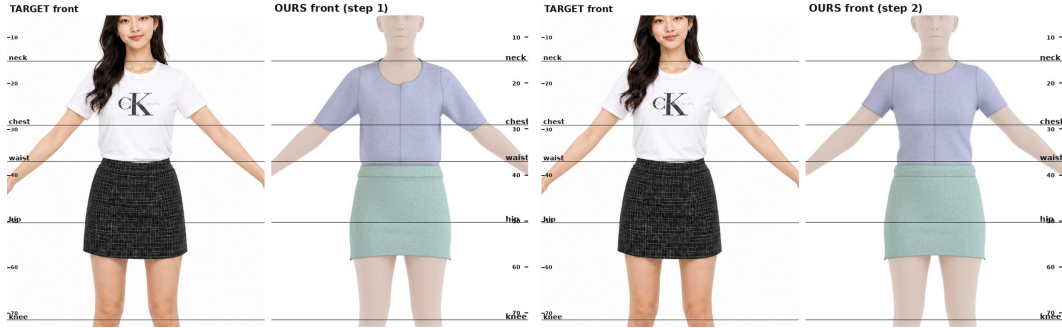


Figure 13: **The observation  $O^t$  given to the agents in the recorded round.** *Left:* the paired front view of step 1 (Iter 1 of the inner layer in Fig. 4), which the Critic and the Tailor read. The skirt has been corrected, and the tee is still wide, deep at the neck, and long in the sleeve. *Right:* the same observation at step 2, after the five fitted edits below. Both panels carry the same landmark lines and body ruler in percent of body height.

### Critique of step 1, returned by the Critic through Scoring

scores upper: construction 9, proportion 6, fit 6, layering 9, feasibility 8  
lower: construction 8, proportion 9, fit 8, layering 9, feasibility 9  
against the previous step: better. "The skirt edit is a clear improvement: its front and back hems now align with the target within measurement noise and its front hem width is essentially matched. The upper garment is unchanged, so its broad torso, deep neckline, and long sleeves remain."  
discrepancies, largest first:

1. [upper, major, width] The target tee is closer and slimmer through the torso, while ours remains visibly wider. Front and back bodice; ours is about 2.4-3.2 cm wider.
2. [upper, minor, hem\_width] The target tee has a narrower lower body, while ours flares wider at the tucked-in lower edge. Front gap +2.2 %bh, about 3.7 cm, though this edge is partly affected by the tuck.
3. [upper, minor, neck\_depth] The target front neckline is shallower, while ours drops lower. Ours is about 2.9 cm deeper; the back comparison is less reliable and shows the opposite direction.
4. [upper, minor, sleeve\_end] The target sleeves finish higher, while ours extend farther down the upper arms. Both sleeve openings; ours is about 2.7-3.4 cm lower.
5. [upper, minor, bottom\_edge] The target visible bodice ends slightly higher, while ours extends lower into the tucked region.
6. [lower, minor, hem\_width] The target back hem is slightly fuller, while ours remains a little narrower: 0.8 %bh, about 1.4 cm. Front hem width is within noise.

summary: "Keep this step: the skirt shortening and flare increase successfully corrected the dominant lower-garment error, with excellent front alignment and only a small back-hem width residual. The remaining visible improvement is in the unchanged tee: reduce its torso width, then address the deeper front neck and overly long sleeves; the fit is not done."

The Orchestrator reads the history and this critique, keeps the checkpoint, and continues with the instruction below.

#### Instruction of the Orchestrator, passed through Tailor

Keep the skirt unchanged, including its successful hem length and flare. First narrow the tee through the torso and especially the visible tucked lower body; then, in the same edit if stable, raise/shallow the front neckline and shorten both sleeves. Leave the skirt's minor back-hem residual alone.

The Tailor then calls the Fit tool once per requested size to solve for the parameter values. For the sleeve end, both sleeve parameters hit their range limit before reaching the target, so the Tailor sets them to the limit and reports the remaining gap in its note.

#### Round of the Tailor: five Fit calls and one Simulate

```
Fit(upper, width, shirt.width, target 15.9)
-> value 1.0484; probes 1.176 -> 27.9, 1.296 -> 30.7, 1.0484 -> 24.9 %bh
 drafted; predicted 15.9 %bh on the drape
Fit(upper, hem_width, shirt.flare, target 15.3)
-> value 0.8617; probes 0.997 -> 27.9, 1.132 -> 31.4, 0.8617 -> 24.4;
 predicted 15.3
Fit(upper, neck_depth, collar.fc_depth, target 17.9)
-> value 0.2786; probes 0.400 -> 23.2, 0.700 -> 27.4, 0.2786 -> 21.5;
 predicted 17.9
Fit(upper, sleeve_end, sleeve.length, target 29.2)
-> value 0.1; "sleeve.length reaches its range limit at 0.1 with
 upper.sleeve_end at 30.3 %bh against a target of 29.2; another parameter
 is needed for the rest."
Fit(upper, sleeve_end, sleeve.connecting_width, target 29.2)
-> value 0.0; "sleeve.connecting_width reaches its range limit at 0 with
 upper.sleeve_end at 29.9 %bh against a target of 29.2; another parameter
 is needed for the rest."
Simulate(shirt.width 1.0484, shirt.flare 0.8617, collar.fc_depth 0.2786,
 sleeve.length 0.1, sleeve.connecting_width 0.0)
note: "Narrow the tee torso and its tucked visible lower body, shallow the
 front neckline to the target, and use the shortest sleeve plus the minimum
 armhole drop to raise the sleeve ends as far as the sleeve parameters
 allow; leave the corrected skirt unchanged."
```

After simulation, the drape matches the sizes that Fit predicted (Fig. 13, right): the front silhouette IoU rises from 0.851 to 0.937, and the Critic judges step 2 better, raising the tee from 6 to 8 on proportion and fit, with the neckline width as the main remaining discrepancy.

## E TEXTURE GENERATION

This section provides the implementation details of texture generation (Sec. 3.4). As in the main text, we first segment each garment with Grounded SAM and map its pixels onto the sewing pattern through the fitted cameras, which gives a partial texture where observed texels hold the colors of the photograph. However, direct UV mapping introduces boundary artifacts, since the observation



Figure 14: **Examples of the five texture classes.** From left to right: a solid fabric, a knit without an exact repeat, placed artwork over a base fabric, an exact repeating lattice, and a non-repeating print, each handled by the tool named below it.

is sparse and part of the cloth is occluded by other garments or folds. Currently, garment texture extrapolation is still an open problem due to the diversity of garment textures. We therefore design a novel agentic generation method: we divide garment textures into five classes (Fig. 14), design a texture extrapolation tool for each, and let the Layer Planner classify each garment and call the corresponding tool. Below we describe the routing and the implementation of each tool.

### E.1 TEXTURE ROUTING

We describe the five texture classes to the Layer Planner and let it classify each garment from a crop of the observation, with other garments masked out. It returns the tool to call, a description of the print, and a box for each piece of placed artwork. We show the prompt below.

#### Texture routing prompt of the Layer Planner

You are routing ONE garment’s texture to one of five tools.  
The image is a crop centered on that garment. Everything that is not the garment -- other garments, skin, background -- is painted flat gray. Judge ONLY the unpainted pixels.

Measured evidence:

- per-channel color spread of the observed pixels: {COLOR STATISTICS}
- repeating-pattern autocorrelation: {REPEAT MEASUREMENT}

Tools:

- PickColor: one uniform fabric color. Small shading variation is still solid.
- WeaveTile: textured fabric without a repeating motif (denim, knit, linen).
- RepeatPattern: IDENTICAL small motifs on an exact regular grid (plaid, stripes, polka dots). Large organic motifs that vary from one to the next are ContinuePrint, even if loosely evenly spaced.
- PasteLogo: any base fabric PLUS distinct placed artwork (a chest logo, slogan text). Name the base tool and report each artwork’s bounding box. A trim, hem, or any band spanning the garment’s full width is fabric structure, NOT artwork.
- ContinuePrint: one non-repeating print covering most of the garment.

Answer with ONLY a JSON object: the tool, the base tool and artwork boxes for PasteLogo, and for prints ONE dense sentence a texture generator could paint from: the motifs and their colors, the background color, the layout on the garment, and the approximate motif scale.

### E.2 TOOL IMPLEMENTATION

**PickColor.** A solid fabric is described by one color. PickColor takes the channel-wise median over the observed texels, a robust estimate under shading and stray silhouette pixels, and fills the panels with it. The other tools reuse the same median as the base color of their fabric.

**WeaveTile.** A fabric such as denim, knit, or melange has no exact repeat, so any clean swatch describes it. We select the swatch among the largest windows inscribed in the observation, preferring the one whose color histogram is closest to the whole garment, which avoids windows in a shadow crease, and rejecting windows observed at grazing angles. FabricDiffusion (Zhang et al., 2024) then turns the swatch into a seamlessly tileable texture. Since the low-frequency shading of the swatch comes from the drape, we remove it from the tile, and clip generated colors to the contrast range of the swatch. The tile is then laid over the panels at its physical scale.

**RepeatPattern.** For a regular lattice such as plaid or stripes, we measure the tile directly from the observation. The repeat comes from the autocorrelation above, and the tile spans the full period of the pattern. Panels whose observed appearance disagrees, such as an argyle front over a solid back, form separate groups with separate tiles. Each panel then aligns the lattice phase to its own observation, so the lattice stays continuous across seams, and observed texels are blended into it.

**PasteLogo.** The base fabric is synthesized first by the tool of its own class, with the texels under the reported artwork marked unobserved, so the fabric continues through the mark. The mark itself is then identified in the partial texture as a compact region off the base color with sharp printed edges, using the reported box as a soft prior, and pasted back at its exact position.

**ContinuePrint.** A print that falls into none of the classes above has no compact rule, so it is continued directly from the observed part. We use FLUX.1 Fill (Black Forest Labs, 2024) to continue the observed print into unobserved cloth, prompted with the description from the router and a swatch of the fabric. Since the unobserved part is unknown, we keep the generation inside the observed color gamut. With  $c_1, \dots, c_m$  the palette centers of the observed colors, every generated color  $c$  is pulled back toward its nearest center  $c_j$ :

$$c \mapsto c_j + \min\left(1, \frac{\tau}{\|c - c_j\|}\right) (c - c_j), \quad (8)$$

where  $\tau$  is a small tolerance that keeps texture detail and removes invented colors. Observed texels are kept, and the completed panels are assembled into a texture atlas for rendering.

## F ADDITIONAL EXPERIMENTS

### F.1 BASELINE IMPLEMENTATION

Table 1 compares methods of very different design, and here we describe how each baseline is implemented and the protocol shared by all methods. Each baseline builds on its official implementation and released weights where they are available, and is otherwise reproduced from its paper.

**SewFormer** (Liu et al., 2023) predicts panels, edges, and stitches from one image with a two-level transformer. We build on the official implementation and checkpoint, and run its full pipeline from the pattern prediction to its own draping of the result. For texture generation, its release binds one fabric image to each garment through the panel layout as UV, with the fabric chosen from a library. We keep this texturing and only replace the library lookup by a swatch cut from the garment region of the photo, so the texture follows the input.

**DressCode** (He et al., 2024) generates a sewing pattern autoregressively from text and adds a tiled PBR texture. We build on the official checkpoints of both its pattern and texture models. Since it takes text rather than an image, we follow its interactive mode: its own prompt turns the photo into the shape and texture descriptions it expects, using the same language model as our agents, and the rest of its pipeline runs unchanged. Its garments are draped with its own pipeline and body. For texture generation, we use its own texture model with the fabric description from the same prompt.

**ChatGarment** (Bian et al., 2025) fine-tunes a vision-language model to write a GarmentCode program from the image. We build on the official checkpoint and its two-step image mode, which first describes the outfit in text and then writes the program from the text and the image. The program is then compiled and draped with its own pipeline on the mean body it targets without texture, since it does not support texture generation.

**GarmentGPT** (Weng et al., 2026) tokenizes panel boundaries into a discrete codebook and decodes them with a vision-language model. We build on the official checkpoints and its single-image inference script, and drape the decoded pattern with its own pipeline. It does not generate texture.

**ISP** (Li et al., 2024b) represents a garment as an implicit sewing pattern with a neural draping model, and fits it to an image by optimizing the mask overlap, one garment at a time from the outermost layer inward. We build on the official implementation and checkpoints, with its own body estimation and segmentation, and keep its optimization schedule. It does not generate texture.

**GarmentRecovery** (Li et al., 2024a) places a learned deformation prior on top of the ISP shape prior, and fits both to the image by optimization. We build on the official implementation, including



its preprocessing for normals, body, and masks, and run its per-category fitting scripts with their default iteration counts for each garment in the outfit. It does not generate texture.

**Dress 1-to-3** (Li et al., 2025) initializes a pattern from SewFormer, generates multi-view images with a diffusion model, and refines the pattern against them through differentiable simulation. Its code is not public, so we reproduce the full pipeline from the paper as faithfully as we can, with the public models it builds on for the initial pattern, multi-view generation, and body estimation, followed by the differentiable refinement with the settings in the paper. For texture generation, we follow the paper and tile a swatch from the photo with FabricDiffusion (Zhang et al., 2024).

We note that every method, including ours, receives the same single input: the photograph in the wild, or the front rendering of the scan on CloSe and 4D-Dress. From there, each method estimates the sewing pattern and drapes the garments, with its own pipeline and simulator. We measure all times in Table 1 on the same workstation, with one NVIDIA RTX PRO 6000 GPU per method and one case at a time. Time is the wall clock from the input image to the final garment mesh.

## F.2 QUANTITATIVE COMPARISON ON TEXTURE GENERATION

Here, we provide additional quantitative results for texture generation, using the in-the-wild photos from our user study (Sec. 4.2). We compare with the baselines that output textured garments: SewFormer (Liu et al., 2023), DressCode (He et al., 2024), and Dress 1-to-3 (Li et al., 2025). We measure the following metrics.

- $\Delta E$  measures how far the garment colors in a render are from those in the photo. We cluster the garment pixels of each image into a small color palette, and compute the Earth Mover’s Distance (Rubner et al., 2000) between the two palettes, with CIEDE2000 (Sharma et al., 2005) as the perceived difference between two colors.
- **CLIP-I** and **DINO** are the cosine similarities between the features of the two garment crops, extracted by CLIP (Radford et al., 2021) and DINOv2 (Oquab et al., 2024). CLIP captures the overall semantics, and DINOv2 is more sensitive to prints and fine texture.
- **PSNR**, **SSIM** (Wang et al., 2004), and **LPIPS** (Zhang et al., 2018) measure the pixel error, the structural similarity, and the learned perceptual distance between two pixel-aligned images.

For a comprehensive comparison, we evaluate in two settings. In the canonical A-pose, the render shares no pixel grid with the photo, so we report only  $\Delta E$ , CLIP-I, and DINO. In the target pose, we repose each garment to the body in the photo and render it through the camera recovered from the photo, so the render lands on the person. This lets us add the pixel-aligned metrics. All metrics are computed only inside the garment region, so the body and the background never affect them. The results are shown in Table 8.

Table 8: **Quantitative comparison on texture generation.** We evaluate the texture fidelity of each method against the input photo on different metrics. Dress 1-to-3 reconstructs the garment directly in the target pose, so we do not report its results in the canonical A-pose.

| METHOD       | CANONICAL A-POSE      |                   |                 | TARGET POSE     |                 |                    |                       |                   |                 |
|--------------|-----------------------|-------------------|-----------------|-----------------|-----------------|--------------------|-----------------------|-------------------|-----------------|
|              | $\Delta E \downarrow$ | CLIP-I $\uparrow$ | DINO $\uparrow$ | PSNR $\uparrow$ | SSIM $\uparrow$ | LPIPS $\downarrow$ | $\Delta E \downarrow$ | CLIP-I $\uparrow$ | DINO $\uparrow$ |
| SewFormer    | 20.61                 | 0.661             | 0.252           | 8.39            | 0.543           | 0.433              | 14.05                 | 0.681             | 0.324           |
| DressCode    | 9.67                  | 0.633             | 0.305           | 8.68            | 0.565           | 0.407              | 12.93                 | 0.652             | 0.275           |
| Dress 1-to-3 | -                     | -                 | -               | 8.26            | 0.531           | 0.410              | 21.26                 | 0.650             | 0.229           |
| <b>Ours</b>  | <b>9.57</b>           | <b>0.821</b>      | <b>0.658</b>    | <b>9.94</b>     | <b>0.615</b>    | <b>0.352</b>       | <b>8.43</b>           | <b>0.800</b>      | <b>0.597</b>    |

Our method achieves the best results on all metrics in both settings. We observe that the gap is largest on DINO, which responds most to prints and logos. The baselines cover each garment with a tiled fabric swatch, which keeps a plausible average color but loses the print.

## F.3 QUANTITATIVE COMPARISON ON LAYER STRUCTURE

Multi-layer reconstruction is one of the central claims of our work, so here we measure the layer structure directly. We compare with the methods that model more than one layer: DressCode (He

et al., 2024), ISP (Li et al., 2024b), and GarmentRecovery (Li et al., 2024a). For a comprehensive comparison, we evaluate on both synthetic data and real photos. We randomly sample 200 outfits from the GarmentCodeData-XL test set, with 50 outfits for each of 1 to 4 layers, where the ground-truth layer number and structure come with the data. DeepFashion2 (Ge et al., 2019) has no such labels, so we select 60 photos by hand, with 20 for each of 1 to 3 layers (since people rarely wear more than three layers), and compare the results against the photo by human judgment. We measure two aspects of the layer structure: *Layer Count Acc.* is the fraction of outfits whose number of layers matches the ground truth. *Layer Structure Acc.* is conditional on a correct count: among the outfits whose count is correct, it is the fraction whose inner-to-outer order and relations between garments (e.g., tucked into, worn over) also match the ground truth. We define it this way because an outfit with a wrong number of layers cannot have the right structure, so the two metrics separate the two kinds of errors. The results are shown in Table 9.

Table 9: **Quantitative comparison on layer structure.** We report the accuracy of the layer count, and the accuracy of the layer structure conditional on a correct count, on GarmentCodeData-XL and DeepFashion2. Only the methods that can model more than one layer are included.

| METHOD          | GARMENTCODEDATA-XL<br>(MAX 4 LAYERS) |               |                  | DEEPPHOTO2<br>(MAX 3 LAYERS) |              |                  |
|-----------------|--------------------------------------|---------------|------------------|------------------------------|--------------|------------------|
|                 | COUNT ACC. $\uparrow$                | STRUCT. ACC.  | COUNT $\uparrow$ | COUNT ACC. $\uparrow$        | STRUCT. ACC. | COUNT $\uparrow$ |
| DressCode       | 34.0%                                | 76.5%         |                  | 31.7%                        | 52.6%        |                  |
| ISP             | 50.0%                                | 87.0%         |                  | 65.0%                        | 84.6%        |                  |
| GarmentRecovery | 49.5%                                | 85.9%         |                  | 61.7%                        | 83.8%        |                  |
| Ours            | <b>95.5%</b>                         | <b>100.0%</b> |                  | <b>96.7%</b>                 | <b>98.3%</b> |                  |

Our method recovers the most correct number of layers and the structure, on both synthetic data and in-the-wild photos. This robustness comes from the Layer Planner, whose skills turn the perception ability of the base model into an explicit layer structure before any pattern is estimated. GarmentRecovery cannot recognize more than two layers, and its count accuracy is zero on the three-layer and four-layer outfits. ISP also struggles beyond two layers, though it models the order and relations slightly better than GarmentRecovery. DressCode behaves very differently on synthetic data and real photos. It often misjudges the number of layers, and after reposing its garments often interpenetrate in simulation, which obscures the layer order (see Fig. 6).

We note that our agents and skills support arbitrary number of layers in a zero-shot way. The Layer Planner infers the number of layers from the photo, and each layer is predicted and refined from innermost to outermost, so the task is well defined for varying number of layers. In everyday clothing, people rarely wear more than three layers, as we observe in DeepFashion2, so we evaluate outfits with up to four layers throughout the paper. This range covers what people wear and what garment production serves, and we consider it sufficient to support the conclusion on generalization.

#### F.4 FAITHFULNESS OF SYNTHESIZED VIEWS

To provide sufficient supervision for every layer and to support zero-shot reconstruction, Stage 1 synthesizes the canonical front  $I^f$ , the back  $I^b$ , and the peeled views of inner layers with generation models. In this way, our method builds on the strong prior of modern image generation models. We verify their faithfulness here, and ablate their effect on the final recovery in Appendix G.1.

**Faithfulness to the photograph.** To measure how faithful a synthesized view is, we compare the canonical front with the photograph it comes from, on the same CloSe and 4D-Dress datasets as Table 1. 4D-Dress records each outfit as a multi-frame sequence, so for reference, a real second photograph of the same outfit, taken from another frame of the same capture, gives the similarity of a faithful view. On both datasets, the photographs of all other outfits give the similarity of an unrelated garment. We crop the garment region of every image with a SegFormer clothes parser (Xie et al., 2021), and compare the crops with CLIP-I, DINO, and  $\Delta E$  from Appendix F.2. Since the images differ in pose, we leave out pixel-aligned metrics.

As shown in Table 10, the canonical view preserves both the colors and the identity of the garment. In color, its distance to the photograph is close to that of a real second photograph, and far below that of other outfits. In features, it is also clearly closer to its own photograph than to the photographs of

Table 10: **Faithfulness of the canonical view.** We compare garment crops with the input photograph. *Second photo* is another frame of the same 4D-Dress capture, a real view of the same outfit. *Other outfits* compares the canonical view with the photographs of all other outfits in the dataset.

| COMPARISON               | CLOSe             |                 |                         | 4D-DRESS          |                 |                         |
|--------------------------|-------------------|-----------------|-------------------------|-------------------|-----------------|-------------------------|
|                          | CLIP-I $\uparrow$ | DINO $\uparrow$ | $\Delta E$ $\downarrow$ | CLIP-I $\uparrow$ | DINO $\uparrow$ | $\Delta E$ $\downarrow$ |
| Other outfits vs. photo  | 0.869             | 0.678           | 7.55                    | 0.705             | 0.391           | 23.29                   |
| Second photo vs. photo   | -                 | -               | -                       | 0.917             | 0.776           | 3.46                    |
| Canonical view vs. photo | 0.907             | 0.845           | 5.49                    | 0.806             | 0.729           | 4.31                    |

other outfits. The margin is smallest on CloSe, where most outfits are a single layer with a top and pants. In this way, we show that the canonical view stays faithful to the garment, and serves as clear supervision for our pipeline, free of the occlusion, unusual poses, and motion-driven deformation in the photograph, with the cloth at rest under gravity.

### F.5 STRESS TEST: REFINEMENT FROM A PLAIN T-SHIRT

To fully demonstrate the capability of Stage 2, we conduct an interesting stress test: from a plain T-shirt to various styles. Here we replace the coarse program with the same plain T-shirt on every case and ask the agentic trio to optimize the outfit in the photo from there.

For every case, we discard the program estimated by Stage 1 and replace it with a plain T-shirt. Everything else stays as in the main experiments, including the canonical views, the recovered body, the layer plan, the backbone, the budget of six simulations, and the agent trio. The trio thus starts from the same T-shirt on every outfit and has to rebuild the garments from this program. Fig. 15 shows four such runs on in-the-wild photos. In every case the first edit changes the construction, and refine the sizes in the later steps.

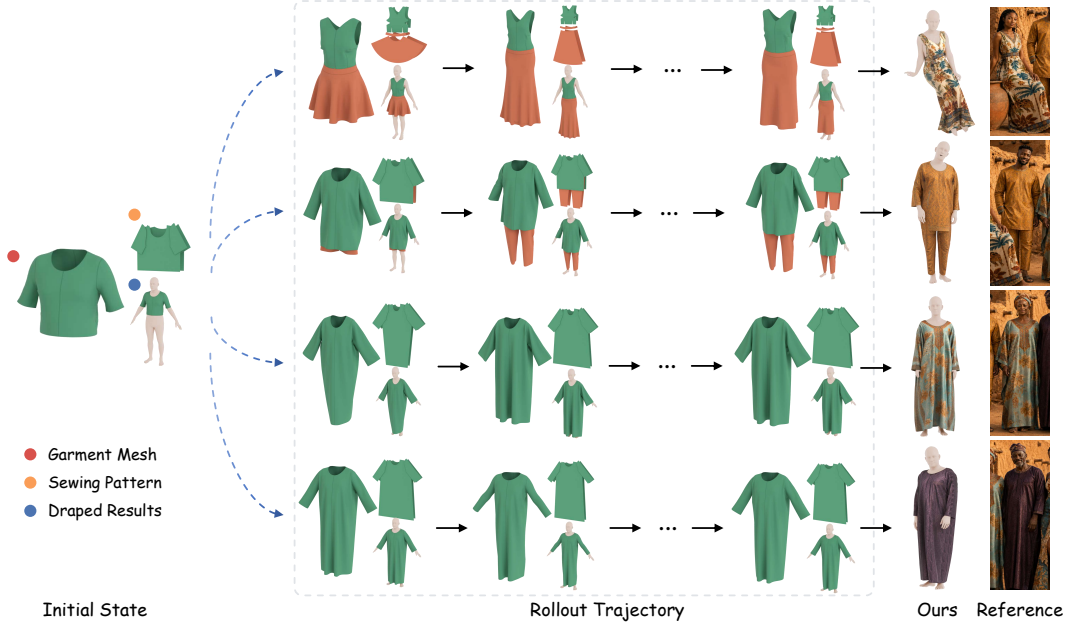


Figure 15: **From one T-shirt to any garment.** We show four examples of refinement runs that start from the same plain T-shirt (left). The trio first intelligently modifies the construction, then refines the sizes to match the photo.

We also run this setting on the real scans of CloSe and 4D-Dress, with the same cases and metrics as Table 1. Fig. 16 shows the per-garment silhouette IoU of the final garment after each simulation, and Table 11 reports the geometry of the reposed garment against the scan.

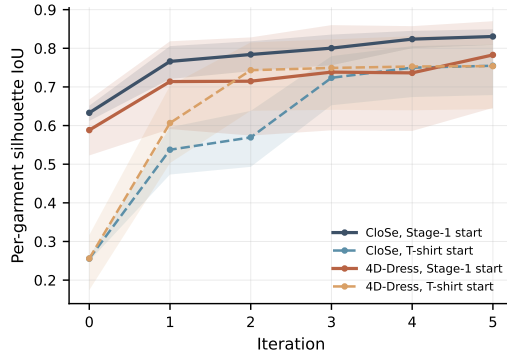


Figure 16: **Refinement from a plain T-shirt.** Silhouette IoU on CloSe and 4D-Dress over the refinement iterations, starting from the Stage-1 program (solid) or from a plain T-shirt (dashed). Bands show the 95% bootstrap confidence interval.

Table 11: **Refinement from a plain T-shirt.** Comparison of the geometry of the reposed garment against the scan, with the protocol of Table 1.

| STARTING POINT              | GEOMETRY (CLOSe) |              | GEOMETRY (4D-DRESS) |              |
|-----------------------------|------------------|--------------|---------------------|--------------|
|                             | CD ↓             | IoU ↑        | CD ↓                | IoU ↑        |
| From Plain T-shirt          | 1.75             | 0.699        | 1.31                | 0.871        |
| From Stage-1 Program (Ours) | <b>1.20</b>      | <b>0.764</b> | <b>1.21</b>         | <b>0.880</b> |

The results show that starting from the T-shirt, the agentic trio rebuilds the outfit close to the full pipeline. After six simulations, the silhouette IoU rises to 0.754 on 4D-Dress and 0.755 on CloSe, against 0.783 and 0.831 from the Stage-1 program. The trio first rewrites the construction and adds the missing garments, then adjusts the sizes, while the Critic and the checkpoints keep the loop from drifting. The agentic trio thus recovers both the style and the sizes through closed-loop refinement, which shows its strong capability.

The results also confirm the importance of Stage 1. The coarse program provides a reasonable and good initial solution, which fixes the construction from the photo in a single forward pass. From the T-shirt, the loop has to find the construction by trial, so the first two or three simulations go to it, and the curve lags the full pipeline by about two iterations and ends 0.03 to 0.08 lower at the same budget. With the coarse program, the loop starts close to the target and spends almost all its budget on the fit, so it converges to a better result in fewer iterations. The closed loop is thus the engine of accuracy, and the coarse estimate is the warm start that saves the budget.

## F.6 ADDITIONAL VISUAL RESULTS

Here, we present more visual results of our method on in-the-wild photos, covering different layering, poses, and body shapes (Fig. 23).

## G ABLATION STUDY

### G.1 ABLATION ON SYNTHESIZED VIEWS

To validate the source of supervision, we replace our synthesized views with weaker supervision and measure the final recovery against real scans, on the same CloSe and 4D-Dress datasets as Table 1. We compare four sources of supervision: *Input only* supervises the agents with the photograph alone, with no canonical view, back view, or peeling. *Front only* keeps the canonical front and the peeling, and removes every back view. *No peeling* keeps the layer plan, and reads and refines every layer against the original canonical A-pose views. *Full* is our method. All settings share the recovered body and the budget of six simulations.

Table 12: **Ablation on synthesized views.** We report the geometry of the reposed garment against the scan, graded as in Table 1. Since CloSe has few multi-layer outfits, while 4D-Dress captures outfits with an outer garment and labels it as its own class, we test no peeling only on 4D-Dress.

| SETTING     | CLOSe       |              | 4D-DRESS    |              |
|-------------|-------------|--------------|-------------|--------------|
|             | CD ↓        | IoU ↑        | CD ↓        | IoU ↑        |
| Input only  | 1.36        | 0.743        | 2.28        | 0.782        |
| Front only  | 1.29        | 0.750        | 1.73        | 0.833        |
| No peeling  | -           | -            | 1.64        | 0.846        |
| <b>Full</b> | <b>1.20</b> | <b>0.764</b> | <b>1.21</b> | <b>0.880</b> |

As shown in Table 12, each synthesized view improves the recovery. The canonical view brings the largest gain, and we find that the gain grows with the difficulty of the pose. It is modest on CloSe, whose inputs are standing scans already close to the A-pose, and large on the dance poses of 4D-Dress. The back view lowers the error further, which shows that the generated back carries real information about the garment. Peeling further improves the recovery by providing supervision for every layer, which is essential for multi-layer outfits. In this way, we show that our synthesized views are an effective source of supervision, and each of them brings the recovery closer to the real garment. Together with Appendix F.4, we show that the synthesized views are faithful to the photograph and serve as reliable supervision for our pipeline.

## G.2 ABLATION ON AGENTIC TRIO DESIGN

**w/o Rollback Tool.** Rollback is a key control tool of our agentic trio. It returns the sewing pattern to an earlier checkpoint and keeps the failed attempt in memory as a reference for later edits (Sec. 3.3). When an edit overshoots the target or compromises physical plausibility, the Orchestrator rolls it back at once. This keeps the refinement from optimizing further in a wrong region of the design space. To ablate it, we remove Rollback, so every edit builds on the latest step and the last step is delivered. Both variants start from the same coarse pattern. As shown in Table 13, without Rollback a wrong construction edit or a tangled drape is carried to the end. Structure accuracy and IoU drop, and the delivered drapes carry four times as many self-intersecting edges.

Table 13: **Ablation on Rollback.** We ablate the Rollback tool of the agentic trio. Self-intersection is the share of cloth edges that intersect the garment itself on the delivered drape.

| COMPARISON   | GARMENTCODEDATA-XL | DEEPFASHION2 | SELF-INTERSECTION (%) ↓ |
|--------------|--------------------|--------------|-------------------------|
|              | STRUCT. ACC. ↑     | IoU ↑        |                         |
| w/o Rollback | 58.70              | 0.736        | 0.16                    |
| <b>Full</b>  | <b>73.59</b>       | <b>0.839</b> | <b>0.04</b>             |

## G.3 ABLATION ON SINGLE-AGENT REFINEMENT

Our refinement is built on an agentic trio with complementary roles. The Tailor edits the program, the Critic judges the drape, and the Orchestrator controls the session, and each agent keeps its own tools and its own memory (Appendix D.1). This decomposition separates editing, evaluation, and control, so an edit is judged by an independent Critic and a bad step is rolled back to a previous checkpoint. To measure how much this design contributes, we ablate it and hand the whole refinement to a single agent that judges its own drape and rewrites the program by itself.

Since no prior work provides such a baseline for single-agent garment refinement, we build one on TextGrad (Yuksekgonul et al., 2024), which gives the single-agent loop a clean and well-defined form. TextGrad treats an LLM system as a computation graph and optimizes a variable with *textual gradients*, which replace the numeric gradients of backpropagation with natural-language feedback. In our setting, the variable is the program  $P^t$  of the current layer, and the forward pass is the simulation that turns it into an observation  $O^t$ . A VLM first evaluates this observation against the reference

views and writes the loss in natural language:

$$\mathcal{L}^t = \text{VLM}(O^t, I_k^f, I_k^b), \quad (9)$$

where  $\mathcal{L}^t$  is a textual evaluation that names what differs between the drape and the photograph. The backward pass then turns this evaluation into feedback on the variable, the textual gradient:

$$\frac{\partial \mathcal{L}^t}{\partial P^t} = \nabla_{\text{VLM}}(P^t, O^t, \mathcal{L}^t), \quad (10)$$

where  $\nabla_{\text{VLM}}$  is a VLM call that reads the program, its observation, and the loss, and writes which parameters to change and how. Finally, a textual gradient descent step rewrites the variable according to the feedback:

$$P^{t+1} = \text{TGD.step}\left(P^t, \frac{\partial \mathcal{L}^t}{\partial P^t}\right), \quad (11)$$

which is a VLM call that outputs the updated parameter map of  $P^t$ , with the same tools available as to our Tailor. All three calls are made by the same agent, which therefore judges its own drape and edits its own program.

For a fair comparison, both methods start from the same coarse pattern and use the same simulator, backbone, observations, tools, memory, and budget of ten simulations per layer. The only difference is the role decomposition: the single agent evaluates its own drape, plans its next edit, and edits the program within one context, where the trio separates these into the Critic, the Orchestrator, and the Tailor. We further implement two variants that differ in which step is delivered. The plain variant delivers the last iterate, as TextGrad does. The self-selection variant lets the agent score each drape itself and delivers its best-scored step, which is the most generous reading of a single agent. We evaluate both on the real scans of CloSe and 4D-Dress to show robustness to in-the-wild photos, the same setting as Table 1.

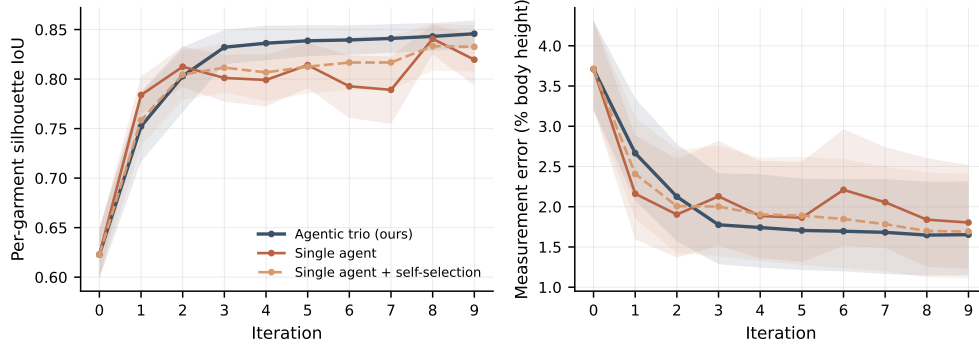


Figure 17: **Ablation on single-agent refinement.** Per-garment silhouette IoU and measurement error of the delivered garment after each simulation on CloSe and 4D-Dress. Bands show the 95% bootstrap confidence interval. The single agent oscillates and plateaus early, while our agentic trio converges progressively.

Table 14: **Ablation on single-agent refinement.** All methods use the same coarse pattern, backbone, tools, memory, and 10 simulations per layer. Measurement error is in percent of body height. Tokens and time are the output tokens and VLM time per case. We find that the single agent is worse in quality, biased in self-evaluation, and higher in cost.

| METHOD                        | IOU $\uparrow$ | MEAS. ERR. $\downarrow$ | EDITS / ITER. | TOKENS (K) $\downarrow$ | TIME (S) $\downarrow$ |
|-------------------------------|----------------|-------------------------|---------------|-------------------------|-----------------------|
| Single agent                  | 0.820          | 1.80%                   | 6.2           | 38.7                    | 748                   |
| Single agent + self-selection | 0.833          | 1.69%                   | 6.2           | 38.7                    | 748                   |
| Agentic trio (ours)           | <b>0.846</b>   | <b>1.65%</b>            | <b>2.8</b>    | <b>15.0</b>             | <b>412</b>            |

Fig. 17 and Table 14 show the results, which reveal three problems of the single agent.



**Worse quality.** The single agent plateaus early and oscillates: its IoU reaches 0.812 after three simulations and stays at 0.820 after ten, while our trio keeps improving to 0.846 with lower measurement error. Between consecutive steps, its IoU moves by 0.063 on average, twice the 0.035 of the trio. This oscillation comes from its edits: although free to change any subset of the parameters, the single agent touches more than twice as many per iteration as the trio and reverses its own edits more often, since the textual evaluation describes the current drape but not how far each parameter moved it. Our agentic trio avoids this: The Critic reports a few discrepancies with measured evidence, the Orchestrator picks which of them to address in this round, and the Tailor edits only those parameters, so every round makes a small, targeted change and the loop converges progressively with previous memory for reference.

**Self-bias.** The single agent evaluates the drape, derives the feedback, and rewrites the program with the same model, so every edit is judged by its own author. The self-selection variant isolates this judgement: it sees every step of the run with the same views and measurements the Critic sees, and differs from the trio only in who scores them. Selecting by its own scores raises the IoU from 0.820 to 0.833, only half of the way to the 0.846 of the trio, so the remaining gap comes from the judgement itself, not from the evidence. This is consistent with the self-bias in Sec. 3.3 (Xu et al., 2024; Panickssery et al., 2024; Huang et al., 2024): a model tends to favor its own outputs, so the agent that made an edit is a poor judge of it, and an independent Critic provides exactly the judgement it lacks.

**Higher cost.** Every iteration of the single agent is three long generations, an evaluation, a textual gradient, and a rewrite of the full parameter map, so it emits  $2.6\times$  the output tokens and spends  $1.8\times$  the VLM time of the trio, and it still needs more iterations for lower quality. The calls of the trio are short, since each role answers one well-posed question through structured function calling (Schick et al., 2023) and the tools carry the arithmetic. The fixed system prompt and tool schema of each agent also form a stable prefix across rounds, which prompt caching (Gim et al., 2024) reuses, and older views are compacted into text notes so the context stays bounded (Appendix D.1).

#### G.4 ABLATION ON TEXTURE GENERATION

We ablate our texture generation with two variants of naive UV mapping (Fig. 18): direct back-projection without the instance masks of Grounded SAM, and with the masks restored.

Without the masks, every pixel behind the drape silhouette counts as texture. Colors therefore bleed across garments: the stripes of the tee print onto the pants, and residual skin and background enter along every silhouette.

With the masks, the garments separate cleanly, yet the projection is still a fixed mapping. The cloth no camera saw admits no extrapolation at all, and even the observed print lands warped on the flat panels, since the folds and foreshortening of the drape are baked into every sample.

Our method instead divides garment textures into five structured classes, each handled by a carefully designed extrapolation tool. Letting the agent judge the class itself keeps the stage robust across in-the-wild garments.

## H USER STUDY

Here, we show examples of the Google Forms questions used in our user study. Fig. 19 shows the instructions and the geometry question, where non-artistic participants pick the result that best matches the shape of the clothes in the photo among all methods. Fig. 20 and Fig. 21 show the plausibility and texture questions, where participants rate each result from 1 to 5. Methods are anonymized and shuffled for every case.

## I LIMITATIONS AND FUTURE WORK

Although DressAgent generates structured, simulation-ready sewing patterns for a wide range of in-the-wild outfits, three directions remain open for future work:

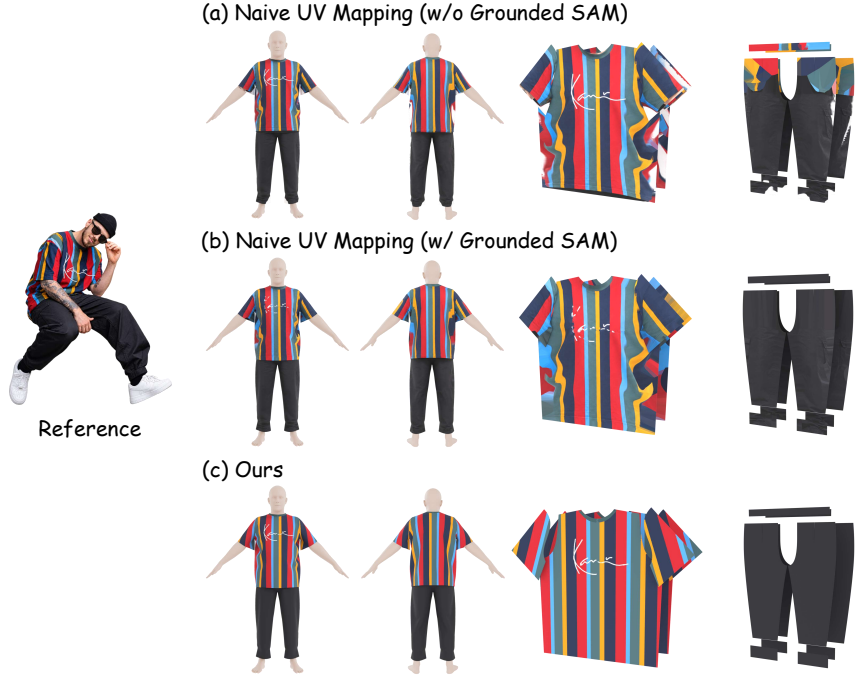


Figure 18: **Ablation on texture generation.** (a) Naive UV mapping without Grounded SAM mixes garments: the stripes of the tee bleed onto the pants. (b) With Grounded SAM, the garments separate, but unobserved cloth admits no extrapolation and the observed print lands warped. (c) Our agentic generation method continues the stripes at their measured repeat, restore the logo, and keep the pants a uniform black.

**Irregular decoration.** GarmentCode-XL expresses the vast majority of everyday outfits, but there are free-form ornament that cannot be represented by the current representation, such as the loose sash in Fig. 22. Future work may let the agent write code to extend the representation on its own, with the input image as guidance.

**Cloth thickness.** Current method does not support inverse garment design of garments with thickness, such as a padded down jacket. As simulation advances and agents grow stronger, future work may let the agent author thickness and material properties along with the pattern.

**API cost.** Currently, the full system prioritizes high-fidelity, simulation-ready generation over cost. In our experiments with GPT-5.6-luna, the average API cost per case is \$0.52, \$0.75, \$0.91, and \$1.49 for outfits with one to four layers. Cheaper or open-source models lower the cost but also the quality (Fig. 8), reflecting a quality-cost trade-off.

## User Study of Image-to-Garment Reconstruction

Thank you for taking part in this study.

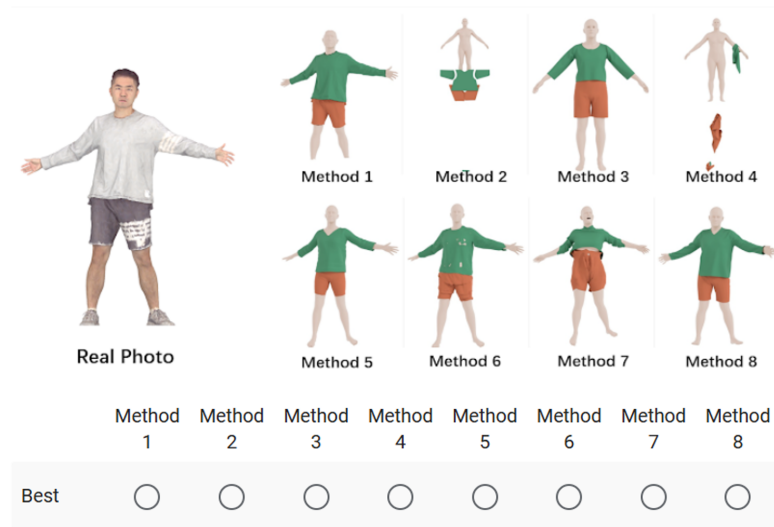
We are comparing different methods that create 3D clothes from a single photo. The clothes should look like the photo and can be used in simulation (for example, draped on a body).

Please rate **how good each result looks in shape and in appearance** (colors and patterns), according to their questions.

There are no right or wrong answers. Please go by what you see. The study takes about 10–15 minutes. Thank you!

Look at the real photo first. Then look at the 8 results beside.

**Which result best matches the overall shape of the clothes in the photo** (length, width, sleeves, collar, etc.)?



|      | Method 1              | Method 2              | Method 3              | Method 4              | Method 5              | Method 6              | Method 7              | Method 8              |
|------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| Best | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

Figure 19: Screenshot of the user study question on geometry.

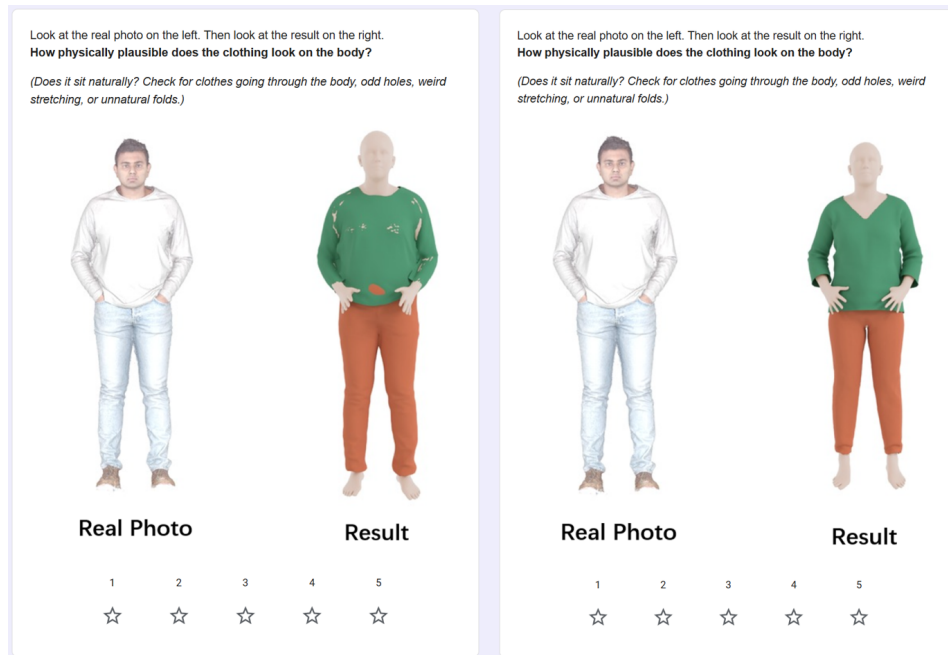


Figure 20: Screenshot of the user study question on plausibility.

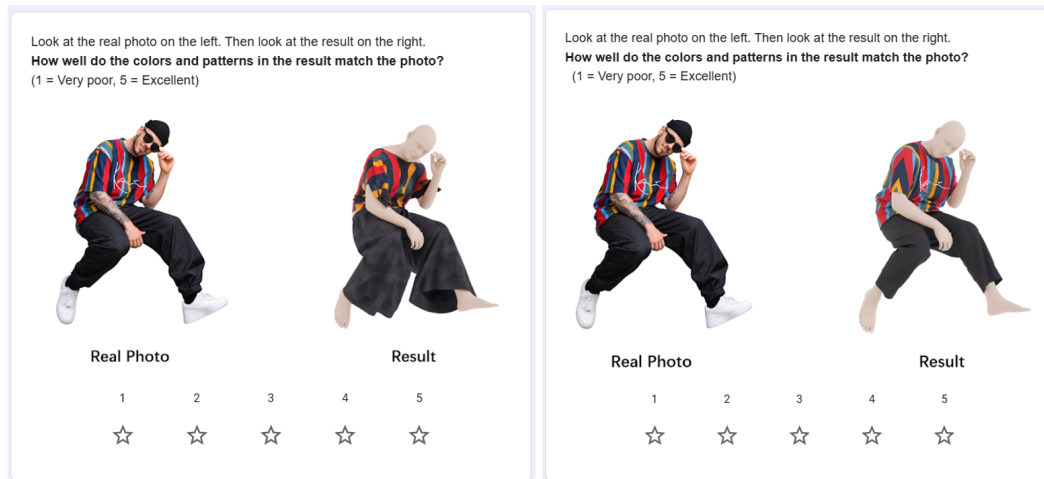


Figure 21: Screenshot of the user study question on texture.

Figure 22: **Failure case.** The sash hanging free at the waist cannot be recovered by the current representation. Future work may let the agent write code to extend the representation on its own.



Figure 23: Additional results on in-the-wild photos with different layering, poses, and body shapes.