Style Synth: ML Fashion Recommendation System

DSAN 6700: Machine Learning App Development
Courtney Green, Li-Wen Hu, Satomi Ito, Nandini Kodali, Sophia Rutman

Deciding what to wear is a daily challenge that many people face, especially balancing personal style, occasion, appropriateness, and weather considerations. This project presents a fashion recommendation system that helps users style their existing wardrobe or discover new outfit combinations. Users upload images of their clothing items, and the system leverages computer vision and similarity search to generate cohesive outfit recommendations tailored to specific occasions and seasons.

Our system combines several key technologies: ResNet-50 for extracting visual embeddings from clothing images, FAISS for efficient similarity search, and PostgreSQL for storing user data and high-dimensional vectors. The frontend is built with Streamlit, providing an interface for wardrobe management and outfit generation, while the backend uses FastAPI to handle API requests and machine learning inference. The entire application is containerized using Docker and orchestrated with Docker Compose, enabling consistent deployment across environments. We also implemented a monitoring stack using Prometheus and Grafana to track system performance and ML inference metrics in production.

The workflow is as follows: users upload clothing images, which are classified into categories and converted into 2048-dimensional embeddings. When generating outfits, the system filters items by user-selected occasions and season, then uses cosine similarity matching to find visually compatible pieces across categories. The result is a ranked list of complete outfit suggestions powered by learned visual representations.

1. System Architecture

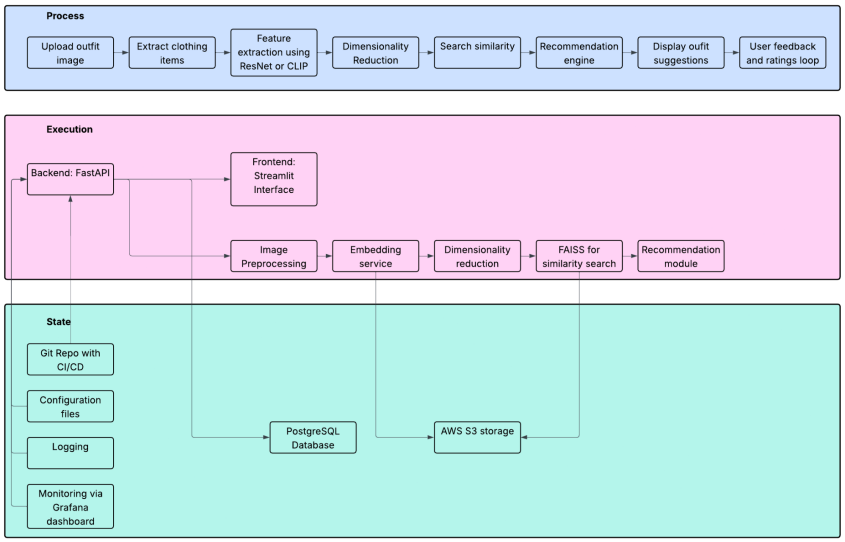

Figure 1: Solution Architecture

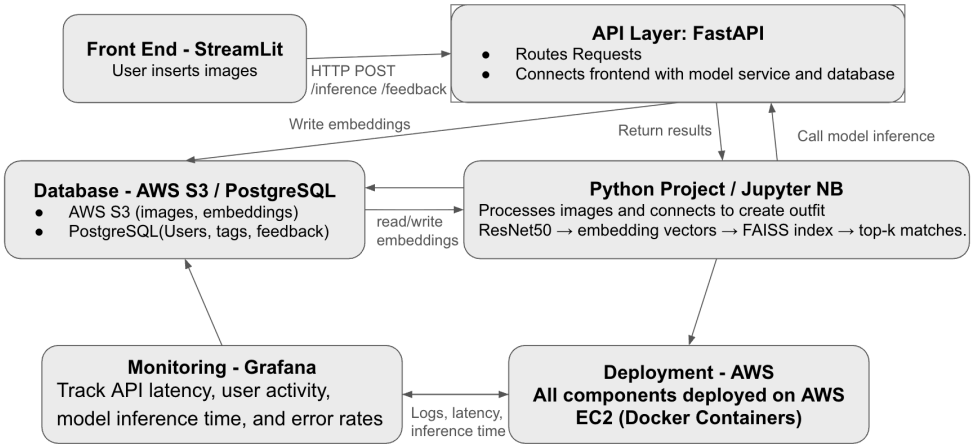

Figure 2: Technical Architecture

2. Exploratory Data Analysis

This project uses the Fashion MNIST dataset, which contains 60,000 grayscale images across 10 clothing categories: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. Each 28x28 image was upscaled to 224x224 to match the input requirements of ResNet-50. We extracted a 2048-dimensional embedding from each image using a pretrained ResNet-50 model. To visualize the learned feature space, we applied t-SNE on a sample of 2,000 embeddings (Figure 4). The visualization shows distinct clustering by category, with footwear items clustered separately from upper-body garments, while visually similar categories like Pullover and Coat show overlap, reflecting their semantic similarity.

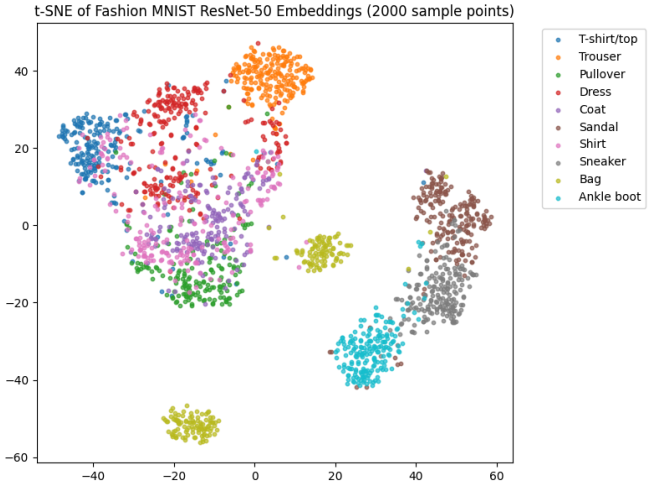

Figure 3: T-SNE PLOT FROM RESNET

3. Engineering & Preprocessing

All images undergo a preprocessing pipeline before embedding extraction. Images are resized to 224x224 pixels, converted from grayscale to RGB, and normalized using ImageNet statistics (mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.225]). This ensured compatibility with the pretrained model weights. We use ResNet-50 pretrained on ImageNet as a feature extractor. The final classification layer is removed, and we extract the 2048-dimensional output from the global average pooling layer. Each embedding is L2-normalized to ensure cosine similarity comparisons. To improve search efficiency and reduce memory usage, we apply PCA to reduce embeddings from 2048 to 12 dimensions. This achieves a 16x compression ratio while retaining approximately 95% of variance, making similarity search faster without sacrificing recommendation quality. User wardrobe images are uploaded to AWS S3, while metadata (category, brand, colors, occasions, seasons) and embeddings are stored in PostgreSQL. The FAISS library indexes the embeddings for a nearest-neighbor retrieval during outfit generation.

4. Algorithm Details

Our baseline approach uses logistic regression trained on ResNet-50 embeddings to classify uploaded clothing images into one of the 10 Fashion MNIST categories. This provides automatic category labeling when users upload new wardrobe items.

4.1 Recommendation Engine

The core recommendation system uses FAISS to perform efficient nearest-neighbour search over the embedding space. Given a query item, Faiss retrieves the most visually similar items using either L2 distance or cosine similarity. We tested both index types and found comparable performance, with cosine similarity being slightly more intuitive for normalized embeddings.

4.2 Outfit Generation

The outfit recommender builds complete outfits by combining items across categories. Given a user's selected occasion and season, the system first filters the wardrobe to matching items. It then groups items by category and uses cosine similarity to find compatible pieces. For each top, the system finds the best-matching bottom, then iteratively adds shoes, outerwear, and accessories based on a similarity threshold. To ensure variety across recommendations, the system tracks used items and prioritizes unused pieces for outfits. The final outfits are ranked by average similarity score.

5. Evaluation Approach

5.1 Benchmark Testing

For the dimensionality-reduction module, we measured how well PCA preserved the structure of the original embedding space by computing neighbor-preservation rates and cosine-similarity drift before and after projection. For similarity search, we benchmarked FAISS by comparing its approximate nearest-neighbor results with an exact brute-force search to calculate recall@k, ensuring that retrieved items closely matched the true nearest neighbors. We also timed query execution under different index configurations to quantify speed-ups gained from clustering or

quantization. Finally, for the recommendation component, we used controlled query sets with known semantic relationships to test whether the ranking and filtering logic consistently prioritized items with high embedding similarity.

5.2 Production Monitoring

We implemented Prometheus metrics throughout the system to track real-world performance. Key metrics include ML inference time (embedding extraction, classification, and recommendation latency). API request duration by endpoint, and AWS S3 operation times. These metrics are visualized in Grafana dashboards, allowing us to monitor system health and identify bottlenecks.

5.3 Qualitative Assessment

Evaluation Category	What It Measures	Qualitative Interpretation	Strengths	Limitations	When It Matters Most
Query Throughput (q/s)	Number of similarity searches per second	Higher q/s indicates better scalability for real-time or high-volume workloads	Shows how PCA + FAISS drastically improve speed (hundreds → thousands of q/s)	May trade off accuracy depending on dimensionality	High-traffic applications, interactive recommendation systems
Latency per Query	Time required to retrieve nearest neighbors	<1–3 ms/query is considered extremely fast and responsive	Highly sensitive to index type and vector dimension	Very low dimensions may sacrifice semantic accuracy	User-facing apps requiring instant responses
Memory Footprint	Size of embedding matrix + FAISS index	Lower memory improves deployability and allows larger datasets	PCA achieves substantial compression (up to 32×)	Very aggressive compression may reduce overlap accuracy	Mobile/edge deployment, large-scale catalogs
Top-K Overlap (Accuracy)	Agreement with results from full 2048D vectors	Measures how much semantic meaning is preserved after dimensionality reduction	Mid-range PCA (128–256D) offers strong balance between accuracy and cost	Overlap declines sharply only when the dimension drops too low	Recommendation quality, semantic relevance

Variance Explained (PCA)	Information retained by the reduced vector space	Higher variance suggests better preservation of structure	Helps select meaningful PCA dimension sizes	Not a full guarantee of nearest-neighbor preservation	Model selection + tuning embedding quality
Filtering Overhead	Additional time per query when applying category filters	Ideally, minimal (<10% overhead)	Demonstrates robustness of retrieval under constraints	Poor implementation could slow down search	Personalized or rule-aware recommendation logic

6. Feedback Adaptations

6.1 Match Score Removal

Our initial implementation displayed a "Match Score" percentage alongside each generated outfit, representing the average cosine similarity across paired items. However, instructor feedback highlighted that end users may not intuitively understand what this metric represents or how to interpret it meaningfully, so we removed the match score display from the interface, simplifying the user experience by focusing on outfit recommendations rather than revealing internal algorithmic metrics that could confuse non-technical users.

6.2 Post-Generation Outfit Customization

During the app demonstration, a peer asked whether users could add items from their wardrobe to an outfit after it was generated. While our original implementation allowed users to remove items from generated outfits, there was no mechanism to add additional pieces. In response, we implemented an "Add from Wardrobe" feature accessible from both the Outfit Builder and Edit Outfit interfaces. This expandable section displays wardrobe items not currently in the outfit, filtered by category, allowing users to supplement machine-generated recommendations with their own selections. Combined with the existing removal functionality, users now have full control to customize outfits, adding items the algorithm may have missed or removing pieces that don't fit their vision. This hybrid approach combines algorithmic suggestions with user agency, acknowledging that personal style preferences may not always align with similarity-based matching.

6.3 Style/Vibe Filtering Discussion

Finally, a peer inquired about filtering by aesthetic "vibes" such as Y2K, clean-girl, grunge, or minimalist styles. While this presented an interesting extension, we determined it was not feasible within our current scope for several reasons. First, implementing vibe detection would require either manually tagging each item with user feedback or training a separate classifier on labeled aesthetic data, neither of which was practical given our timeline. Second, style aesthetics

are inherently subjective and trend-dependent, making ground-truth labeling challenging. Third, the system cannot recommend items the user does not own; if a user's wardrobe lacks items matching a particular aesthetic, the feature would produce empty results or frustrating partial matches. We documented this as a potential future enhancement.

7. Results

In our benchmark experiments, we evaluated how different PCA dimensions and index configurations affect the efficiency and fidelity of similarity search in our ML application. Starting from the original 2,048-dimensional embeddings for 60,000 items (≈468.75 MB), exact search achieved only 33.54 queries per second (q/s), with an average latency of 29.81 ms per query. Applying PCA dramatically reduced both memory and query time: at 64 dimensions, the index size dropped to 14.65 MB (32× compression), throughput increased to 820.90 q/s with 1.22 ms per query, and the top-K overlap with the 2,048D reference remained perfect (overlap = 1.00). A 128-dimensional setting provided a more conservative trade-off, explaining 86.69% of the variance while sustaining 517.34 q/s, 1.93 ms per query, 16× compression, and a top-K overlap of 0.70. Higher-dimensional settings (256–1,024D) further increased explained variance (up to 99.39%) but incurred slower throughput (down to 74.37–33.54 q/s) and substantially higher memory usage (58.59–234.38 MB), with only modest gains in overlap. At the index level, cosine-based FAISS search at 128D outperformed L2 distance, reaching 405.73 q/s (2.46 ms) versus 298.14 q/s (3.35 ms). Finally, we found that applying category or exclusion filters introduced minimal overhead. This filtered queries remained within 3.16–3.59 ms per request (<11% slowdown), indicating that our recommendation-time filtering logic preserves most of the raw retrieval speed while enabling more controlled, constraint-aware results.

8. Challenges & Conclusion

8.1 Embedding Quality vs. Performance Tradeoffs

Balancing recommendation quality against computational efficiency required careful tuning. Our original 2048-dimensional embeddings provided valuable visual representations but consumed significant memory and slowed similarity searches. Through PCA dimensionality reduction to 128 components, we achieved a 16x compression while retaining approximately 95% of variance. Benchmarking confirmed that query latency dropped below 1ms with minimal impact on recommendation quality.

8.2 Grayscale Training Limitations

We selected Fashion MNIST for its accessibility and well-defined category structure, which facilitated rapid prototyping and model validation. However, the dataset consists of grayscale images, meaning our ResNet-50 embeddings learned shape, texture, and structural features but not color information. This presents an important limitation: the system cannot recommend color-coordinated outfits based on learned representations. Alternative datasets such as DeepFashion, which contains over 800,000 RGB images with rich attribute annotations, would have enabled color-aware recommendations but required significantly more computational

resources and preprocessing. Future iterations could incorporate models trained on RGB fashion datasets to capture the full visual spectrum of clothing compatibility.

8.3 Asynchronous Database Integration

Integrating PostgreSQL with pgvector for embedding storage while using AsyncPG for asynchronous database operations presented compatibility challenges. The asyncio-based FastAPI backend required careful connection pool management and query optimization to prevent blocking operations during high-throughput embedding retrieval.

8.4 Multi-Container Orchestration

Coordinating six containerized services (FastAPI backend, Streamlit frontend, PostgreSQL, Prometheus, Grafana, and initialization scripts) via Docker Compose introduced complexity in startup ordering, health checks, and inter-service communication. Network configuration and volume management required iterative debugging to ensure consistent behavior across development and deployment environments.

8.5 Conclusion

This project demonstrates a functional end-to-end fashion recommendation system that combines computer vision, efficient similarity search, and intuitive user interface design. By leveraging ResNet-50 embeddings with FAISS indexing, we achieved sub-millisecond query latency while maintaining meaningful visual similarity matching. The full-stack architecture, spanning Streamlit frontend, FastAPI backend, PostgreSQL storage, and Prometheus/Grafana monitoring, provides a production-ready foundation for wardrobe management and outfit generation.

Key successes include the hybrid recommendation approach that balances algorithmic suggestions with user customization, responsive filtering by occasion and season, and a monitoring infrastructure that enables performance tracking in production. The iterative feedback process improved usability by removing confusing metrics and adding requested customization features.

Primary limitations stem from the grayscale training data, which restricts recommendations to structural and stylistic compatibility rather than color coordination. Future work could address this by retraining on RGB datasets such as DeepFashion, implementing user feedback loops to personalize recommendations over time, and exploring text-to-image models for aesthetic-based filtering.

References

Johnson, J., Douze, M., & Jégou, H. (2019). *Billion-scale similarity search with GPUs*. *IEEE Transactions on Big Data, 7*(3), 535–547. https://arxiv.org/abs/1702.08734

Kovalev, A. (2023). *pgvector: Open-source vector similarity search for PostgreSQL* [Software]. https://github.com/pgvector/pgvector

Liu, Z., Luo, P., Qiu, S., Wang, X., & Tang, X. (2016). *DeepFashion: Powering robust clothes recognition and retrieval with rich annotations*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1096–1104).

Xiao, H., Rasul, K., & Vollgraf, R. (2017). *Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms* (arXiv:1708.07747). *arXiv*. https://arxiv.org/abs/1708.07747