

Combining FashionCLIP and Local Language Models for Outfit Recommendation

Deven Shah | CS664 Artificial Intelligence | Term Project

I. Introduction

The wardrobe copilot recommends outfits using photos of clothing in a selected local wardrobe. Users can upload their own garments or browse a separate DeepFashion2 validation collection, review suggested categories, and request styles such as business casual. Follow-up requests can revise an earlier option. Every suggestion displays the selected source photos. The objective was to combine computer vision, a local language model, and explicit Python rules in one useful application. The central question was whether those components could produce valid, responsive recommendations on a personal computer without paid inference or internet access during normal use.

The application uses pretrained models and stores both wardrobes locally. Personal uploads are intended to show one garment or pair of shoes per photo. DeepFashion2 photos may contain multiple garments; they are used as an image-level evaluation collection, not as detected garment crops. Virtual try-on, shopping, and live weather are outside the project scope.

II. Methodology

The application uses a React and TypeScript interface with a Python FastAPI backend. Table 1 identifies the main tools and their roles. Model weights are downloaded during setup, then loaded from the local cache during normal use.

Table 1. Main components of the application.

Component	Model or code	Role
Interface	React, TypeScript, FastAPI	Upload and edit garments; chat and display outfits
Image handling	Pillow	Validate orientation and create normalized photos
Recognition and retrieval	Marqo-FashionCLIP, ViT-B/16	Suggest tags and rank garments against style requests
Conversation	Gemma 3 4B via Ollama	Interpret requests; select and explain outfits
Rules and storage	Python, SQLite	Build and validate combinations; save wardrobe and chat

Users can upload photos, correct personal item names and tags, mark pieces unavailable, and review outfits with their original images. Sequential batch uploads allow individual retries. The evaluation wardrobe contains 16,411 validation JPEGs manually placed in data/deepfashion2. That is 48.7%, approximately 50%, of the 33,669 images in the published validation split (Ge et al., 2019; DeepFashion2 project). A local indexing script applies FashionCLIP to each whole photo and saves suggested labels and embeddings in SQLite. No annotation JSON files were supplied with this local subset, so the suggested labels are not ground truth.

Marqo-FashionCLIP supplies pretrained ViT-B/16 image and text encoders. The application compares each photo embedding with embeddings of garment descriptions such as "A photo of a button-down shirt." The description with the highest cosine similarity determines the suggested subtype and broad category. Separate description lists suggest color, pattern, and broad style. This is zero-shot classification: the application compares pretrained embeddings with candidate labels instead of training a new classifier. The available labels are limited to the descriptions defined in vision.py.

For a style request, FashionCLIP also ranks saved garments by text-image similarity. The backend keeps up to eight candidates per category, plus explicitly required pieces. If image ranking is unavailable, manually categorized items can still be considered using metadata. The stylist follows a bounded Python workflow: interpret the request, resolve

references, retrieve items, assemble combinations, ask the language model to choose, and validate the output. It normally makes two Ollama calls and allows one correction attempt.

The first Gemma call extracts style, required items, exclusions, and references to earlier options. Python matches item names against the wardrobe and asks for clarification when several pieces fit a reference. Constraints persist across follow-ups; a new conversation clears them. Replacing shoes in one option keeps the other pieces and excludes the original shoes. Python constructs outfits from a top and bottom, or a one-piece garment, plus footwear when available. Outerwear and accessories are optional. To bound the search, it retains at most 64 core combinations and 80 combinations in total after optional pieces are added. The second Gemma call chooses from these combinations and writes short explanations. A JSON schema restricts selections to existing combinations, and temporary item aliases reduce copying errors. The model receives item metadata and IDs rather than photos. Python rejects invented or unavailable IDs, incompatible garment roles, excluded or missing required items, and duplicate combinations. It also checks inventory again after generation. When a full outfit is impossible, the response identifies missing components or returns fewer than three options. These checks establish outfit validity, not fashion quality. A structurally valid combination may still be unsuitable for the requested occasion.

The recommendation benchmark has ten fixed prompts: eight independent requests for styles or exclusions, followed by two refinements of earlier results. Each prompt uses only the DeepFashion2 wardrobe. The run records rule-valid and distinct outfits, missing components, and wall-clock response time. Style quality is reviewed separately rather than inferred from structural validity.

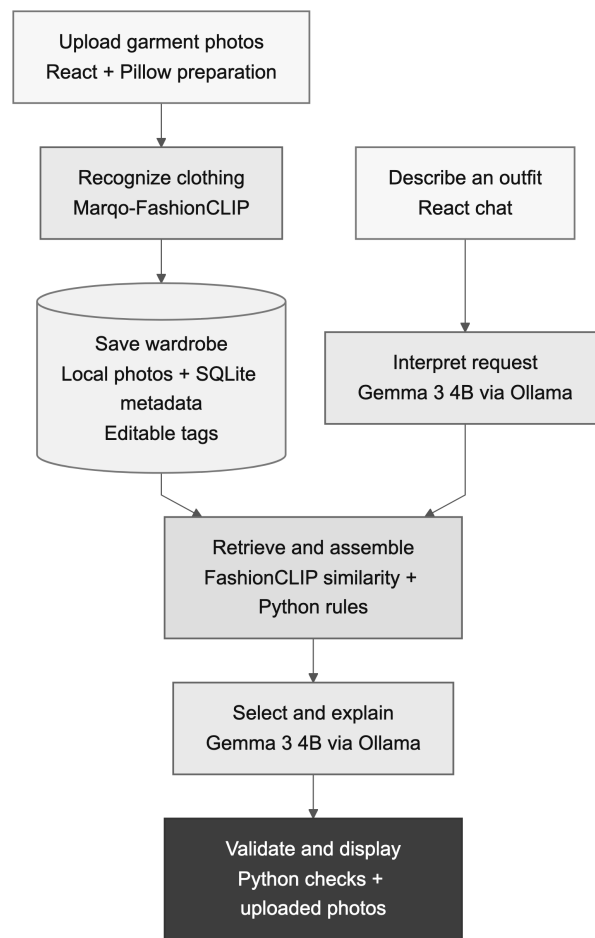


Figure 1. The wardrobe copilot workflow.

III. Results

The full local DeepFashion2 index contained 16,411 photos. FashionCLIP assigned 5,404 to bottoms, 4,478 to tops, 3,332 to one-piece garments, and 3,197 to outerwear. These are model predictions on whole images, not accuracy measurements against DeepFashion2 annotations. The dataset has no footwear category, so the benchmark can produce structurally valid but partial outfits.

The benchmark was run on an Apple M4 Mac with 16 GB of memory using local Gemma 3 4B through Ollama. Eight prompts start with no prior conversation; the last two refine an option from the business-casual and casual cases, respectively. Table 2 gives the exact prompts and the number of outfits that passed the application rules.

Across ten requests, 10 returned at least one outfit and 22 outfits passed the structural rules. 22 were explicitly marked partial; the measured request time totaled 204.0 seconds. These are development wall-clock measurements, not isolated performance benchmarks. The checks reject nonexistent IDs, unavailable items, incompatible garment roles, duplicate combinations, and violated exclusions. They cannot determine whether an outfit looks appropriate for the requested occasion. The exact images can also show more than one garment because no item-level detection was applied to DeepFashion2.

Table 2. Ten-prompt recommendation benchmark on the DeepFashion2 wardrobe.

Prompt	Valid	Partial	Time (s)
Give me three casual outfits.	3	3	31.38
Give me three business-casual outfits.	3	3	18.36
Give me three formal outfits.	3	3	18.28
Give me three athletic outfits.	3	3	20.46
Give me three outfits for a warm day.	3	3	20.76
Give me three relaxed weekend outfits.	3	3	19.56
Give me three casual outfits without trousers.	1	1	17.04
Give me three business-casual outfits without black clothing.	1	1	17.27
Make option 2 more casual.	1	1	19.67
Make option 1 a little more formal.	1	1	21.23

Every returned combination was distinct within its prompt and referred to indexed DeepFashion2 photos. Missing footwear was reported rather than filled with an invented item. A business-casual option used one photo tagged as a blue sling top that showed a striped shirt, and another tagged as white trousers that showed a full-body look with dark trousers. Both images contain more than the proposed garment, so the displayed combination is difficult to judge as a new outfit. Rule-valid IDs alone do not establish visual style quality.

IV. Conclusion

The wardrobe copilot combines pretrained vision, text-image retrieval, a local language model, and Python validation. The ten-prompt DeepFashion2 run demonstrates that the application can produce traceable outfit suggestions from a large local image collection while keeping the personal wardrobe separate. The main limitations are image-level classification, missing footwear in DeepFashion2, slow language-model responses, and style judgments that cannot be established by structural checks alone. The 16,411-image subset is approximately half of the published validation split, but its local annotation files were unavailable for measuring recognition accuracy.

Further evaluation should use annotated single-garment images, footwear, and independent ratings of outfit quality. A garment detector and learned compatibility model could address these gaps, but both would require additional data and separate validation.

V. References

- Ge, Y., Zhang, R., Wang, X., Tang, X., & Luo, P. (2019). DeepFashion2: A versatile benchmark for detection, pose estimation, segmentation and re-identification of clothing images. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 5337–5345). https://openaccess.thecvf.com/content_CVPR_2019/html/Ge_DeepFashion2_A_Versatile_Benchmark_for_Detection_Pose_Estimation_Segmentation_and_CVPR_2019_paper.html
- Marqo. Marqo-FashionCLIP model. <https://huggingface.co/Marqo/marqo-fashionCLIP>. The project pins revision 44f4c655124ed71e90cbf528d82f508d69d8a81b.
- Ollama. Gemma 3 model library. <https://ollama.com/library/gemma3>. The project uses the gemma3:4b model.
- Ge, Y., Zhang, R., Wu, L., Wang, X., Tang, X., and Luo, P. (2019). DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images. arXiv:1901.07973. <https://arxiv.org/abs/1901.07973>.
- DeepFashion2 project. Dataset repository and validation statistics. <https://github.com/switchablenorms/DeepFashion2>.