StyleSavvy Final Presentation

By: Uzair, Vidit, Sakshi, Esha

INTRODUCTION

Introduce the project

01

DESIGN

Discuss the technologies and design decisions

02

IMPLEMENTATION

Show the implementation of the application

03

OPTIMIZATION

Areas of future work/improvement

04

RESULTS & LESSONS LEARNED

Overall results and lessons learned

05

01

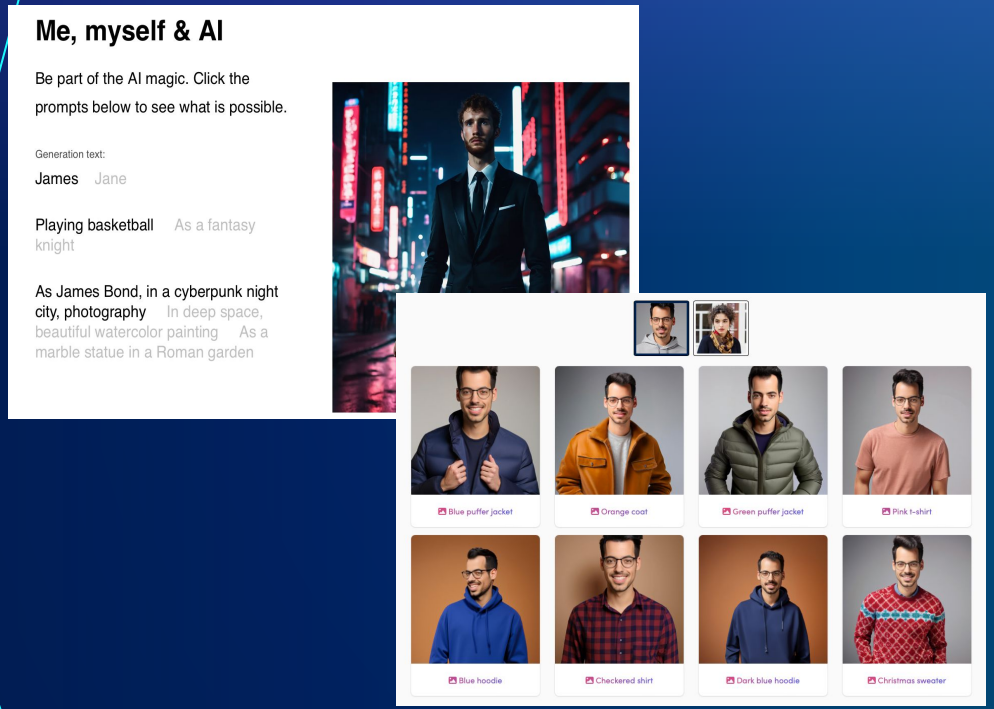
INTRODUCTION

PROJECT INTRODUCTION

Motivation: Often times we have a lot of clothes and don't know how to style them or need ideas for what would look good. This mobile application allows users to upload a photo using their devices camera and then input a name of a person and then recommends a similar, aesthetic outfit .

Concept: Develop an application that can detect the clothes inputted and provide an outfit idea with Amazon links.

RELATED WORK & NOVELTY



Machine Learning has been used by others to achieve a similar goal as our proposed solution. However, to the best of our knowledge, our approach for recommending clothing items and accessories to users is the only one that not only takes into account the user's current wardrobe, but also makes the recommendation based on the co-occurrence of accessories in the form of fashionable and aesthetic outfits. To do this, we will develop a Clothing Segmenter, Accessory Recommender, and an Image Scraper.

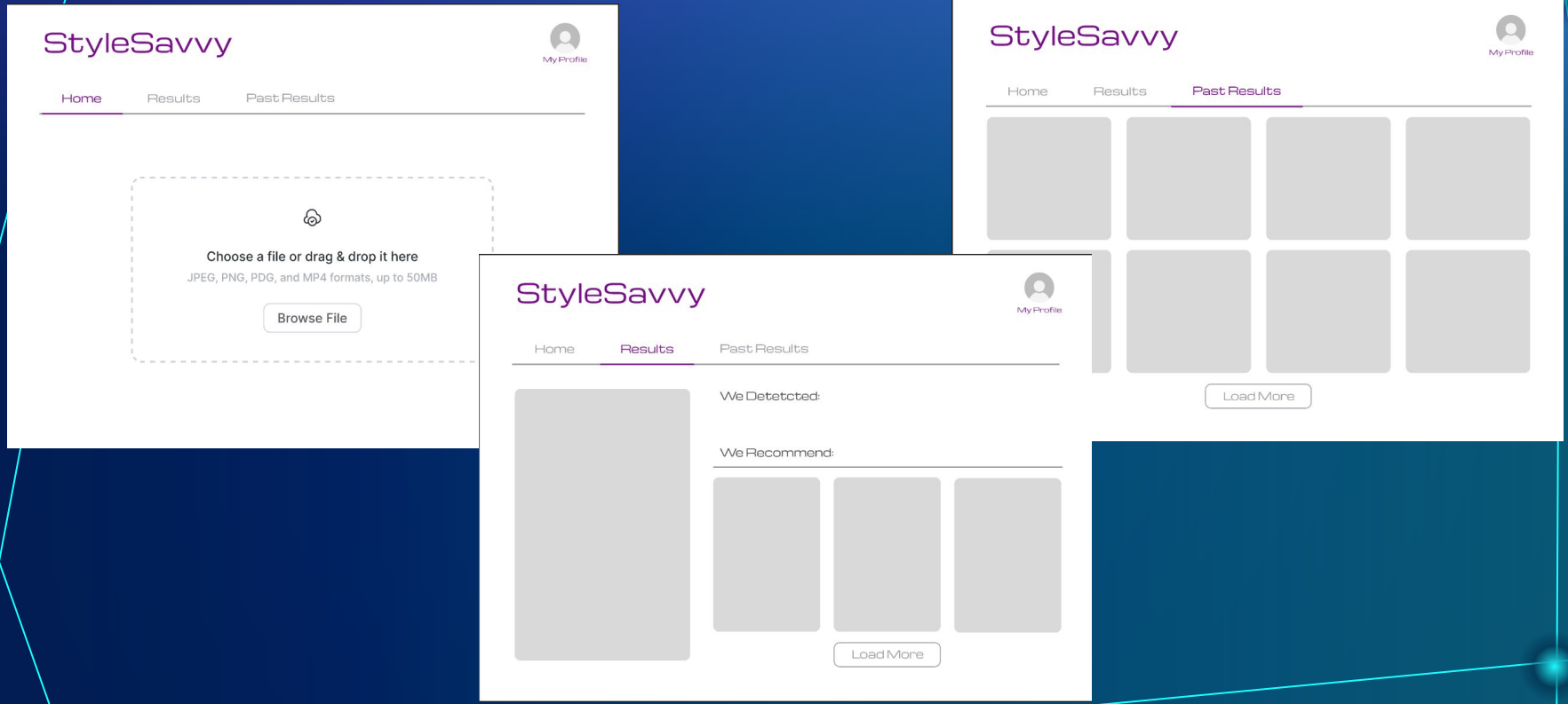
02 DESIGN

DESIGN

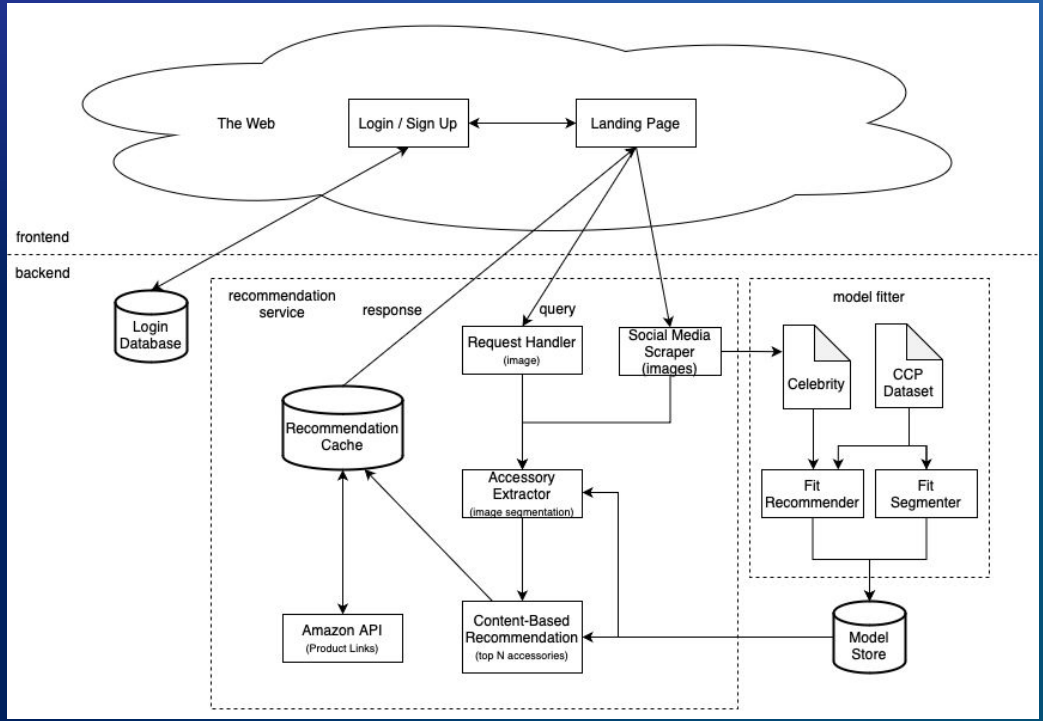
Goal: The goal of the project is to create a full stack application that would take images of the owned clothes and some text and output potential outfits

The project relies on a machine learning model to understand the image and then searches the internet for outfits with those items and then outputs the missing fashion pieces back to the user

UI MOCKUPS


StyleSavvy
My Profile
Home
Results
Past Results
Choose a file or drag & drop it here
JPEG, PNG, PDG, and MP4 formats, up to 50MB
Browse File
StyleSavvy
My Profile
Home
Results
Past Results
Load More
StyleSavvy
My Profile
Home
Results
Past Results
We Detetcted:
We Recommend:
Load More

ARCHITECTURE



03

IMPLEMENTATION

Fashion-Based Recommendations

- Recommendation engines do not take into account:
  - Items that may **complement user's current wardrobe/attire**.
  - Items that may nudge user's style towards **target fashion icons**.

Fashion-Based Recommendations

- Recommendation engines do not take into account:
  - Items that may **complement user's current wardrobe/attire**.
  - Items that may nudge user's style towards **target fashion icons**.

- We propose a two-step solution:
  - **Baseline recommender** which leverages user's wardrobe.
  - Mechanism to slightly **bias model** towards user's target icons.

Fashion-Based Recommendations

- Recommendation engines do not take into account:
  - Items that may **complement user's current wardrobe/attire**.
  - Items that may nudge user's style towards **target fashion icons**.

- We propose a two-step solution:
  - **Baseline recommender** which leverages user's wardrobe.
    - Detect user's wardrobe => **Segmentation**!
    - Recommend complementary item => **Co-Occurance**!
  - Mechanism to slightly **bias model** towards user's target icons.
    - Look at target's wardrobe!

Clothing-Co-Parsing* (CCP) Dataset

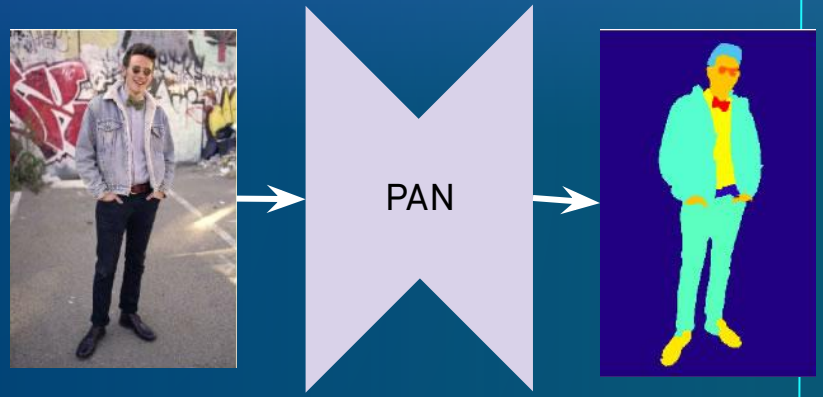
- **~1k** images of:
  - Fashionable outfits and
  - their pixel-level annotations.

- Will be used for learning:
  - Segmentation.
  - Co-Occurances.

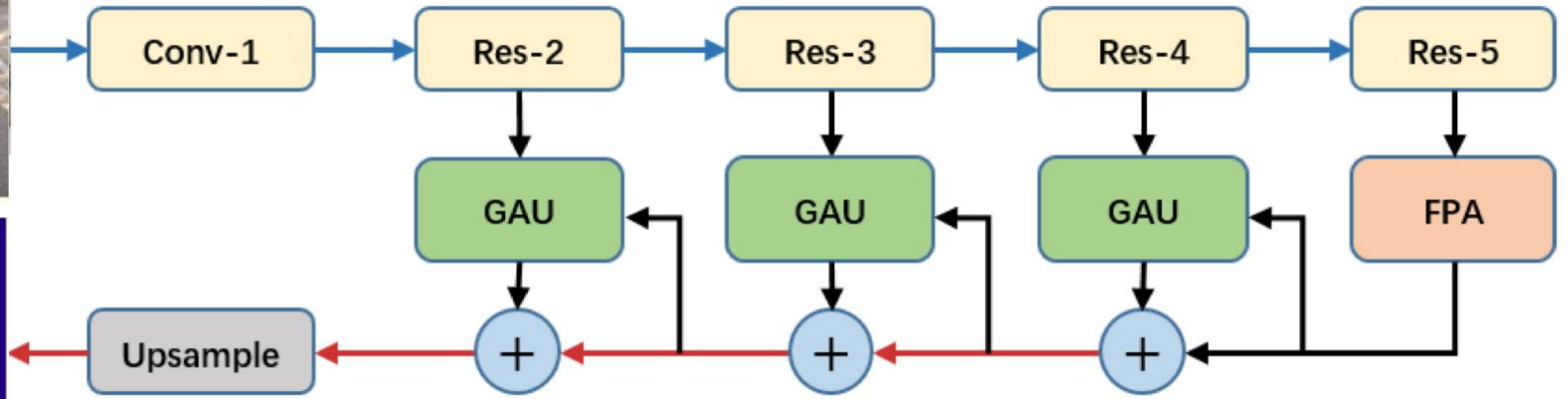
* https://github.com/bearpaw/clothing-co-parsing

CCP Segmentation; Implementation

- Learning a segmenter is possible by fitting a neural network to CCP data:
  70, 20, 10 split of train, validation, test data
- We selected the **Pyramid Attention Network (PAN)** architecture with imagenet initialization.



CCP Segmentation; Implementation



PAN architecture uses Resnet & Attention mechanism for better pixel-wise prediction*.

* https://arxiv.org/pdf/1805.10180

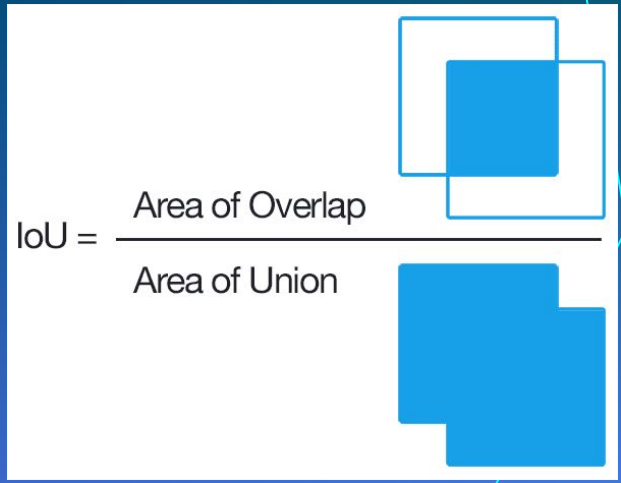
CCP Segmentation; Metrics

- **Intersection over Union (IoU):**
  Tries to capture the similarity of a set of pixels A and B by computing

  $$|A \cap B| \,/\, |A \cup B|$$

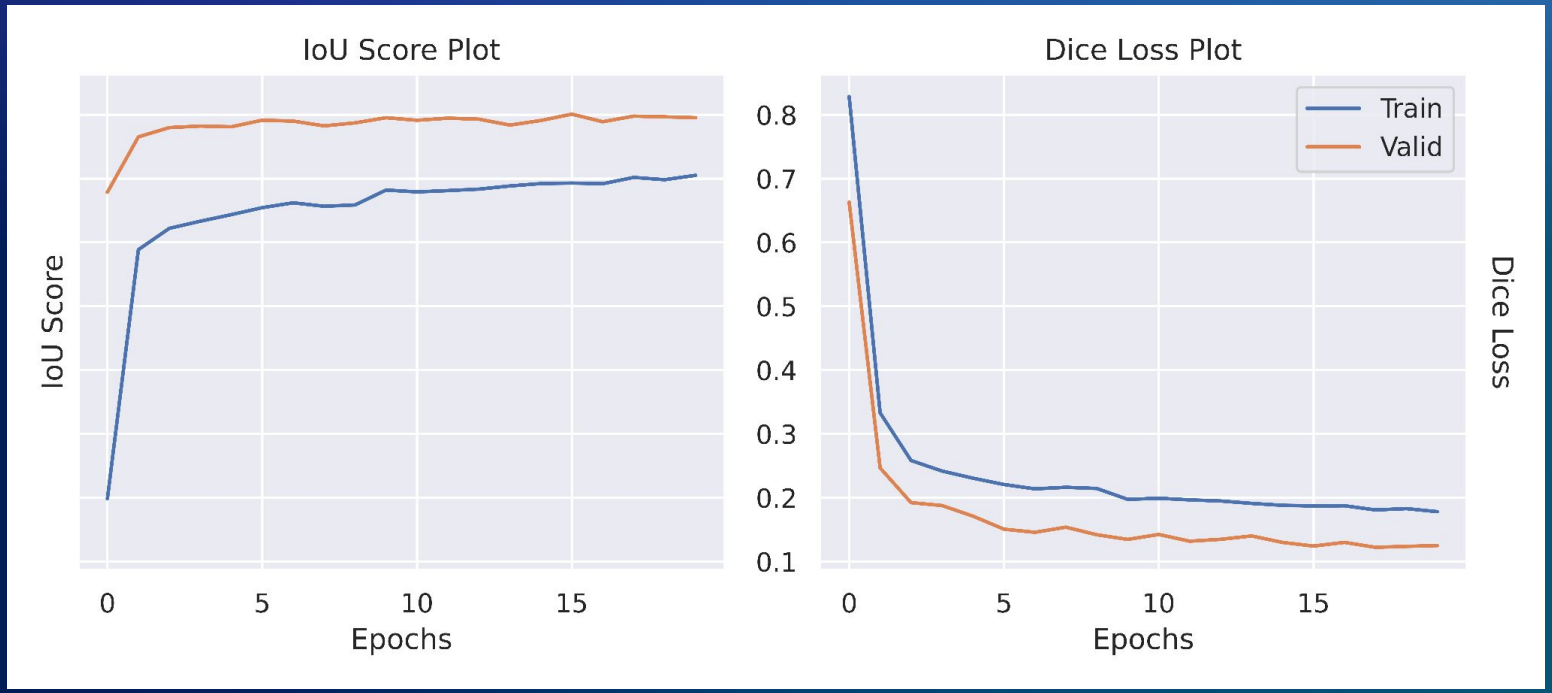
- **Dice Coefficient*:**
  A differentiable surrogate of IoU which is give by

  $$2\,|A \cap B| \,/\, (|A| + |B|)$$



* arxiv.org/pdf/1707.03237

CCP Segmentation; Training & Evaluation



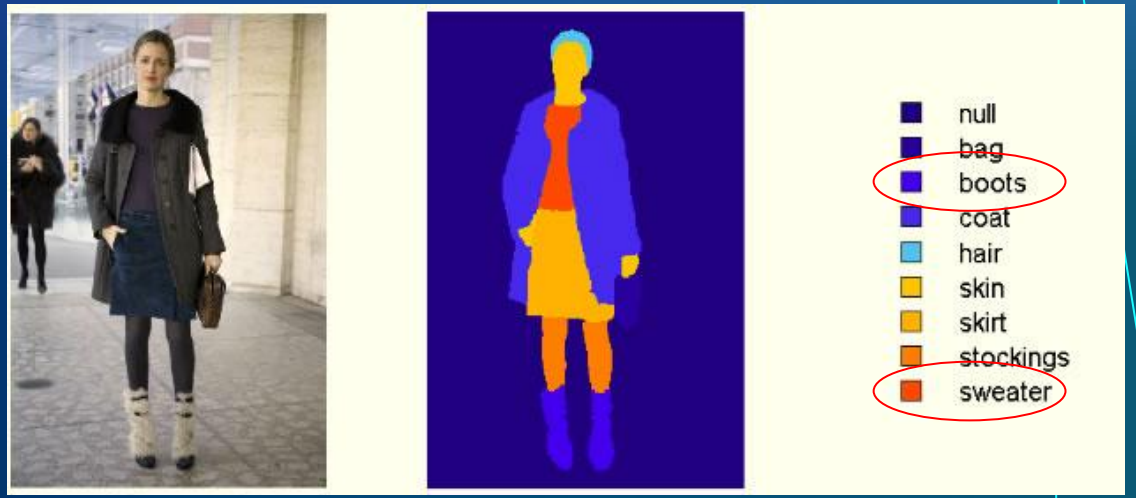
CCP Segmentation; Results

IoU	Dice Loss
0.8097	0.1189

CCP Co-Occurance

Observation:

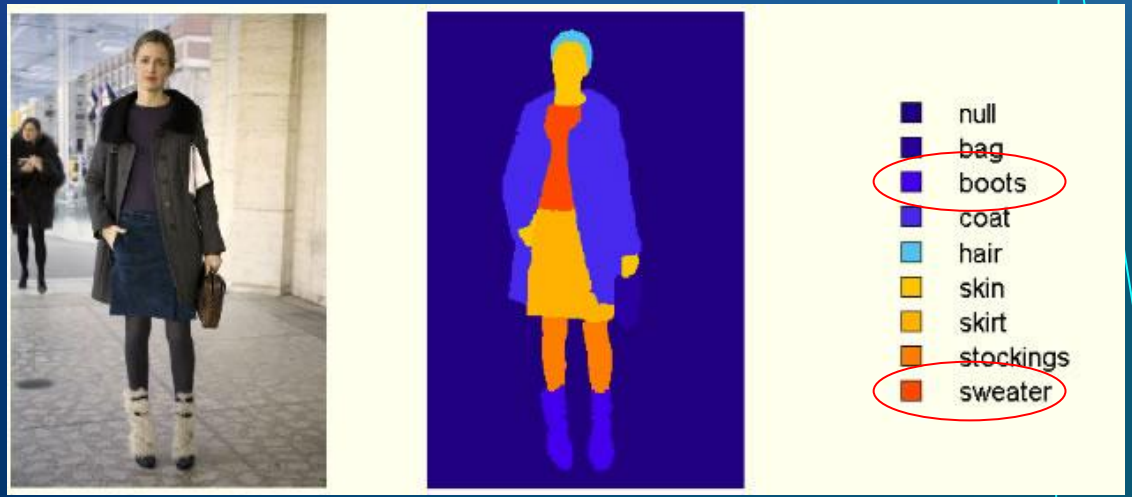
- Complementary items often ‘co-occur’ in aesthetic outfits.



CCP Co-Occurance

Observation:

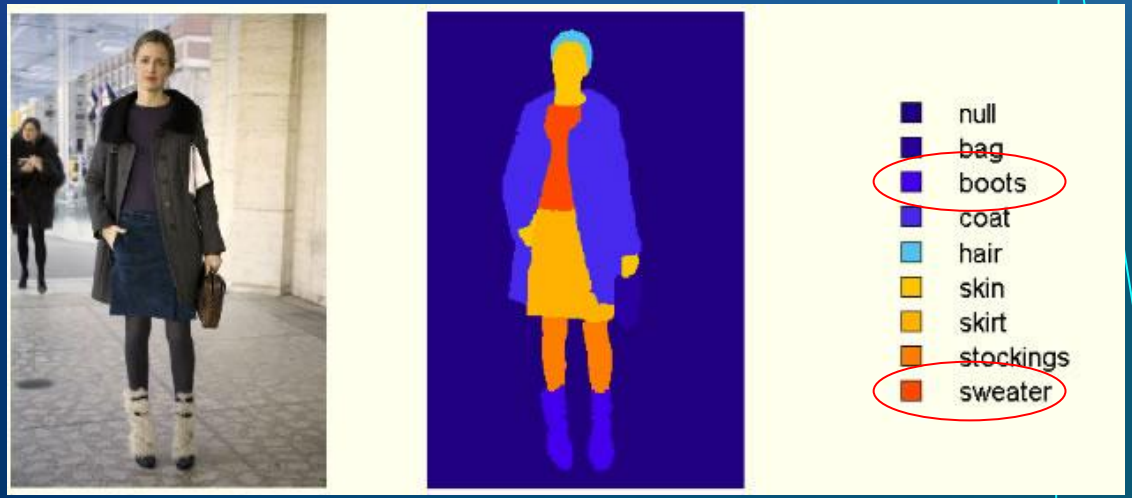
- Complementary items often ‘co-occur’ in aesthetic outfits.
- Items can be represented as set of features, e.g.:
  - Type (shoe vs. pants)
  - Color
  - Texture/Material
  - etc.



CCP Co-Occurance

Observation:

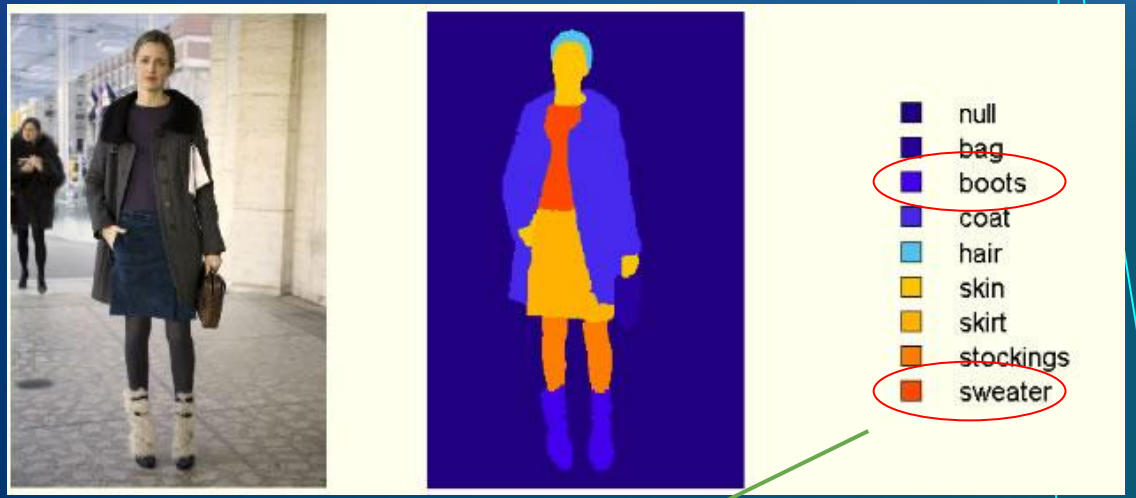
- Complementary items often ‘co-occur’ in aesthetic outfits.
- Items can be represented as set of features, e.g.:
  - Type (shoe vs. pants)
    => Segment!
  - Color
    => Average on Seg. mask
  - Texture/Material
  - etc.



CCP Co-Occurance

Observation:

- Complementary items often ‘co-occur’ in aesthetic outfits.
- Items can be represented as set of features, e.g.:
  - Type (shoe vs. pants)
    => Segment!
  - Color
    => Average on Seg. mask
  - Texture/Material
  - etc.



“brown sweater”

CCP Co-Occurance; Association Matrix

Build Association Matrix:

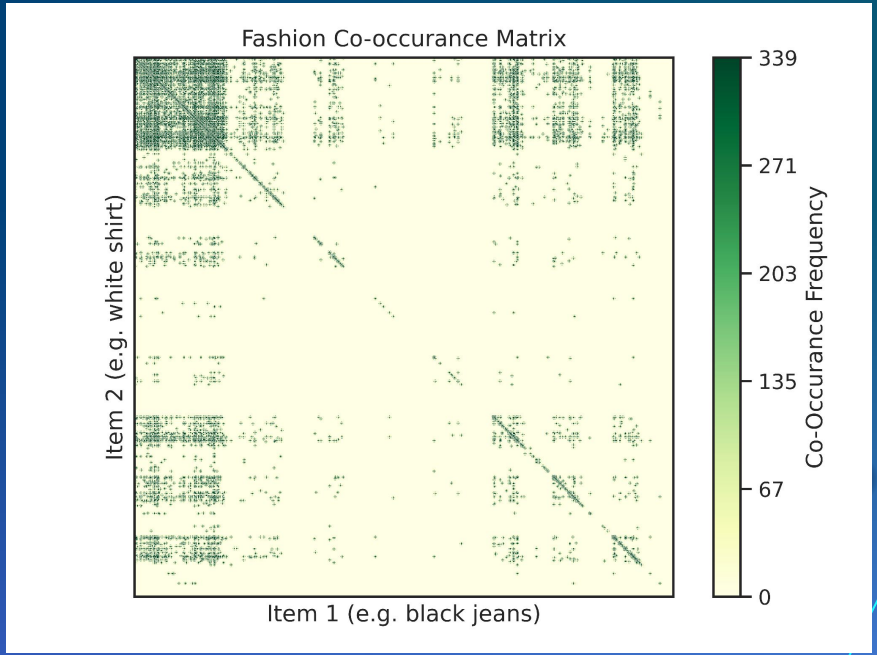
- Take 90% data (train+test).
- Count co-occuring items.
- Populate matrix **M**.

Usage – Query x:

- Return top N items of row **M[x]**.

Observation/Limitation:

- Matrix is sparse!



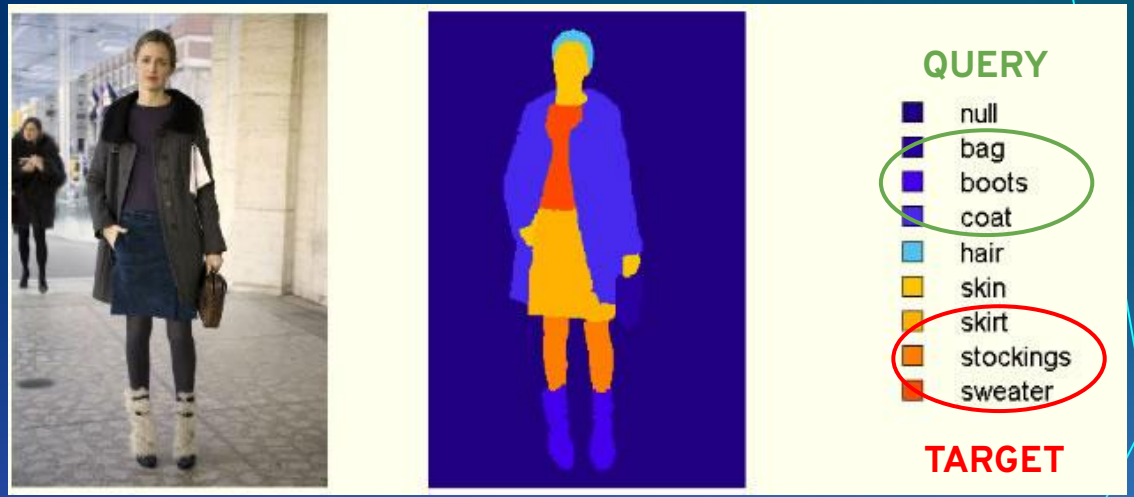
CCP Co-Occurance; Evaluation

For each 10% CCP Test Images:

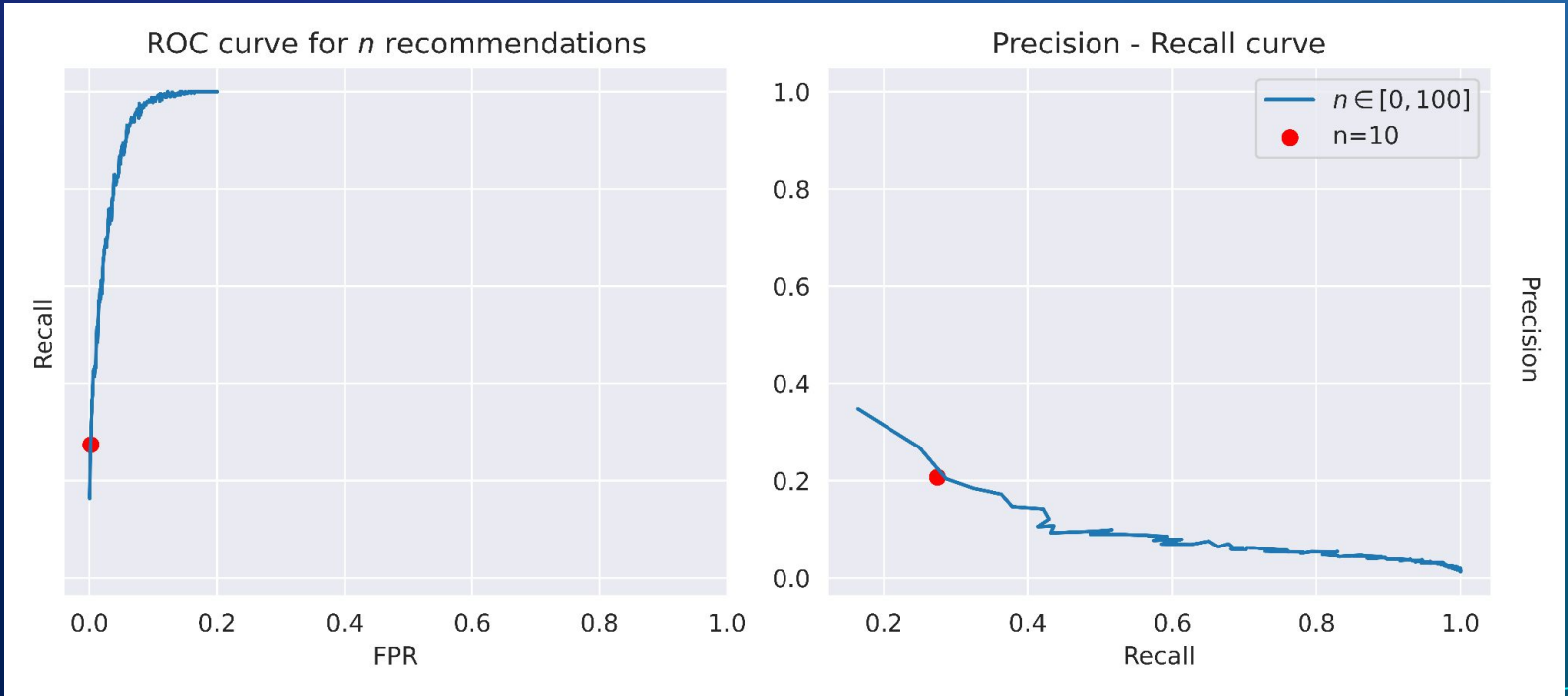
- Randomly split items into:
  - Query **x**
  - Target **t**
- Query matrix **M** with **x**.
- Compare result with **t.**

Evaluation:

- Average on multiple runs.
- Results vs. response size **n**.



CCP Co-Occurance; Evaluation



Fashion-Based Recommendations; Pipeline

Segment

Extract Item Features

Query Association Matrix

RESPONSE

CCP Co-Occurance; Target Bias

Problem:

- Given a target ‘fashion icon’ (i.e. celebrity), how want to bias our recommendations towards their style.

Solution:

- Scrape social media for target images.
- Use images to bias relevant items by scaling their counts in **M**.

Social Media Scraper

- Using Beautiful Soup and Selenium, we are web scraping social media sites such as Twitter, Instagram, Pinterest, and Facebook to get images of celebrities in their outfits
  - These images will be used to help generate recommendations
- Selenium is essential here because these websites offer image and other content dynamically which means we must utilize a web driver in order to render the page for web scraping
- This is a completely optional step as the user can skip entering a celebrity name and use the regular recommendation algorithm

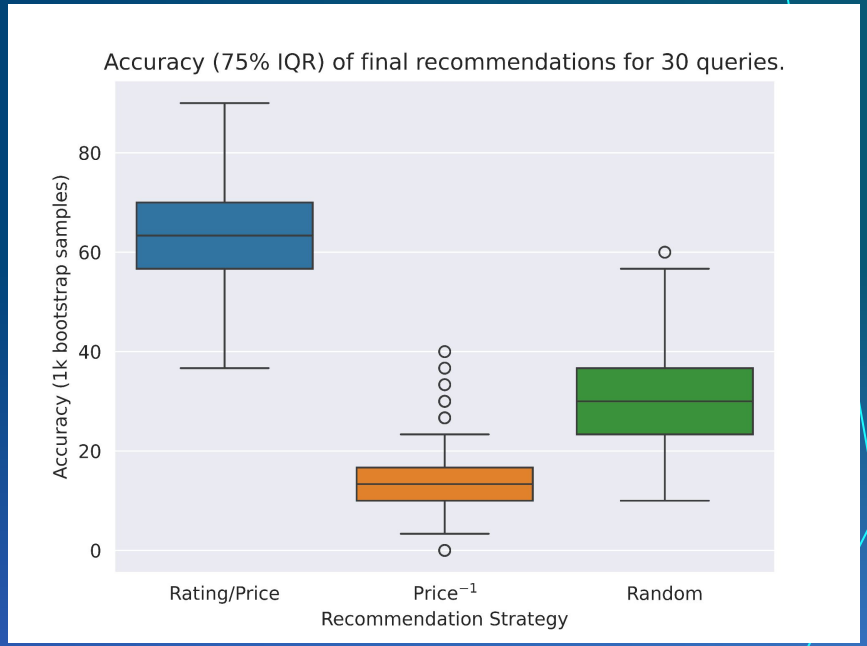
Amazon Products API

- Instead of using Amazon's own API, which features a limited amount of information, we are using Rainforest.
- Rainforest is a database of Amazon products that has been web scraped and can be accessed with an API endpoint
- Using this API, we can enter a search term and return the best product based on its value
    - Value is defined as Average Ratings/Price with Review Count being factored

End-End Recommendation Evaluation

Measuring Recommendation Accuracy:

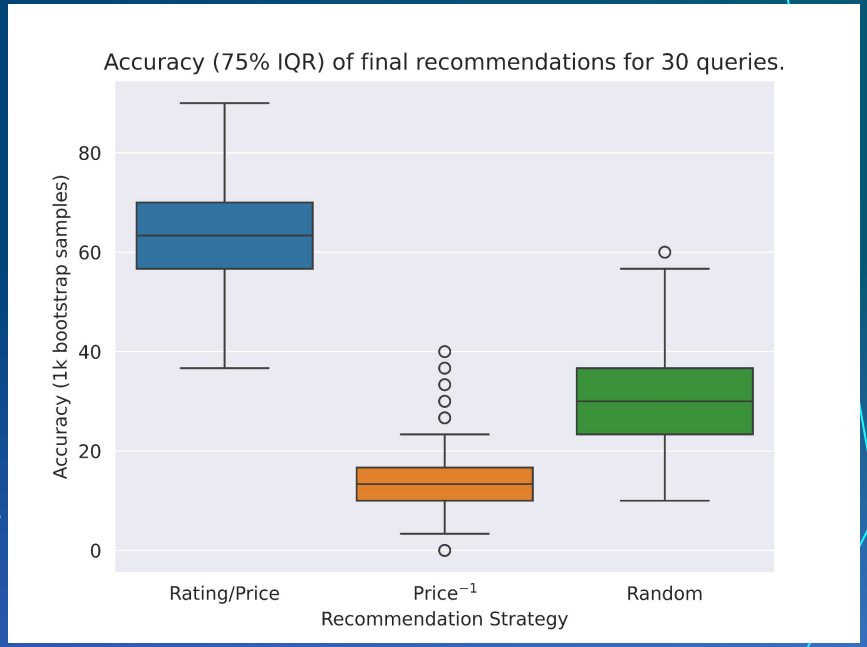
- We considered three metrics to retrieve final recommendations from Amazon:
  - Price^{-1}: Pick the **cheapest** item that matches with our API call query.
  - Rating/Price: Pick item with best rating/price that matches our query.
  - Random: Make a **random query** to API, pick item with best rating/price ratio.
- We did 30 API calls each (**6 fashion queries**).
- Team members were allowed to mark final recommendations as good/bad via app.



End-End Recommendation Evaluation

Recommendation Accuracy Results:

- Rating/Price performed best, and is set as our default metric.
- Price^{-1} performed worst because the API call often retrieved irrelevant cheap items, e.g. paper-clips.
- Random metric represents a random ‘fashion’ rating/price search without the use of our ML pipeline (segmentation+coOccurance matrix).
    - It underperformed our method, showing the effectiveness of our approach!



OPTIMIZATION 04

Mitigating API Costs; Caching

Problem:

- API casts can be large (both in terms of latency and monetarily).
- Avoid calling (Rainforest) API if item has already been queried.

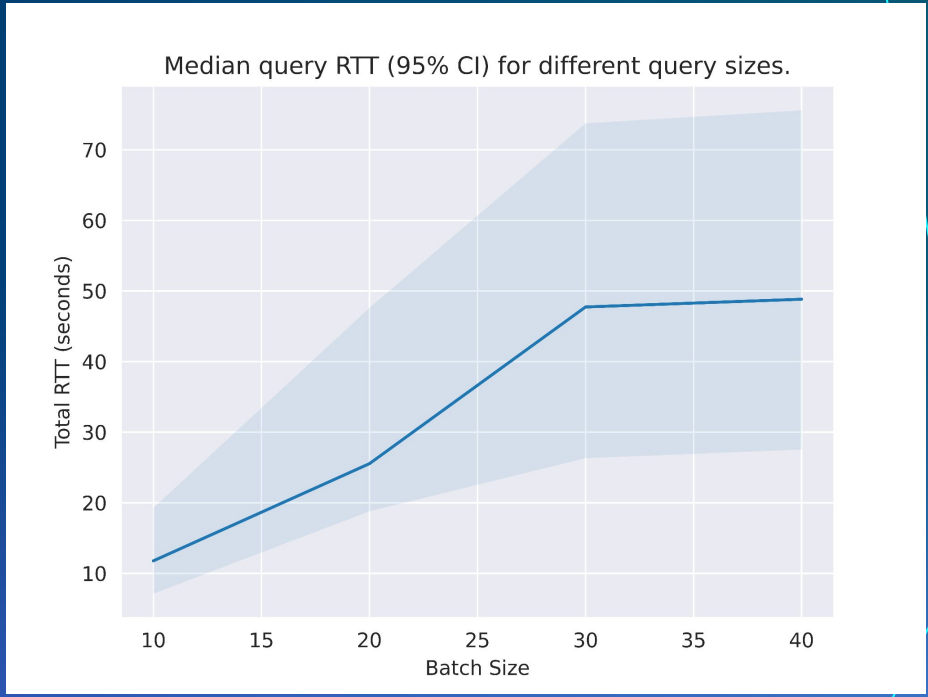
Mitigating API Costs; Caching

Measuring Latency:

- We measured query Round Trip Time (RTT) 10 times for different batch sizes.

Observations:

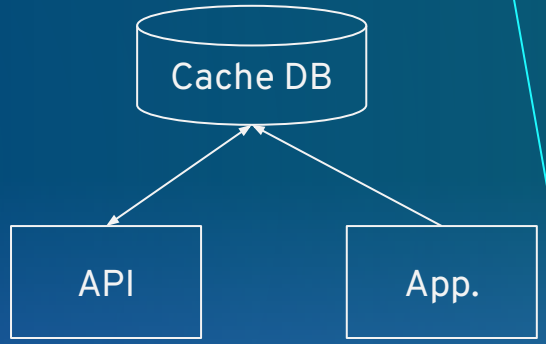
- RTT is large, even for small batches.
  - Use cache for lower latency!
- We noticed a diminishing increase in RTT as we batch more queries together.
  - Larger batches might give higher throughput (future work!).



Mitigating API Costs; Caching

Problem:

- API costs can be large.
- Avoid calling (Rainforest) API if item has already been queried.



Solution:

- Cache API results.
  - API will only be called if cache is missed.
  - With large application usage, cache hits will be large and save casts.

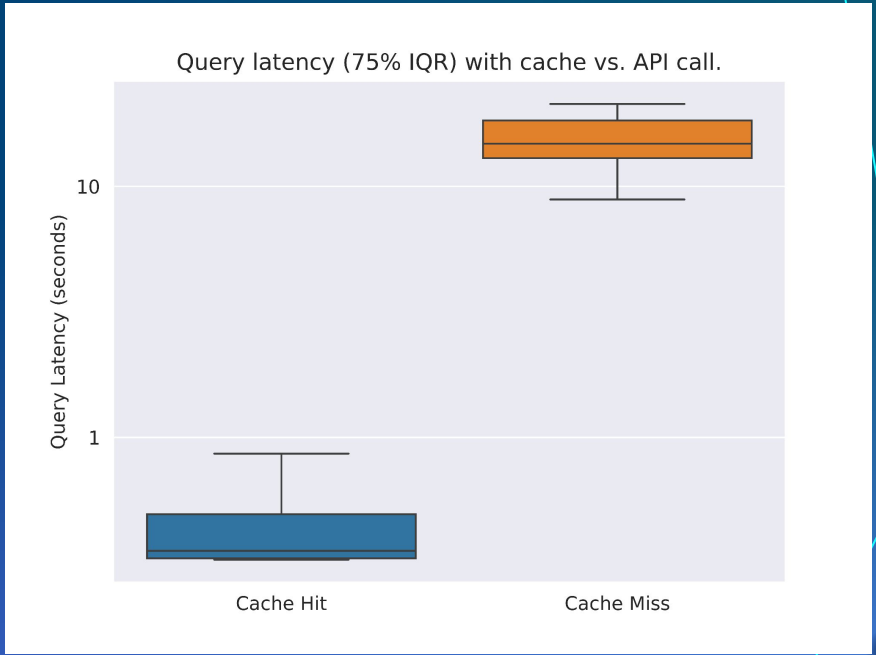
Mitigating API Costs; Caching

Results:

- We measured latency for ~10 queries With and without the cache.
- Results (right, log-scale) show more than order of magnitude improvement.

Assumptions:

- For this measurement, we assumed 100% cache hit rate.
- Assuming a more reasonable ~50% hit rate, we still get a significant (~2x) improvement in latency.



CCP Co-Occurance; Sparsity

Problem:

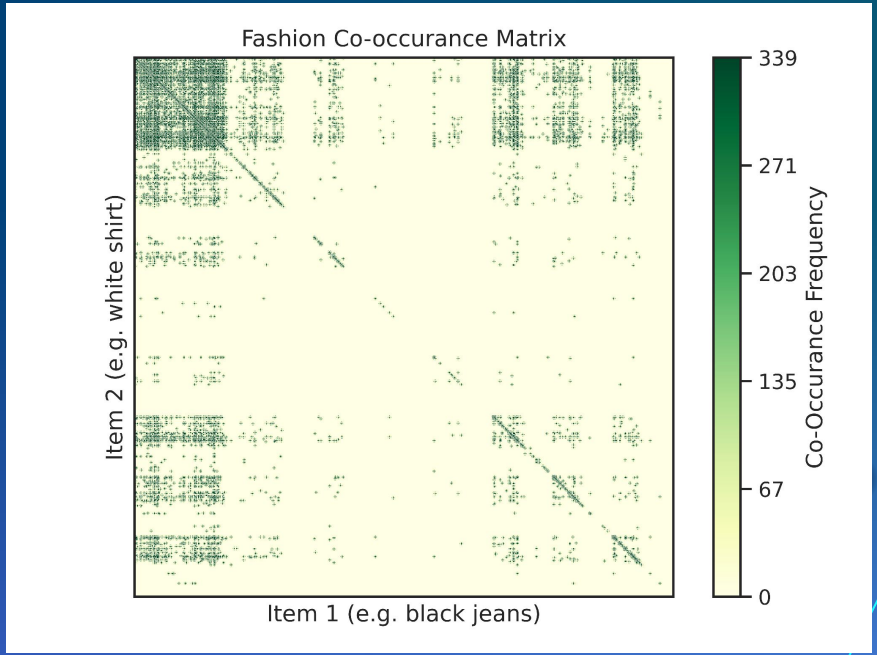
- Association matrix is sparse.
- Most entries cannot even be queried.

Solution:

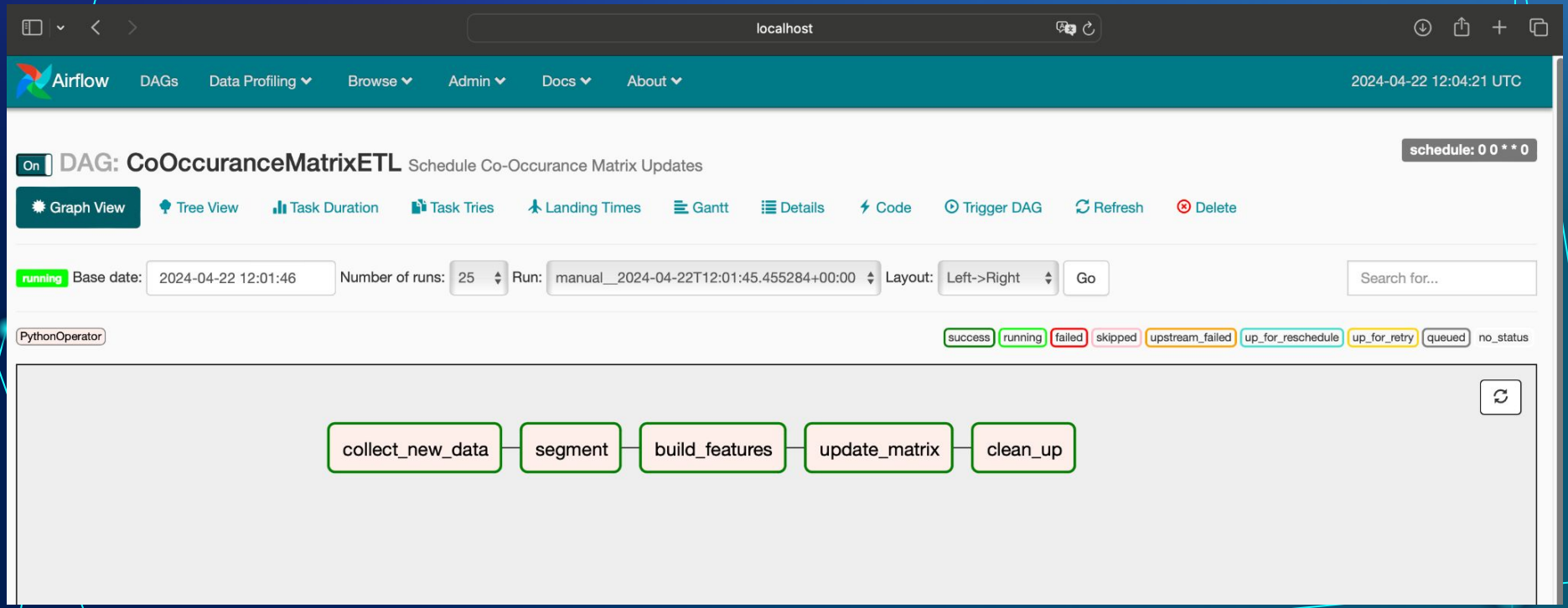
- Use scraped images as additional data.
  - Segment.
  - Extract items with features.
  - Update matrix with co-occurances.

Limitations/Problems:

- Poor segmenter will cause garbage data.



CCP Co-Occurance; Sparsity



05

CONCLUSION

Demo

https://youtu.be/wFeF4KocW64

Future Work

- In terms of future work, given more time to continue development we'd primarily focus our efforts on improving the overall accuracy of data segmentation so that our model would be able to more accurately detect various pieces of clothing in each photo.
- Additionally, we'd broaden the overall scope of the types of recommendations we provide and the links to items given back to users.
- We would like to allow the user to take a photo straight from the application rather than only supporting uploads.

What We Learned

- How to scrape various web pages with dynamic content using BeautifulSoup + Selenium
- How to manage Login Information using AWS RDS
- How to piece together a full stack application from barebones UI mockups to an actual functional final product
- How to work cross functionally on a team with various skill sets and technical backgrounds

Concluding Remarks

- We solved a novel *fashion-based clothing item recommender* problem.
- Implemented our segmentation and recommendation pipeline.
- Evaluated the performance with metrics like IoU and precision/recall respectively.
- We scrapped social media images for our use case and queried APIs.
- And we integrated everything in the form of a React+Flask application.

THANK YOU!
QUESTIONS?