AMENDED Exhibit 504

PLAINTIFFS' OMNIBUS OPPOSITION TO DEFENDANTS' MOTIONS FOR SUMMARY JUDGMENT

General Recommendation Intro

The purpose of this doc aims to give a gentle introduction into how the recommendation system works in an easy to understand, conceptual manner. Please note this doc is written in a manner that is meant to be accompanied by a spoken presentation.

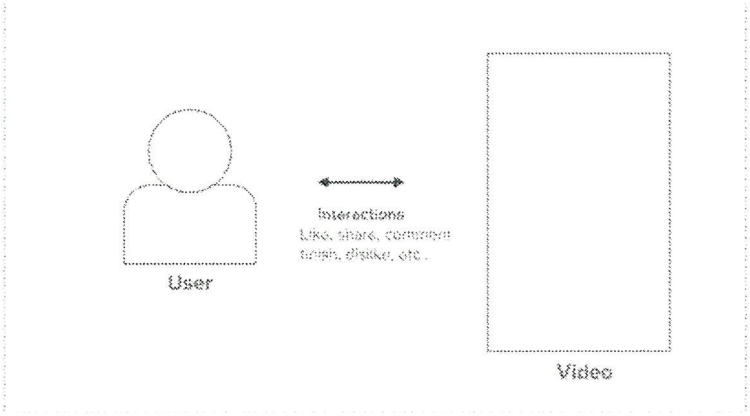
To watch the presentation, please see this link:

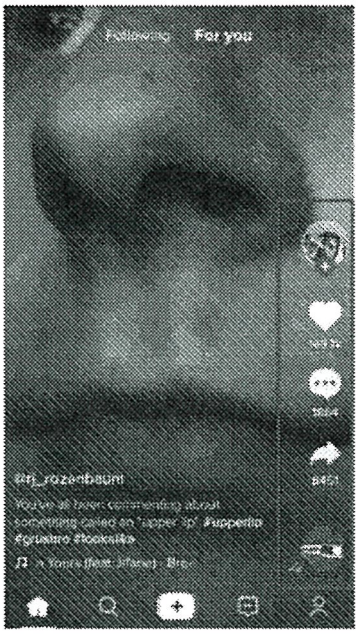
[redacted]

Simple Intro into how Recommendation Works

The goal of TikTok's recommender system is to find, for every user, a limited set of videos that have the highest chance to be enjoyed by the user.

TikTok, in essence, simply consists of the interactions between (1) users and (2) videos:





Likes, follows, comments, shares are some simple examples of the inputs that help the model learn the types of content the user likes

Based on the history of interactions between users and videos, we can learn to predict which videos a given user might like.

Examples of user feedback interactions on TikTok include:

- Like
- Comment
- Finish
- Share
- etc...

We can create a giant matrix to store **the interactions between users and videos**:

	VIDEO 1	VIDEO 2	VIDEO 3	VIDEO 4	VIDEO 5	...	VIDEO N
USER 1							
USER 2							
USER 3							
USER 4							
USER 5							
...							
USER N							

In the below example, we can annotate which users "liked" a video by marking with a "1":

Commented [1]: In reality, there are a wide range of interactions we use (like, follow, share, comment, finish, etc...)

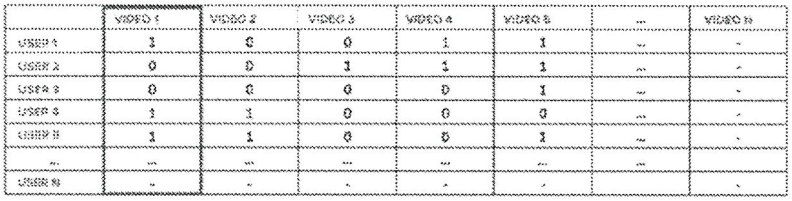
	VIDEO 1	VIDEO 2	VIDEO 3	VIDEO 4	VIDEO 5	...	VIDEO N
USER 1	1	0	0	1	1	...	.
USER 2	0	0	1	1	1	...	..
USER 3	0	0	0	0	1	...	.
USER 4	1	1	0	0	0	...	.
USER 5	1	1	0	0	1	...	.
...	...	...	...	...	...	...	
USER N	.	.	.	.	.		.

We can then create a vector for each user (and video) explaining their interactions with each video, with a "1" representing a "like" by the user, and a "0" representing a no like.

	VIDEO 1	VIDEO 2	VIDEO 3	VIDEO 4	VIDEO 5	...	VIDEO N
USER 1	1	0	0	1	1	...	.
USER 2	0	0	1	1	1	...	.
USER 3	0	0	0	0	1	...	.
USER 4	1	1	0	0	0	...	.
USER 5	1	1	0	0	1	...	.
...	...	...	...	...	...	...	
USER N	.	.	.	.	.		.

user 1 vector = [1 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 ....]

At the same time, we can also generate a video vector, based on the interactions of all users on the video, including which users "liked" the video with a "1", and which videos were not liked with a "0":



	VIDEO 1	VIDEO 2	VIDEO 3	VIDEO 4	VIDEO 5	...	VIDEO N
USER 1	1	0	0	1	1	...	...
USER 2	0	0	1	1	1	...	...
USER 3	0	0	0	0	1	...	...
USER 4	1	1	0	0	0	...	...
USER 5	1	1	0	0	1	...	...
...	...	...	...	...	...	...	
USER N	...	...	...	...	...		...

video 1 vector = [1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 ....]

If you can imagine that there are hundreds of millions of users and billions of videos, the user and video vectors mentioned above are called "sparse" vectors because they contain many, many 0's. We need a way to transform these sparse vectors into "dense" vectors, which are smaller, more compact and efficient representations to be used in our our recommendation system.

Neural networks can help us learn the hidden representations between users and videos, in order to predict which videos a user might like. Based on the historical user feedback interactions between users and videos outlined above, we input these user and video sparse vectors [0,1,1,0,0,1] into the neural network, and output a recommendation vector, representing at a high level, the user's interests.

There are a variety of techniques to map users and videos into lower dimensional vectors. In such systems, a higher inner product between a user vector and a video vector indicates that the video better suits the user's preference. Traditionally, retrieving the most suitable videos is done by scoring and sorting all videos. Videos with the highest scores would then be recommended to the user.

In reality, we employ a "multi-objective function" that explains a variety of objectives we optimize for when predicting whether a user will find interest in a video, including:

- Like
- Follow
- Share
- Comment
- Finish
- Playtime
- etc...

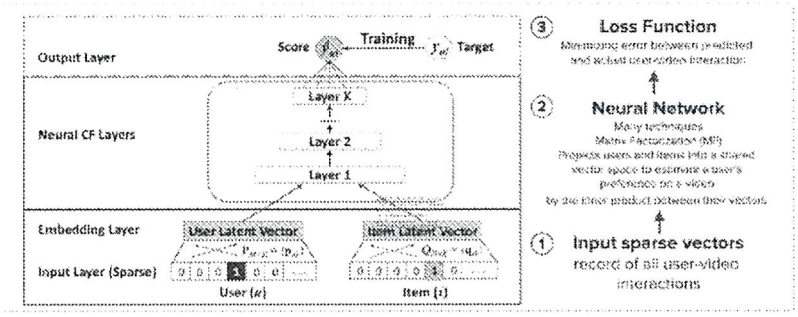
We can apply different weights to each action mentioned above, to create a weighted, multi-objective formula to explain the predicted interaction between users and videos:

Weight 1 * predict(like) + weight 2 * predict(follow) + weight 3 * predict(share)...

- Weights are numerical values that increase or decrease the importance of each specific action
- Called the Value Tree, the optimization and assigning of weights to specific action plays a big part in improving the overall performance of the recommendation system.

Simple example of neural collaborative filtering:

> The objective is to minimize the loss function describing the predicted like action vs the actual like action between a user and a given video (did the user like the video or not):



YouTube's recommendation system architecture in 2016: [HYPERLINK "https://static.googleusercontent.com/media/research.google.com/zh-CN//pubs/archive/45530.pdf" \h]

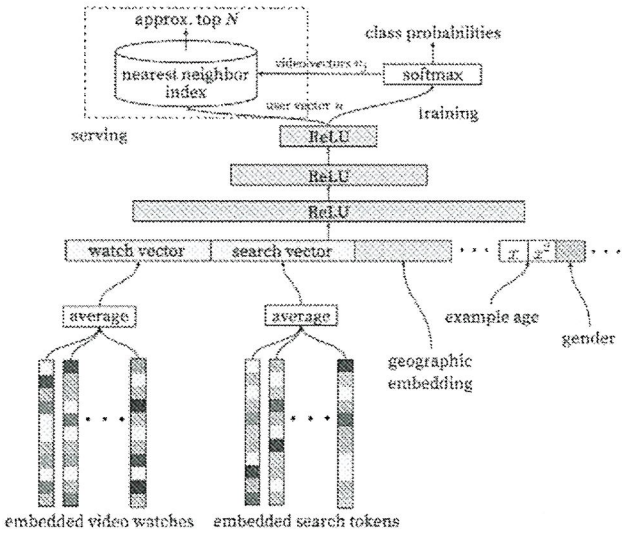


Figure 3: Deep candidate generation model architecture showing embedded sparse features concatenated with dense features. Embeddings are averaged before concatenation to transform variable sized bags of sparse IDs into fixed-width vectors suitable for input to the hidden layers. All hidden layers are fully connected. In training, a cross-entropy loss is minimized with gradient descent on the output of the sampled softmax. At serving, an approximate nearest neighbor lookup is performed to generate hundreds of candidate video recommendations.

YouTube's recommendation in 2019: [HYPERLINK "https://daiwk.github.io/assets/youtube-multitask.pdf" \h]

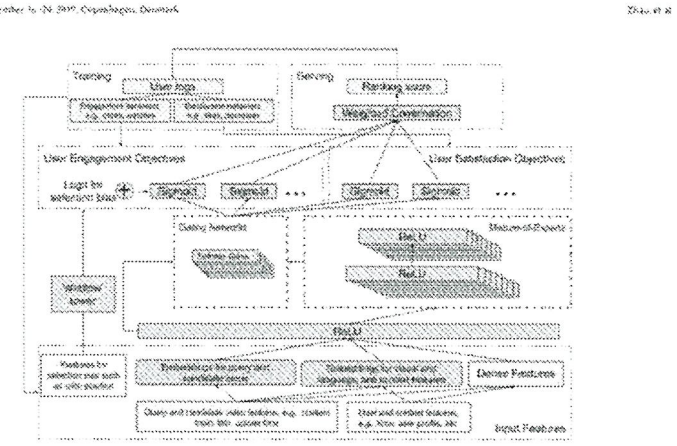


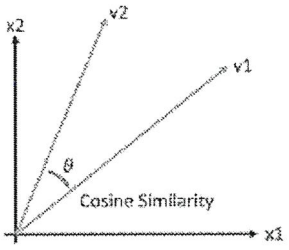
Figure 3: Model architecture of our proposed ranking system. It consumes user logs as training data. Builds Multi-gate Mixture-of-Experts layers to predict two categories of user behaviors, i.e., engagement and satisfaction. It corrects ranking selection bias with a side-tower. On top, multiple predictions are combined into a final ranking score.

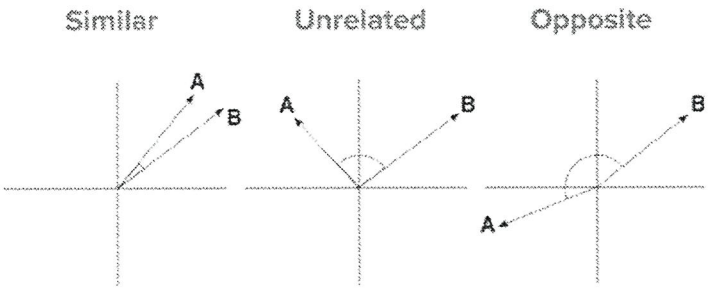
Learn more at Google ML Developer: [HYPERLINK "https://developers.google.com/machine-learning/crash-course/embeddings/motivation-from-collaborative-filtering" \h]

What the final user embedding output looks like:

With this vector we can do two things:

1. User similarity, and nearest neighbor search (look up most similar users or videos)
2. Compare the distance between two vectors.
   a. Recommend videos to user: Input user vector, find video vector with closest distance
   b. Find similar users: Input user vector, find nearest user vectors with closest distance
   c. Find similar videos: Input video vector, find nearest video vectors with closest distance





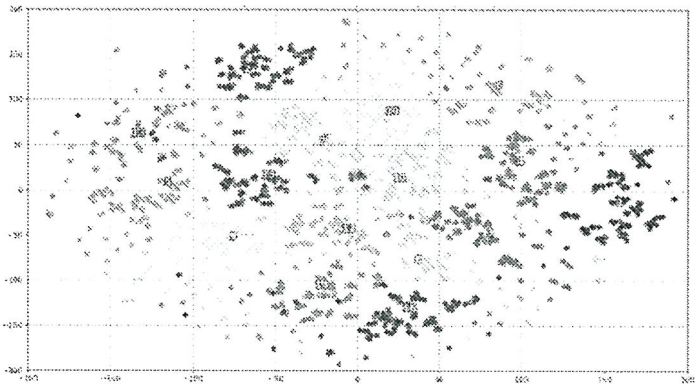
Because each user has a recommendation vector, we can easily measure the cosine similarity to find the similarity between two users.

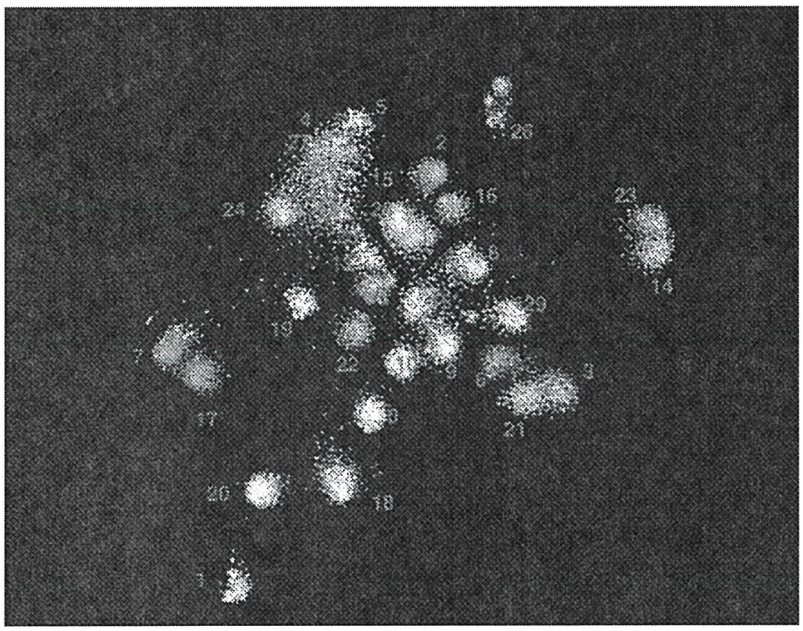
cosim = 1 means perfect similarity and cosim = 0 no similarity.

We can use cosine similarity to find similar videos and users that share similar interests, as well as the similarity between a user and a video.

Clustering

Unsupervised clustering: generate groups of similar users or videos





Recommendation briefly over the past 10 years:

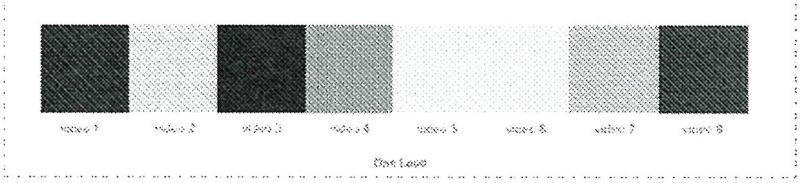
Early works on recommendation systems typically use Collaborative Filtering (CF) to model users' preferences based on their interaction histories [26, 43]. Among various CF methods, Matrix Factorization (MF) is the most popular one, which projects users and items into a shared vector space and estimate a user's preference on an item by the inner product between their vectors [26, 27, 41]. Another line of work is item-based neighborhood methods [20, 25, 31, 43]. They estimate a user's preference on an item via measuring its similarities with the items in her/his interaction history using a precomputed item-to-item similarity matrix. Recently, deep learning has been revolutionizing the recommendation systems dramatically. The early pioneer work is a two-layer Restricted Boltzmann Machines (RBM) for collaborative filtering, proposed by Salakhutdinov et al. [42] in Netflix Prize1 . https://www.netflixprize.com

One line of deep learning based methods seeks to improve the recommendation performance by integrating the distributed item representations learned from auxiliary information, e.g., text [23, 53], images [21, 55], and acoustic features [51] into CF models. Another line of work seeks to take the place of conventional matrix factorization. For example, Neural Collaborative Filtering (NCF) [12]

estimates user preferences via Multi-Layer Perceptions (MLP) instead of inner product, while AutoRec [44] and CDAE [57] predict users' ratings using Auto-encoder framework. [HYPERLINK "https://arxiv.org/pdf/1904.06690.pdf" \h]

Loads

When a new user downloads and installs TikTok, when they first open the app, a request is sent to the recommendation system for a "first load of videos". This load contains 8 videos and these 8 videos are chosen from a small pool of videos that are known to be very popular on TikTok and that have also passed safety moderation queues so we have verified they are appropriate for a general audience:



As the user watches these videos, the algorithm records all of the user's interactions with the videos, for example, did the user do any of the following:

- Like, share, comment, follow, finish, skip, etc...

Once the user finishes this first load of 8 videos, a new load of 8 videos is requested from the recommendation system. The same process ensues throughout the lifetime of the user and their "feed" can be said to be made up of many, many loads. For example, a user who watched 1,000 videos, has requested 125 loads from the recommendation system:

As a user finds an interest(s), the recommendation system begins to recommend more and more "similar" videos, and their feed can be said to have a specific "feel" to it:

To make sure that a user's feed does not become too "similar" and that there remains an element of "diversity" within the user's feed, we calculate a "load score", translating to the overall similarity of the 8 videos in the load. The similarity is calculated by comparing the recommendation score for each video, and how close these scores are to each other, returning a number between 0-1 (0 representing zero similarity and 1 representing perfect similarity, or 100% similarity meaning it's the same video). We will call a load highly similar if its score is >= 60%.

Example of Rules

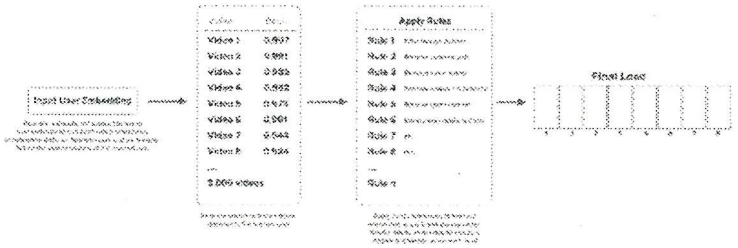
The rules of the system have a major impact on the results of the recommendation. Below is a simple example of some rules that would impact the final user load of 8 videos. Each region has its own rules/strategies which update as content / user structure changes over time. Please check with your local algo teammate for the latest rules:

1. **Content localization**: ensures that only videos published by an author of the same country as the user will appear in the user's recommendation feed. For example, users in the US will, for the large part, only see videos published by authors from the US. Generally speaking, for most countries, for every 8 videos recommended, at least 4 (8:4) of those videos are from authors of the same country as the user.
2. **Same Sounds**: for every 8 videos recommended, the same sound will never appear more than once (8:1).
3. **Same Author:** for every 20 videos, the same author will never appear more than once (20:1).
4. **Short Videos:** videos less than 7 seconds are not recommended.
5. **Vulgar / Inappropriate**: a spam detection model identifies low quality videos or videos with vulgar / inappropriate content, removing them from the recommendation pool.
6. **Newly Published Video Learning:** videos that are newly published have not run through the recommendation system enough times to get adequately scored and recommended to their corresponding users. Nevertheless, these newly published videos need to appear in enough feeds to get adequately scored and added to the recommendation pool if they score high enough.
7. **Not Recommended**: videos that have been NR'd will not be recommended
8. **etc...**

Simplified Basic Recommendation Steps

1. Input user embedding and calculate cosine similarity with all items in the recommendation pool
2. Rank order items with highest similarity
3. Apply filter rules to legalize the load of 8 videos; shuffle the videos to ensure the similarity of the videos within the load is not too high

4. Rank order top 8 remaining videos to get the final load
5. Repeat steps 1-4 as the user completes each load to generate the FYP



[HYPERLINK

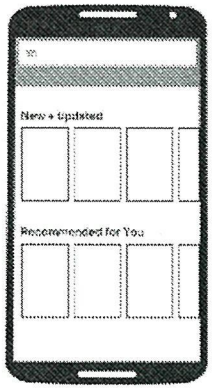
[redacted]

Appendix

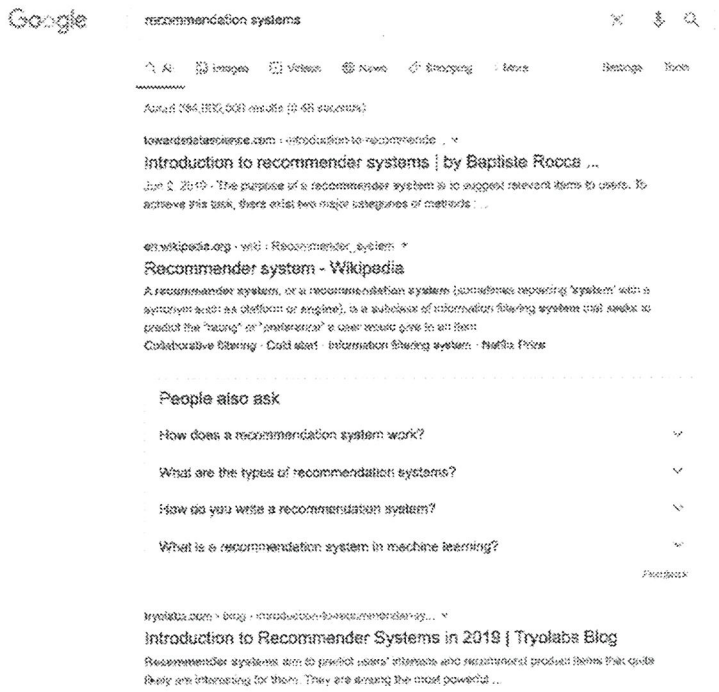
Examples of "loads" in other recommendation products:

Google Play App Store:

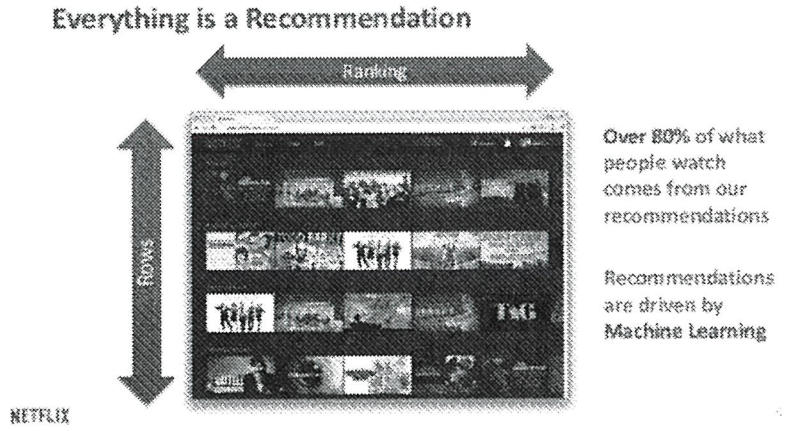
If you go to the Google Play Apps homepage, you may see something like this:



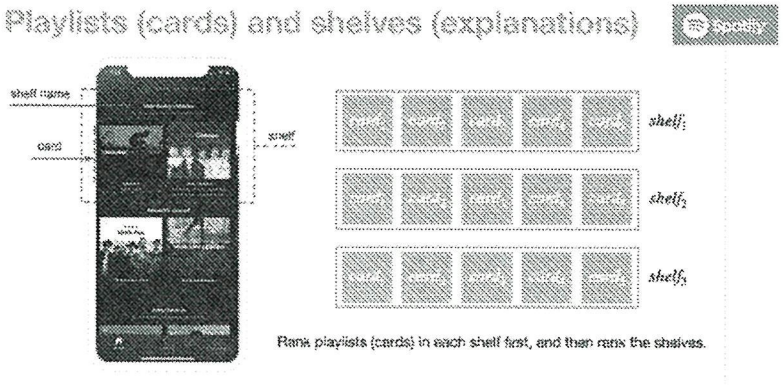
Google Search:



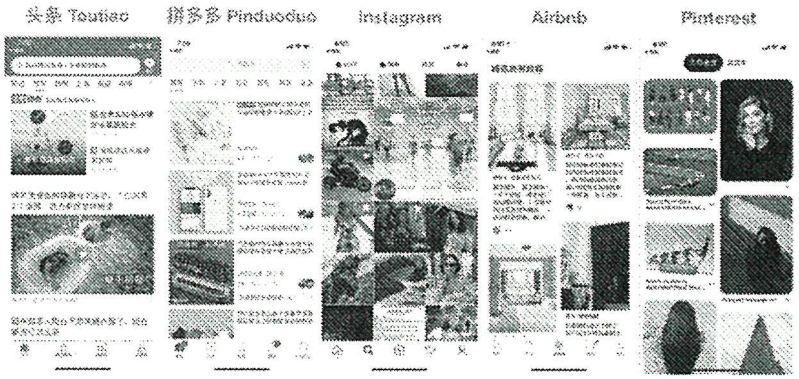
Netflix



Spotify Playlists



Other Examples:



[HYPERLINK [redacted]
[HYPERLINK
[HYPERLINK
[HYPERLINK

全文点赞列表

DOCUMENT METADATA

Bates Number: TIKTOK3047MDL-045-LARK-00468321-TIKTOK3047MDL-045-LARK-00468335

BEGATTACH: TIKTOK3047MDL-045-LARK-00468247

ENDATTACH: TIKTOK3047MDL-068-LARK-01047298

Attachment Count: 0

PRODVOL: MDL-045

Custodian: MAHER, REAGAN; [redacted]

File Path: /TIKTOK3047MDL-045-LARK-00468321.PDF

Confidentiality Designation:

HASHVALUE: DOCCNZ2LBZQRS8CPMVTCNLLMGZH.DOCX

DocType:

Author:

Create Date: 2/10/2021 12:00 AM

Last Modified Date: 2/10/2021 12:00 AM

LAST MODIFIED BY:

TRACK CHANGES:

COMMENTS:

HASHIDDENDATA:

Filename: DOCCNZ2LBZQRS8CPMVTCNLLMGZH.DOCX

Title:

DOCEXT: DOCX

Email From:

Email To:

Email CC:

Email BCC:

Received Date:

Sent Date:

TIMEZONE:

Subject:

THREADID:

REDACTIONS:

REDACTION TYPE: