-- This SQL query generate the cluster to top k tweets index based on tweet engagements.

-- The engagement type is decided by USER\_TWEET\_ENGAGEMENT\_TABLE\_SQL.

with vars as (

SELECT {HALF\_LIFE} AS halfLife, -- Default: 8 hour halfLife in millis

UNIX\_MILLIS("{CURRENT\_TS}") AS currentTs,

),

user\_tweet\_engagement\_pairs AS (

{USER\_TWEET\_ENGAGEMENT\_TABLE\_SQL}

),

-- A sequence of filters to get eligible tweetIds for tweet embedding generation

-- Apply min interaction count filter

user\_tweet\_interaction\_with\_min\_interaction\_count\_filter AS (

SELECT userId, user\_tweet\_engagement\_pairs.tweetId, tsMillis

FROM user\_tweet\_engagement\_pairs, vars

JOIN (

SELECT tweetId, COUNT(DISTINCT(userId)) AS interactionCount

FROM user\_tweet\_engagement\_pairs

GROUP BY tweetId

HAVING interactionCount >= {MIN\_INTERACTION\_COUNT} -- Only generate tweet embeddings for tweets with >= {MIN\_INTERACTION\_COUNT} interactions

) eligible\_tweets USING(tweetId)

),

-- Apply min fav count filter

user\_tweet\_interaction\_with\_fav\_count\_filter AS (

{TWEET\_INTERACTION\_WITH\_FAV\_COUNT\_FILTER\_SQL}

),

-- Apply health and video filter

user\_tweet\_interaction\_with\_health\_filter AS (

{TWEET\_INTERACTION\_WITH\_HEALTH\_FILTER\_SQL}

),

-- Final filtered user tweet interaction table

-- Read the result from the last filter

user\_tweet\_interaction\_processed\_table AS (

SELECT \*

FROM user\_tweet\_interaction\_with\_health\_filter

),

-- Read consumer embeddings

consumer\_embeddings AS (

{CONSUMER\_EMBEDDINGS\_SQL}

),

-- Update tweet cluster scores based on interaction events

tweet\_cluster\_scores AS (

SELECT tweetId,

STRUCT(

clusterId,

CASE vars.halfLife

-- halfLife = -1 means there is no half life decay and we directly take the sum as the score

WHEN -1 THEN SUM(clusterNormalizedLogFavScore)

ELSE SUM(clusterNormalizedLogFavScore \* POW(0.5, (currentTs - tsMillis) / vars.halfLife))

END AS normalizedScore,

COUNT(\*) AS engagementCount)

AS clusterIdToScores

FROM user\_tweet\_interaction\_processed\_table, vars

JOIN consumer\_embeddings USING(userId)

GROUP BY tweetId, clusterId, vars.halfLife

),

-- Generate tweet embeddings

tweet\_embeddings\_with\_top\_clusters AS (

SELECT tweetId, ARRAY\_AGG(

clusterIdToScores

ORDER BY clusterIdToScores.normalizedScore DESC

LIMIT {TWEET\_EMBEDDING\_LENGTH}

) AS clusterIdToScores

FROM tweet\_cluster\_scores

GROUP BY tweetId

),

clusters\_top\_k\_tweets AS (

SELECT clusterId, ARRAY\_AGG(STRUCT(tweetId, normalizedScore AS tweetScore) ORDER BY normalizedScore DESC LIMIT {CLUSTER\_TOP\_K\_TWEETS}) AS topKTweetsForClusterKey

FROM tweet\_embeddings\_with\_top\_clusters, UNNEST(clusterIdToScores) AS clusterIdToScores

WHERE engagementCount >= {MIN\_ENGAGEMENT\_PER\_CLUSTER}

GROUP BY clusterId

)

SELECT \*

FROM clusters\_top\_k\_tweets