namespace java com.twitter.ann.common.thriftjava

#@namespace scala com.twitter.ann.common.thriftscala

#@namespace strato com.twitter.ann.common

namespace py gen.twitter.ann.common

include "com/twitter/mediaservices/commons/ServerCommon.thrift"

include "com/twitter/ml/api/embedding.thrift"

/\*\*

\* Thrift schema for storing file based Index mapping

\*/

struct FileBasedIndexIdStore {

1: optional map<i64, binary> indexIdMap

}

enum DistanceMetric {

L2, Cosine, InnerProduct,

RESERVED\_4, RESERVED\_5, RESERVED\_6, RESERVED\_7, EditDistance

} (persisted = 'true', strato.graphql.typename='DistanceMetric')

struct AnnoyIndexMetadata {

1: i32 dimension

2: DistanceMetric distanceMetric

3: i32 numOfTrees

4: i64 numOfVectorsIndexed

} (persisted = 'true', strato.graphql.typename='AnnoyIndexMetadata')

struct AnnoyRuntimeParam {

/\* Number of vectors to evaluate while searching. A larger value will give more accurate results, but will take longer time to return.

\* Default value would be numberOfTrees\*numberOfNeigboursRequested

\*/

1: optional i32 numOfNodesToExplore

}

struct HnswRuntimeParam {

// More the value of ef better the recall with but at cost of latency.

// Set it greater than equal to number of neighbours required.

1: i32 ef

}

// These options are subset of all possible parameters, defined by

// https://github.com/facebookresearch/faiss/blob/36f2998a6469280cef3b0afcde2036935a29aa1f/faiss/AutoTune.cpp#L444

// quantizer\_ prefix changes IndexIVF.quantizer parameters instead

struct FaissRuntimeParam {

// How many cells to visit in IVFPQ. Higher is slower / more precise.

1: optional i32 nprobe

// Depth of search in HNSW. Higher is slower / more precise.

2: optional i32 quantizer\_ef

// How many times more neighbours are requested from underlying index by IndexRefine.

3: optional i32 quantizer\_kfactor\_rf

// Same as 1: but for quantizer

4: optional i32 quantizer\_nprobe

// Hamming distance threshold to filter neighbours when searching.

5: optional i32 ht

}

// Every ANN index will have this metadata and it'll be used by the query service for validation.

struct AnnIndexMetadata {

1: optional i64 timestamp

2: optional i32 index\_size

3: optional bool withGroups

4: optional i32 numGroups

} (persisted = 'true')

struct HnswIndexMetadata {

1: i32 dimension

2: DistanceMetric distanceMetric

3: i32 numElements

} (persisted = 'true')

struct HnswInternalIndexMetadata {

1: i32 maxLevel

2: optional binary entryPoint

3: i32 efConstruction

4: i32 maxM

5: i32 numElements

} (persisted = 'true')

struct HnswGraphEntry {

1: i32 level

2: binary key

3: list<binary> neighbours

} (persisted = 'true', strato.graphql.typename='HnswGraphEntry')

enum IndexType {

TWEET,

USER,

WORD,

LONG,

INT,

STRING,

RESERVED\_7, RESERVED\_8, RESERVED\_9, RESERVED\_10

} (persisted = 'true', strato.graphql.typename='IndexType')

struct CosineDistance {

1: required double distance

}

struct L2Distance {

1: required double distance

}

struct InnerProductDistance {

1: required double distance

}

struct EditDistance {

1: required i32 distance

}

union Distance {

1: CosineDistance cosineDistance

2: L2Distance l2Distance

3: InnerProductDistance innerProductDistance

4: EditDistance editDistance

}

struct NearestNeighbor {

1: required binary id

2: optional Distance distance

}

struct NearestNeighborResult {

// This list is ordered from nearest to furthest neighbor

1: required list<NearestNeighbor> nearestNeighbors

}

// Different runtime/tuning params while querying for indexes to control accuracy/latency etc..

union RuntimeParams {

1: AnnoyRuntimeParam annoyParam

2: HnswRuntimeParam hnswParam

3: FaissRuntimeParam faissParam

}

struct NearestNeighborQuery {

1: required embedding.Embedding embedding

2: required bool with\_distance

3: required RuntimeParams runtimeParams,

4: required i32 numberOfNeighbors,

// The purpose of the key here is to load the index in memory as a map of Option[key] to index

// If the key is not specified in the query, the map value corresponding to None key will be used

// as the queryable index to perform Nearest Neighbor search on

5: optional string key

}

enum BadRequestCode {

VECTOR\_DIMENSION\_MISMATCH,

RESERVED\_2,

RESERVED\_3,

RESERVED\_4,

RESERVED\_5,

RESERVED\_6,

RESERVED\_7,

RESERVED\_8,

RESERVED\_9

}

exception BadRequest {

1: string message

2: required BadRequestCode code

}

service AnnQueryService {

/\*\*

\* Get approximate nearest neighbor for a given vector

\*/

NearestNeighborResult query(1: NearestNeighborQuery query)

throws (1: ServerCommon.ServerError serverError, 2: BadRequest badRequest)

}