Engagement-weighted and style-aware scoring for fashion compatibility

Sally Lee
Stanford University
sunminl@stanford.edu

Melissa Liu
Stanford University
mliu1785@stanford.edu

Abstract

We enhance outfit compatibility prediction by extending the OutfitTransformer framework with two contributions: engagement-weighted loss and style-aware scoring. For the loss, we incorporate the number of likes an outfit received on Polyvore as a per-example weight in the focal loss, applying log-scaling and clipping to normalize the effect of highly popular outfits. For style-aware scoring, we use Sentence-BERT to embed outfit descriptions and style keywords into a shared space, allowing the model to adjust compatibility scores based on a user-specified style at inference time. We curate a Polyvore-based dataset containing outfit images, item descriptions, and likes, and design an effective negative sampling strategy by selectively replacing fashion items. Experimental results show that likes-weighted loss improves precision over the baseline by prioritizing widely liked outfits, while style-aware scoring demonstrates strong alignment with human judgments, particularly for outfits with clear and distinct stylistic themes. Our findings suggest that incorporating human engagement signals and stylistic conditioning can improve the personalization and interpretability of outfit recommendation systems.

1. Introduction

Fashion compatibility prediction is an emerging challenge in AI-powered retail, where the goal is to determine whether a group of clothing items forms a coherent and visually appealing outfit. This task is important because consumers increasingly expect personalized styling recommendations that reflect their tastes and the latest trends. We can leverage the growing availability of outfit-level interaction data, such as Polyvore outfits and how many likes they received, to align model predictions more closely with human preferences and stylistic intent.

Our motivation stems from the observation that existing models often treat all training examples equally and do not account for the subjective and style-driven nature of fashion. Yet in real-world settings, some outfits resonate more with users than others, and whether an outfit is going for a boho or minimalist look can dramatically shift how people assess its compatibility. To address this, we propose two enhancements to OutfitTransformer, the 2023 model by Sarkar et al. [11]. As discussed in the later Related Work section, OutfitTransformer uses a transformer encoder to compute a global embedding representation of an outfit in order to capture higher-order interactions between items in an outfit and outputs a scalar compatibility score $s \in [0, 1]$, where a higher score indicates higher compatibility of the outfit items [11]. We aim to modify this model so that:

- The training loss is weighted by the number of likes an outfit received on Polyvore.
- Compatibility scoring can be conditioned on an optional style keyword.

This approach bridges aesthetics with human engagement and honors stylistic nuances, allowing for more personalized and meaningful fashion recommendations. The inputs to our model at inference time are:

- An outfit, represented as an unordered set of 2 to 8 fashion items, each including an image and optional text description
- An optional style keyword, such as "boho" or "minimalist"

The model outputs are:

- A scalar compatibility score $s \in [0, 1]$
- A style-specific compatibility score $s_s \in [0, 1]$, if a style keyword is provided, representing how well that outfit fits the target style

2. Related Work

There were several approaches to generating compatibility scores in fashion recommendation systems.

2.1 Bi-LSTM

A Bidirectional Long Short-Term Memory (Bi-LSTM) network treats outfit items as sequential elements, similar to words in a sentence, and processes the outfit in both forward and backward directions, capturing dependencies among items [3]. Each item embedding is passed through the Bi-LSTM, and the hidden states are combined to predict a compatibility score for the outfit [3].

Although the Bi-LSTM model is simple and intuitive because it treats clothing items as a sequence, outfits inherently lack a natural or meaningful order. While Bi-LSTM captures local contextual relationships, it is less suited for the unordered structure of outfits; the sequential dependencies it assumes to exist and models may not be accurate [3]. Additionally, as outfit size grows, Bi-LSTM becomes harder to train and can suffer from vanishing gradients, limiting its scalability for larger or more complex outfits [3].

2.2 Pairwise Comparisons

Several approaches generate outfit compatibility scores by explicitly computing pairwise relationships between items and aggregating them. For instance, Wang et. al's Multi-Layered Comparison Network (MCN) extracts hierarchical features with CNNs and Global Average Pooling (GAP) to compare items at multiple semantic levels, such as color, texture, and style [16]. The model learns compatibility from type-specified pairwise similarities and uses backpropagation gradients to diagnose incompatible item pairs [16]. Type-aware embedding methods can also be used to project general image embeddings into type-specific compatibility spaces, allowing the model to learn specialized embeddings for different item type pairs, such as top-shoes or top-bottom [12]. Lu et. al's Multi-layer Non-Local Feature Fusion (MNLFF) framework enhances pairwise comparison by combining visual features across different CNN layers to incorporate both low-level and high-level attributes [8].

Pairwise comparison methods generate outfit compatibility scores by explicitly comparing all possible item pairs and aggregating their relationships. One major strength of this approach is its fine-grained modeling of item interactions, allowing the model to directly assess how individual pairs of items contribute to overall outfit compatibility. However, these methods assume that outfit compatibility can be fully decomposed into independent pairwise relations, ignoring higher-order or global outfit coherence. Thus struggle to capture the holistic aesthetic of outfits. Additionally, as outfit size increases, the number of item pairs grows quadratically, leading to scalability challenges and increased computational costs.

2.3 Graph-Based Approaches

Graph-based methods predict outfit compatibility by modeling outfits as graphs, converting the compatibility scoring task into a graph inference problem. A node-wise Graph Neural Network (NGNN), for instance, represents each fashion item category as a node and item interactions as edges [1]. Two directed edges for each pair of categories are used to capture asymmetric relationships, reflecting that compatibility can differ based on interaction direction; matching socks to shoes differs from matching shoes to socks [1]. Meanwhile, Vivek et. al's dot-attention Graph Neural Network uses dot-product attention to compute the weight between nodes, constructing outfit graphs where nodes are fashion items and edges model visual relationships [13]. Wang et. al's unified representation model constructs a graph that integrates visual-semantic information to predict compatibility and models the latent dependencies among outfit items using graph attention networks [14]. Li et al. also proposed OCPHN, which represents outfits as hypergraphs; they extend graphs to model interactions between multiple categories simultaneously and employ hypergraph convolution to learn refined node representations [6].

A key advantage of graph-based approaches is their ability to capture complex, high-order relationships among multiple outfit items beyond simple pairwise comparisons, leading to more holistic compatibility evaluation. However, a notable weakness is the high computational cost, especially at inference time when new items are introduced into the catalog, as reconstruction of potentially large and dynamic graphs is required.

2.4 Transformer-Based Approaches

Transformer-based models are considered the state-of-the-art for outfit compatibility prediction. They model outfits as unordered sets and capture complex item relationships through self-attention. Sarkar et al. proposed OutfitTransformer, which uses a Transformer encoder with a learnable outfit token to produce a global outfit embedding, effectively capturing higher-order item interactions [11]. Jung et. al's History-Aware Transformer (HAT) extends this architecture with a second Transformer to incorporate user purchase history, enabling personalized compatibility predictions [5]. More recently, Firmansyah et al. also proposed Slay-Net enhanced Transformer-based outfit recommendation, which introduces a curriculum learning strategy that trains the model in stages; combined classification and contrastive objectives on easier examples are used first, followed by fine-tuning on harder contrastive tasks [2].

Overall, Transformer-based models excel at capturing global outfit structure and higher-order item dependencies through attention mechanisms. However, their high model capacity typically demands large training datasets and sig-

nificant computational resources.

3. Methods

3.1 Baseline: OutfitTransformer

The OutfitTransformer models an outfit as an unordered set of items, each embedded via a pretrained image encoder (ResNet-18) and optional text encoder (Sentence-BERT) [11]. A special "outfit token" is prepended to the sequence of item embeddings, and a transformer encoder processes the entire set. The final representation of the outfit token is used to predict a compatibility score via an MLP. The model is trained using focal loss, which is a modification of cross-entropy loss first used by Lin et al. to down-weight easy examples and focus on harder ones for binary classification [7]. The focal loss for a single example i is

$$\mathcal{L}_i(p_i, y_i) = \begin{cases} -\alpha(1-p_i)^\gamma \log(p_i), & \text{if } y_i = 1 \\ -(1-\alpha)p_i^\gamma \log(1-p_i), & \text{if } y_i = 0 \end{cases}$$

where y_i is an indicator variable for whether outfit i is a ground-truth compatible outfit, p_i is the MLP head's output predicted probability that outfit i is compatible, $\alpha \in [0, 1]$ is a weighting factor to handle class imbalance, and $\gamma \geq 0$ is a parameter to down-weight easy examples [7]. Note that for easy examples, either p_i is large for $y_i = 1$ or p_i is small for $y_i = 0$ [7].

3.1 Likes-Weighted Loss

We weight each training example's focal loss by the number of likes that outfit received to reflect human preferences, inspired by previous work that scales the loss contribution of each example by how informative it is [10, 15]. Specifically, the loss for the ith outfit becomes

$$\mathcal{L}_{i_{weighted}}(p_i, y_i) = w_i \cdot \mathcal{L}_i(p_i, y_i)$$

where w_i is the weight of the ith example. This emphasizes high-like outfits more during training, encouraging the model to learn compatibility patterns aligned with popular and widely approved outfits. To compute w_i, we use

$$w_i = 1 + a \cdot \log(1 + l_i/b)$$

where l_i is the number of likes received by the ith outfit; Hu et. al used this equation to assign confidence scores to training examples [4]. This technique is helpful for our task because log scaling dampens the impact of high-like outfits, preventing a few examples with a disproportionately high number of likes from exerting too much influence on the loss [4]. Additionally, the 1 in the log ensures $\log(1 + l_i/b)$ is always non-negative, thus allowing the 1 at the front to give all examples $w_i \geq 1$. Note that an example with 0 likes recieves $w_i = 1$. We attempt hyperparameter tuning of a and b in the Experiments and Results section.

To accommodate the new data format required for likes-weighted loss, we wrote three new classes for 1) a single example with likes, 2) an entire dataset of examples with likes, and 3) a query for items from an outfit with likes. We also had to code a new collate function and a new loss function.

3.2 Style-Aware Scoring via Title Embedding Similarity

To incorporate style awareness, we extended the original testing pipeline with an additional scoring module. This extension is implemented on top of the baseline testing file without altering the original training or model architecture.

Manual judgment of an outfit's style is inherently challenging, as titles are often user-generated phrases such as "Boho Chic Fall Look" or "Minimalist Streetwear", which encode stylistic intent in an implicit and inconsistent manner. Consequently, it is difficult to objectively determine an outfit's compatibility with specific styles. To address this, our code introduces a flexible scoring mechanism that allows the user to specify a style keyword from a predefined list of 20 popular fashion styles, such as "casual", "boho", or "goth", to obtain a style-aware compatibility score. This list of style keywords is hardcoded in the script to ensure consistency in comparisons across different outfits.

For style-aware scoring, we utilized the Sentence-BERT model by Reimers and Gurevych [9], pretrained on large-scale language data for semantic similarity tasks. We load Sentence-BERT during inference to encode each outfit's textual information into a continuous vector, referred to as the style embedding. This dense semantic representation is intended to capture the stylistic characteristics embedded in the outfit's description without requiring retraining.

The item metadata used for generating style embeddings is structured as follows:


```
{
    "item_id": 2124272891,
    "url_name": "christmas print tee",
    "description": "A fashion look
    by beebeely-look...",
    "categories": [
      "Clothing",
      "T-Shirts"
    ],
    "title": "christmas print tee",
    "related": [
      "t-shirts"
    ],
    "category_id": 21,
    "semantic_category": "t-shirts"
  },
```


At inference time, our code extracts and concatenates all

item-level metadata, specifically the urlname, description, and title fields, into a unified text string. This text is then fed into Sentence-BERT to produce the outfit embedding. We similarly encode style keywords, mapping both outfits and style labels into the same embedding space to allow direct comparison. Then, this yields embeddings e_{outfit} and e_{keyword}.The raw similarity is computed via cosine similarity:

$$\cos(e_{\text{keyword}}, e_{\text{outfit}}) = \frac{e_{\text{keyword}} \cdot e_{\text{outfit}}}{\|e_{\text{keyword}}\| \, \|e_{\text{outfit}}\|}$$

However, raw cosine similarities range from $(-1, 1)$ and are not directly comparable across different outfits or styles. To address this, the code computes cosine similarities between the outfit embedding and all 20 predefined style embeddings. It then normalizes the selected keyword's similarity relative to the minimum and maximum similarities across all styles for that outfit:

$$\text{normalized\_score} = \frac{c(e_q, e_o) - \min_k c(e_k, e_o)}{\max_k c(e_k, e_o) - \min_k c(e_k, e_o) + \varepsilon}.$$

where $c(x, y)$ is a cosine similarity between x ad y and ε is a small constant for numeric stability. While alternative normalization techniques exist—such as global min-max normalization or softmax scaling—they come with trade-offs. Global min-max normalization applies a uniform linear scaling to all outfits but may fail to differentiate styles for outfits with tightly clustered similarities. Softmax scaling amplifies differences but sacrifices interpretability, which is crucial for user-facing applications. Relative normalization offers contextual scaling and maintains interpretability, making it the most suitable choice for our task.

Then, we obtain the style specific score

$$s_s = s \times \text{normalized\_score}$$

This approach enables the same outfit to receive different compatibility scores depending on the user-specified style. For instance, an outfit described as "Cozy Winter Loungewear" might have $s = 0.8$, but ultimately receive a higher score of $s_s = 0.72$ for the "casual" prompt and a lower score of $s_s = 0.36$ for "boho" or $s_s = 0.21$ for "formal".

4. Dataset and Features

Polyvore.com was a popular online fashion platform where users could create and upload virtual outfits and like other users' outfits, until it was discontinued in 2018.

OutfitTransformer was run on preprocessed Polyvore data that excludes the number of likes and outfit description, which are critical for our likes-weighting and style-specific scoring enhancements, respectively [11]. Additionally, the OutfitTransformer Github repo did not contain the negative examples, nor did the paper detail how they generated negative examples [11]. Thus, we chose to use the Polyvore dataset curated by Han et al. for Bi-LSTM; it contains 22,617 complete outfits created by Polyvore users. Each outfit consists of 2–8 fashion items, with item level images, category labels, outfit-level metadata, and textual titles [3]. The dataset excludes outfits curated prior to 2014 due to concerns that they are outdated [3]. Additionally, it keeps only the first eight fashion items for outfits containing too many items and deletes any items with non-fashion "categoryid" such as background, text, and decorations [3]. The train / validation / test split is 17,316 / 1,497 / 3,804 [3].

We pre-processed the Bi-LSTM data into json files that fit the format used by OutfitTransformer. To ensure our images aligned as closely as possible with the OutfitTransformer images, we computed the mean and standard deviation of the RGB channel values for the pixels of all OutfitTransformer item images and normalized the Bi-LSTM item images by the mean and standard deviation for each RGB channel. Though the original item images vary in pixel dimensions, the OutfitTransformer model includes a script that resizes all images to 224x224 pixels before using them as inputs. Each example outfit is formatted as such:

```
{
    "label": 1,
    "question": [1201612711, 1201612712,
    1201612713, 1201612714, 1201612715,
    1201612716, 1201612717, 1201612718],
    "likes": 9,
    "desc": "A fashion look from April
    2014 featuring destroyed shirt,
    skinny fit jeans and vans shoes.
    Browse and shop related looks."
}
```

where "question" corresponds to a list of item IDs, each of which is referenced to obtain an image of that item. As for the negative examples required for focal loss, we replaced randomly selected items in each positive outfit with random items of the same "categoryid" to create its corresponding negative outfit. Further details are included in the Experiments and Results section. For instance, here is the negative outfit corresponding to the positive outfit above:

```
{
    "label": 0,
    "question": [1685198071, 1201612712,
    1961414984, 1419047385, 1763980522,
    2160255475, 1201612717, 2167947836]
}
```

A "label" of '1' indicates a positive outfit, while a "label" of '0' indicates a negative outfit. Note that six of the eight

Figure 1. Item images for example outfit.

Figure 2. Item images for example negative outfit.

original items have been randomly replaced; only the second and seventh items remain the same.

5. Experiments and Results

5.1 Generating Negative Examples

We experimented with different methods to generate negative examples. First, we tried replacing $min(k, n)$ items in with random items of the same "categoryid", where n is the number of items in a specific outfit, for $k = 3$, $k = 5$, and $k = 7$, $min(k, n)$. The items were randomly chosen to be replaced. $k = 7$ performed the best out of these three trials with a validation f1 score of 0.399 compared to 0.058 and 0.289 for $k = 3$ and $k = 5$, respectively (Table 1). These results suggest that when we only replace a few items, the negative outfits generated are not incompatible enough compared to the original outfits. Thus, the model might have found it difficult to learn the distinction between the positive and negative examples.

Next, we tried replacing $min(k, n - 2)$ random items for $k = 7$; our reasoning was that under plain $k = 7$, all items would be replaced for outfits with fewer than 8 items, resulting in negative outfits that were too random for the model to learn from. This trial, labeled as $k = 7^*$, achieved a validation f1 score of 0.537, which was higher than plain $k = 7$ (Table 1).

Finally, we tried replacing the last $min(k, n - 2)$ items listed for $k = 7$ in a trial labeled $k = 7^{**}$. For instance, if an outfit contained a shirt, pants, shoes, and two bracelets, we would replace the 5-2=3 items, which are the shoes and two bracelets; the reasoning behind this final trial was that the items most integral to the outfit's style and cohesiveness are typically listed first. The validation f1 score for $k = 7^{**}$ was 0.396, which was worse than $k = 7^*$ (Table 1). This may be because the distinction between the original outfits and the negative outfits produced by $k = 7^{**}$ was too small since we kept the most important items the same. Thus, we decided to use $k = 7^*$ in the end.

Table 1. Likes-weighted model epoch 8 performance when trained on negative examples with various generation methods.

k	Train Loss	Val Loss	Train F1	Val F1
3	0.086	0.109	0.417	0.058
5	0.064	0.105	0.667	0.289
7	0.054	0.093	0.745	0.399
7^*	0.069	0.073	0.642	**0.537**
7^{**}	0.070	0.091	0.639	0.396

5.2 Likes-Weighted Loss

We attempted hyperparameter tuning for the likes-weighted model, testing $a = 0.5, 1.0, 1.5$ and $b = 10, 100,$ and1000 for a and b in the w_i equation provided in the Methods section. We used the default settings of $\alpha = 0.5$ and $\gamma = 2.0$ for the hyperparameters in the focal loss equation, rather than tuning then since Sarkar et. al already found these to be the optimal settings [11].

The highest performing model was $a = 1.5$, $b = 10$ at epoch 11, with validation f1 score of 0.713 (Table 2). Larger a increases the difference in loss contribution between highly liked and less liked examples, amplifying the influence of likes more. Smaller b makes the log function more sensitive to changes in like count, especially at the lower end, so that 1 like vs. 10 likes might cause a large difference in weight. Thus, these results suggest that examples with higher likes are particularly valuable for learning compatibility and that the best performing weighting scheme should give high-like outfits significantly more importance, while also reflecting small increases in like count.

Additionally, $a = 1.0$, $b = 10$ performed almost as well $a = 1.5$, $b = 10$, with validation f1 score of 0.707 at epoch 10; this indicates that that the optimal value for a might lie between 1.0 and 1.5, and further tuning could yield better results (Table 2).

The $a = 1.0$, $b = 10$ likes-weighted model had the highest validation f1 score of 0.794 at epoch 27, while the baseline model had the highest validation f1 score of 0.777

at epoch 39. Thus, we ran these epoch 27 likes-weighted model and epoch 39 baseline model on the test data (Table 3).

The likes-weighted model had higher test accuracy and precision (0.619 and 0.614 compared to the baseline's 0.615 and 0.592), indicating that it is better at avoiding false positives and makes more correct predictions overall (Table 3). However, it had lower recall and f1 (0.642 and 0.628 compared to the baseline's 0.740 and 0.658), suggesting that it misses more true positives and has less balanced performance (Table 3). Overall, these results tell us we can more more confident in an outfit deemed compatible by the likes-weighted model compared to an outfit deemed compatible by the baseline, but the likes-weighted model may be too conservative and noninclusive in labeling compatibility.

From the plateauing validation f1 scores, we see that the baseline begins overfitting around epoch 40, while the likes-weighted model begins overfitting a bit earlier around epoch 30 (Figure 3). Overfitting is also evident in the slightly increasing validation loss starting around epoch 25 for both models (Figure 3). The likes-weighted validation loss is more erratic and remains significantly higher than the baseline validation loss, indicating poorer generalization (Figure 3). The more aggressive overfitting and poor generalization of the likes-weighted model may be because the weighted loss function encourages it to put too much emphasis on a few examples.

Table 2. Likes-weighted model performance with different values of a and b; epoch with the highest validation f1 score up to epoch 15 chosen.

a	b	Train Loss	Val Loss	Train F1	Val F1	Epoch
0.5	10	0.086	0.123	0.773	0.703	12
0.5	100	0.058	0.090	0.798	0.680	15
0.5	1000	0.055	0.078	0.762	0.607	11
1.0	10	0.105	0.166	0.774	0.707	10
1.0	100	0.074	0.137	0.778	0.591	11
1.0	1000	0.054	0.085	0.782	0.611	12
1.5	10	0.117	0.230	0.776	**0.713**	11
1.5	100	0.087	0.141	0.766	0.664	10
1.5	1000	0.064	0.087	0.751	0.626	10

Table 3. Test performance of baseline model with highest validation f1 score versus $a = 1.0$, $b = 10$ likes-weighted model with highest validation f1 score.

Model	Accuracy	Precision	Recall	F1
Baseline (epoch 39)	0.615	0.592	**0.740**	**0.658**
Likes-weighted (epoch 27)	**0.619**	**0.614**	0.642	0.628

5.3 Style-Aware Scoring

We applied style-aware scoring to the test set, computing compatibility scores between each outfit and 20 predefined style keywords. Table 4 presents examples of the results, showing each outfit's original compatibility score and the normalized style-aware scores for a selection of style keywords. By examining the normalized scores, we can identify the style keyword with the highest compatibility for each outfit. For instance, outfit 3 achieves the highest score with "business casual" (1.00) and outfit 5 with "cottagecore" (1.00).

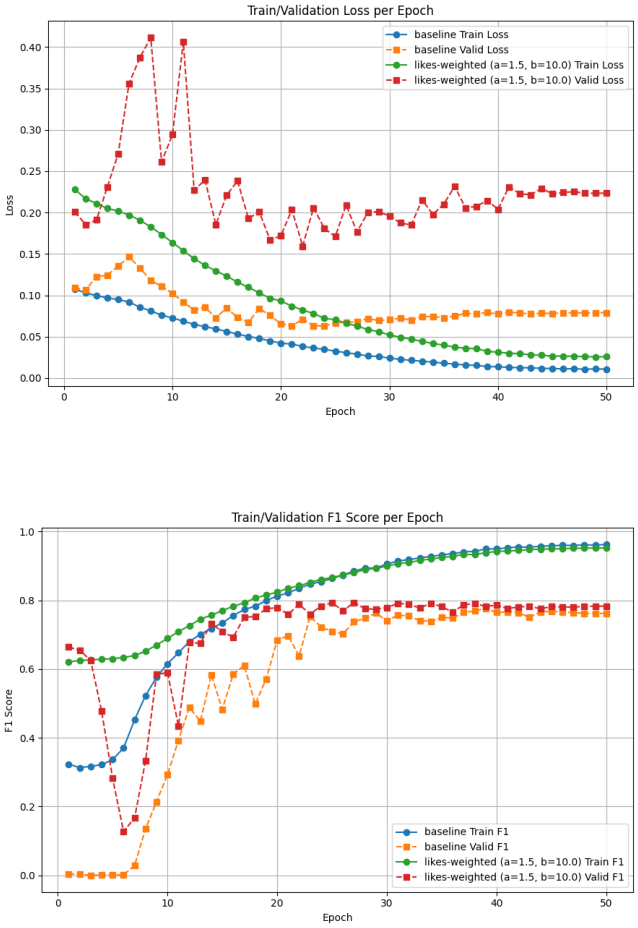

Figure 3. Training and validation loss and f1 scores for baseline and highest-performing likes-weighted model over 50 epochs.

Outfit	Original	Boho	Business Casual	Casual	Chic	Cottage core
1	0.40	0.14	0.35	0.36	0.27	0.07
2	1.00	0.29	0.81	0.62	0.57	0.40
3	1.00	0.58	1.00	0.75	0.29	0.61
4	1.00	0.47	0.11	0.12	0.38	0.89
5	1.00	0.34	0.78	0.82	0.36	1.00

Table 4. Five examples of style-aware scoring (rounded to the hundredth; only 5 styles shown for clarity). Style-aware scores are normalized cosine similarities between outfit descriptions and style keywords.

Outfit	Style keyword with highest compatibility score	Subject 1	Subject 2	Subject 3
	Preppy	Business Casual	Chic	Chic
	Punk	Punk	Casual	Punk
	Sporty	Sporty	Sporty	Sporty
	Goth	Chic	Preppy	Chic

Figure 4. Four outfits with the style keyword of highest compatibility score predicted by the model and the style keywords selected by three human subjects.

5.3.1 Evaluation with Human Subjects

To assess the validity of the style-aware scores, we selected 20 outfits and recorded the highest-scoring style keyword for each outfit according to the model. Three human subjects independently reviewed the same outfits and selected the style keyword they believed best described each one. When comparing the model's top predictions with the human responses, we found notable patterns. For 10 outfits, all three subjects unanimously selected the same style keyword, and the model's highest-scoring style typically matched this unanimous human choice. For instance, in one case, all subjects selected "sporty" as the most fitting description, and the model also ranked "sporty" highest for the same outfit (Figure 5).

However, there were also cases where the subjects disagreed on the appropriate style. In four instances, the model's prediction still aligned with the majority opinion among the subjects. For example, for outfit 2, two subjects labeled the outfit as "punk" while one chose "casual"; the model similarly selected "punk" as the top style. In six instances, the model's prediction diverged from all subject responses. For example, for outfit 4, no subject chose "goth," but the model assigned the highest compatibility score to "goth" (Figure 5).

5.3.2 Error Analysis and Insights

We further analyzed the disagreement cases to better understand the limitations of the style-aware scoring approach. One major factor was that some outfits naturally fit multiple styles, making the labeling inherently ambiguous. For instance, outfit 2 could reasonably be interpreted as both "casual" and "punk" (Figure 5), leading to variation in human judgments and possible divergence from the model's prediction.

Another limitation stems from the model's reliance solely on textual metadata, such as the title and description, without access to visual information like color, material, or silhouette. This lack of visual cues can lead to misclassification when key stylistic elements are omitted from the text. For example, if a hot pink color—often associated with "chic" style—is not mentioned in the metadata, the model might fail to recognize the outfit's alignment with that style as outfit 1 (Figure 5).

Finally, the Sentence-BERT model itself introduces potential biases by overemphasizing certain words during embedding generation. Specific phrases in the metadata can disproportionately influence the resulting embedding. For instance, in the case of outfit 4, the description included the phrase "metal stone finish," which may have caused the model to focus on these words and incorrectly assign the "goth" label, even though the overall aesthetic of the outfit was not goth-like (Figure 5).

Outfit	Original	Subject 1	Subject 2	Subject 3
Outfit 1	1.00	0.68	0.65	0.65
Outfit 2	1.00	1.00	0.75	1.00
Outfit 3	0.10	0.10	0.10	0.10
Outfit 4	1.00	0.57	0.65	0.57

Table 5. Comparison of the original compatibility score with style-aware scores corresponding to human-selected style keywords for each outfit.

5.3.3 Alignment Between Model Scores and Human-Selected Styles

To further evaluate the model's style-aware scoring, we analyzed how well the model scored the specific style keywords selected by human subjects. For each outfit, we recorded the model's compatibility score for the style keyword chosen by each subject (Table 5).

Our analysis reveals several important trends. First, for outfits where human subjects largely agreed, the model tended to assign relatively high scores (0.65–1.00) to the selected styles, even if it did not always rank them as the highest. For example, Outfit 2 received perfect scores (1.00) for two of the subjects' chosen styles and a moderately high score (0.75) for the third, demonstrating strong model-human alignment. Second, in more ambiguous cases, the model still showed moderate agreement. For Outfit 4, where the subjects' opinions varied, the model assigned scores around 0.57–0.65 to the chosen styles, indicating partial recognition of stylistic relevance.

6. Conclusion

In this work, we have introduced likes-weighted loss and style-aware scoring to enhance outfit compatibility prediction.

The likes-weighted model is too cautious in labeling outfits as compatible, suffering from overfitting on a few high-like examples and poor generalization; it rewards popular outfits, possibly at the expense of their more niche counterparts. For applications that seek to prioritize mainstream outfit compatibility, the likes-weighted model may be more reliable. For inclusive outfit recommendations that accept a more diverse array of styles, however, the baseline is more valuable. A future avenue to explore is alternative functions to compute the outfit weights that mitigate the overfitting and lack of outfit inclusivity promoted by the current weighting scheme. As discussed in the Experiments and Results section, we would also attempt further, more focused, hyperparameter tuning, especially to determine the optimal setting for 'a'.

For style-aware scoring, our evaluation showed that even without style conditioning during training, the model can achieve reasonable alignment between style-aware scores and human-annotated style labels. For outfits where human subjects agreed on the stylistic label, the model often produced high scores for the corresponding style keywords, demonstrating strong model-human agreement. The normalization step further helped to emphasize the dominant style per outfit, although in some cases it may have masked the multi-style nature of certain outfits. For future work, we would like to explore alternative normalization techniques that allow an outfit to be recognized as fitting multiple styles simultaneously. Instead of enforcing a single dominant style, we aim to develop scoring strategies that enable high scores for multiple stylistically appropriate keywords.

7. Contributions and Acknowledgements

Sally Lee worked on data augmentation, negative example generation, and Sentence-Bert embeddings for style-specific scores. She helped draft related work, methods, results, and conclusion and revised introduction and data.

Melissa Liu worked on data augmentation, negative example generation, and likes-weighted focal loss. She helped draft introduction, data, methods, results, and abstract and revised related work.

Thanks to the Sarkar et. al for their ideas and Outfit-Transformer code repo.

References

[1] Z. Cui, Z. Li, S. Wu, X. Zhang, and L. Wang. Dressing as a whole: Outfit compatibility learning based on node-wise graph neural networks. *arXiv preprint arXiv:1902.08009*, 2019.

[2] I. Firmansyah, R. Scholz, A. Nahmendorff, N. S. Le, S. Elsayed, and L. Schmidt-Thieme. Learning set embeddings for fashion compatibility recommendation. In *Strategic and Utility-aware Recommendations (SURE) Workshop @ RecSys 2024*. CEUR Workshop Proceedings, 2024. To appear.

[3] X. Han, Z. Wu, Y.-G. Jiang, and L. S. Davis. Learning fashion compatibility with bidirectional lstms. In *Proceedings of the 2017 ACM on Multimedia Conference (MM)*, pages 1078–1086, Mountain View, CA, USA, 2017. ACM.

[4] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM)*, pages 263–272, Pisa, Italy, 2008. IEEE. Work done while Koren was at AT&T Labs – Research.

[5] M. C. Jung, J. Monteil, P. Schulz, and V. Vaskovych. Personalised outfit recommendation via history-aware transformers. *arXiv preprint arXiv:2407.00289*, 2024.

[6] Z. Li, J. Li, T. Wang, X. Gong, Y. Wei, and P. Luo. Ocphn: Outfit compatibility prediction with hypergraph networks. *Mathematics*, 10(20):3913, 2022.

[7] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(12):318–327, 2017. Presented at ICCV 2017.

[8] S. Lu, X. Zhu, Y. Wu, X. Wan, and F. Gao. Outfit compatibility prediction with multi-layered feature fusion network. *Pattern Recognition Letters*, 147:150–156, 2021.

[9] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[10] M. Ren, W. Zeng, B. Yang, and R. Urtasun. Learning to reweight examples for robust deep learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 4334–4343. PMLR, 2018.

[11] R. Sarkar, N. Bodla, M. I. Vasileva, Y.-L. Lin, A. Beniwal, A. Lu, and G. Medioni. Outfittransformer: Learning outfit representations for fashion recommendation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Vancouver, Canada, 2023. IEEE.

[12] M. I. Vasileva, B. A. Plummer, K. Dusad, S. Rajpal, R. Kumar, and D. Forsyth. Learning type-aware embeddings for fashion compatibility. *arXiv preprint arXiv:1803.09196*, 2018.

[13] B. S. Vivek, G. Bhattacharya, J. Gubbi, B. L. V, A. Pal, and P. Balamuralidhar. Personalized outfit compatibility prediction using outfit graph network. *IEEE Access*, 2023.

[14] J. Wang, X. Cheng, R. Wang, and S. Liu. Learning outfit compatibility with graph attention network and visual-semantic embedding. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2021.

[15] X. Wang. *Example Weighting for Deep Representation Learning*. Doctor of Philosophy, Queen's University Belfast, Belfast, United Kingdom, 2020. `https://pure.qub.ac.uk/en/studentTheses/example-weighting-for-deep-representation-learning`.

[16] X. Wang, B. Wu, Y. Ye, and Y. Zhong. Outfit compatibility prediction and diagnosis with multilayered comparison network. In *Proceedings of the 27th ACM International Conference on Multimedia (MM '19)*, pages 329–337, Nice, France, 2019. Association for Computing Machinery.