Beyond Federated Learning decentralised learning on the edge

Quinten van Eijs
Delft University of Technology
Delft, The Netherlands
J.A.Pouwelse@tudelft.nl

Johan Pouwelse
Delft University of Technology
Delft, The Netherlands
J.A.Pouwelse@tudelft.nl

***Abstract*—**

***Index Terms*—Decentralized learning, P2P, Information Retrieval.**

I. Introduction

In the current digital era, artificial intelligence (AI) has become a critical component of search and recommendation systems, altering the manner in which we engage with the extensive data available online. Consider the task of searching through a vast collection of YouTube videos using queries that necessitate an exact match of the title, author, or other criteria that are straightforward for machines to index. Such tasks are ideally suited for a relational database utilizing a language such as SQL. However, when it comes to more nuanced queries like "romantic music," simple similarity metrics based on the number of shared words between phrases are insufficient. For example, the query "climate change" is semantically closer to "global warming" than to "climate control," despite not sharing any words with the former and sharing one word with the latter. AI's role is crucial in enhancing our ability to comprehend and cater to user preferences across various platforms, including Spotify, TikTok, Instagram, and YouTube, in a dynamic and complex manner.

By transforming real-world entities such as text, images, and audio into mathematical representations called vector embeddings, we can capture the nuanced meanings and relationships within the data, enabling more precise and context-aware comparisons. For instance, a song's audio features can be represented as a high-dimensional vector, allowing for the calculation of similarity between songs based on their audio characteristics. This approach, known as embedding-based search, is essential for answering queries that require semantic understanding rather than simple indexable properties. By training machine learning models to map both queries and database items into a common vector embedding space, the distance between embeddings reflects their semantic similarity, with similar items positioned closer together. Identifying closely positioned items, also referred to as the nearest neighbor problem, is a well-studied issue in computer science. Many approximate nearest neighbor (ANN) search algorithms have been developed to efficiently find the most similar vectors in high-dimensional space. These algorithms are designed to provide fast and accurate search results, making it feasible to handle the vast and complex datasets typical in modern AI applications. The ann-benchmarks shows results of different ANN algorithms in this area: Google has developed ScaNN(Scalable Nearest Neighbor), Facebook has created FAISS, and Spotify has implemented Annoy. These systems comes with challenges, particularly around privacy and data management. Training embedding-based search AI models often requires the collection of vast amounts of personal data, raising significant privacy concerns. The enormous cost of datacenters means that only huge Big Tech companies can typically afford such infrastructure. As a response to these challenges, decentralized learning[?] paradigms are emerging as a promising alternative. These paradigms offer superior privacy, security, and scalability by distributing the learning process across multiple nodes, thereby reducing the reliance on centralized data collection and processing. However, federated learning, one of the most well-known decentralized learning[?] paradigms, still has the drawback of needing to aggregate model updates to a central server, which coordinates the training process across distributed nodes. Consequently, it fails to fully achieve the defining feature of the internet of fully decentralized control. The federated learning technique cannot deliver the true decentralized autonomy envisioned for distributing AI models.

Developers can now perform inference on edge devices independently, without requiring continuous communication with central servers, thanks to the advent of TensorFlow Lite and the growing computational power of mobile devices. Furthermore, to augment local processing capabilities,

Google's open-source ScaNN library can be incorporated into TensorFlow Lite models. In this research, we aim to decentralize the Internet by exploring the feasibility of creating a decentralized Spotify-alternative with real Web3 Youtube playback. We leverage a ScaNN-powered AI model and a peer-to-peer architecture to develop "Beyond Federated", a solution in which all participants are autonomous and self-sovereign, collaborating as equals. Each participant possesses equal power and supports all features, with no participant temporarily acting as a leader or coordinator. This decentralized approach ensures that the power and control over the learning process are evenly distributed, mitigating the risks associated with centralization and enhancing the robustness and resilience of the system.

Our research contributions are the following:

1) **Enabling collaborative learning:** Developing robust communication protocols and efficient algorithms for peers to share data and model updates, fostering the collective refinement of knowledge and leading to better, more consistent models across the network.
2) **Facilitating dynamic embedding learning:** Implementing mechanisms for peers to dynamically adjust vector representations based on their local data and interactions, resulting in contextually relevant and personalized models that adapt to the evolving data landscape.
3) **Ensuring privacy-preserving learning:** Exploring privacy-preserving techniques like federated learning and homomorphic encryption to allow peers to contribute to the learning process without directly sharing sensitive data, thereby addressing privacy concerns inherent in decentralized learning.
4) **Robustness to node failure:** Gracefully handles the failure of neighboring peers.

The rest of the paper is organized as follows: Section **??** provides an overview of the background, Section **??** details the design and implementation of the proposed system, Section **??** presents the experimental results and evaluation, and Section **??** concludes the paper with future work directions.

II. Problem Description

A common method to measure YouTube embedding similarity is through their inner product, leading to maximum inner-product search (MIPS). Given the potential size of databases, which can range from millions to billions of items, MIPS often becomes the computational bottleneck, making exhaustive search impractical. Therefore, we employed SCaNN, which trades off a bit of accuracy for significant speed improvements over brute-force search. SCaNN performs as a state-of-the-art solution for MIPS according to the ann-benchmark, focusing on compressing database items and allowing for the approximation of their inner product in a fraction of the time required by brute-force methods.

This compression is achieved through a novel learned quantization approach, which involves training a codebook of vectors from the database to approximately represent its elements. Once the codebook is trained, a search query can be compressed similarly, and the inner product between the query and each database embedding can be computed efficiently. However, a database of millions to billions of items would still require an exhaustive search across all database embeddings to find the nearest neighbor.

To address this, SCaNN clusters database embeddings into k clusters based on their distance using k-means clustering to reduce the search space. This approach requires all the data to be available and retrained to optimally cluster it into k clusters. In centralized learning, this is not a problem since all the data is available in one place. However, in our scenario, data is distributed across multiple devices, and we need to update the model with new data without requiring the entire dataset to be retrained.

Given the limited computational resources of edge devices, and the privacy constraints it would be infeasible to retrain the entire dataset. Therefore, the goal of our system is not only to retrieve the top approximate nearest neighbor but also to learn and retrieve newly added YouTube embeddings efficiently. Additionally, the challenge extends to learning new embeddings based on data from other nodes in the network. To facilitate this, anonymous clicklog data is shared across nodes, allowing for collaborative learning without compromising user privacy. This shared clicklog data enables the system to continually improve its understanding of user preferences and behaviors, ensuring that the embeddings are constantly updated and refined based on the latest interactions. This approach leverages the collective knowledge of the network, enhancing the overall performance and accuracy of the decentralized YouTube search system.

III. Architecture of Beyond Federated

In this section, we present an overview of the Beyond Federated system architecture. Beyond Federated is a proof of concept designed to demonstrate the feasibility of decentralized search and retrieval machine learning models that respect user privacy. This proof of concept includes a functional android application aimed at creating the first decentralized Spotify alternative, featuring Web3 YouTube playback integration. Our implementation focuses on YouTube content to define the scope, but the underlying architecture is versatile enough to handle any type of content. The app consists of four components which include the *user interface* to let the user interact with the application, the actual pre-trained *search model* containing all the learned embeddings, *TensorFlow Lite Support* a library to deploy .tflite models onto the mobile devices and lastly our peer-to-peer gossip network component.

A. User Interface

The Beyond Federated user interface features a simple and practical design, allowing users to search for YouTube videos, which are displayed in a scrollable list. This list dynamically updates based on events triggered when the text box content is modified. Users can also play a video from the list by clicking on a list item, which opens a new page that loads

the YouTube video based on its videoID into the Android native video player. This player utilizes YouTube's own web player to ensure 100% compliance with YouTube's terms of service. Additionally, users can insert new YouTube videos by providing the title, author, and video URL via the 'plus' icon located in the bottom right corner. Figure [?] illustrates the three different screens of the Beyond Federated application.

B. TensorFlow Model

Each distributuion of the application contains a pretrained version of the model to allowing the user to already start searching for content when the application is installed. The model has two important responsibilities. At first it will transform the search query into a high dimensional vector which is called an embedding. This is done through using pretrained models specifiically designed for this task. The model Secondly it contains our pretrained ScaNN artifacts.

1) Text Embedding Models: Careful consideration must be given to the choice of text embedding model, as it significantly impacts the accuracy of the search model. The text embeddings define the vector space where closely positioned vectors are identified, directly influencing the relevance and precision of search results. Given that our system focuses on YouTube content, it is crucial that the text embeddings effectively capture the semantics of the search queries. We utilize the titles and authors of YouTube videos since users commonly search for music on YouTube by song title and artist. Incorporating more attributes such as genre, intensity or video description in future research could further enhance search results.

Our embedding data setup is shown in Table I. This configuration allows the embeddings to search for the artist and title of the YouTube video and return the necessary metadata for our user interface. This setup ensures that users receive relevant and precise search results based on their queries, enhancing their overall experience.

Embedding		Metadata (JSON Formatted)
$\{artist\}$	$\{title\}$	artist , title, youtubeID

TABLE I: Overview of different pre-trained models and their trained dataset.

TensorFlow supports various embedding models, including BERT-based models and Universal Sentence Encoder (USE) models. The key differences between these models lie in their design and application. USE produces a single embedding for an entire sentence, making it efficient for tasks requiring a general understanding of sentence semantics. It is designed to work out-of-the-box for many applications without needing task-specific adjustments. In contrast, BERT generates fine-grained, context-sensitive embeddings for each token in a sentence, with the ability to aggregate these embeddings for sentence-level tasks using the [CLS] token. This makes BERT more suited for applications requiring detailed contextual information and often requires fine-tuning to achieve optimal performance for specific tasks.

Additionally, custom text embedding models are supported as long as they have an input text tensor of kTfLiteString and at least one output embedding tensor (kTfLiteUInt8/kTfLiteFloat32). This flexibility allows for the integration of specialized models tailored to specific requirements, enhancing the system's adaptability and effectiveness. While large embedding models are known to provide more accurate results, they also require more computational resources, making them less suitable for edge devices. Therefore, the choice of embedding model should strike a balance between accuracy and efficiency, ensuring optimal performance on mobile devices.

Training ScaNN configuration After receiving embeddigns from our text embedding model, we quantize the embeddings using the ScaNN algorithm. Quantization therefore requires a codebook to be trained on the embeddings. The codebook is trained by subdividing the embeddings into smaller subvectors, applying k-means clustering to create codebooks for each subvector set, and encoding the original vectors into compact representations using these codebooks.

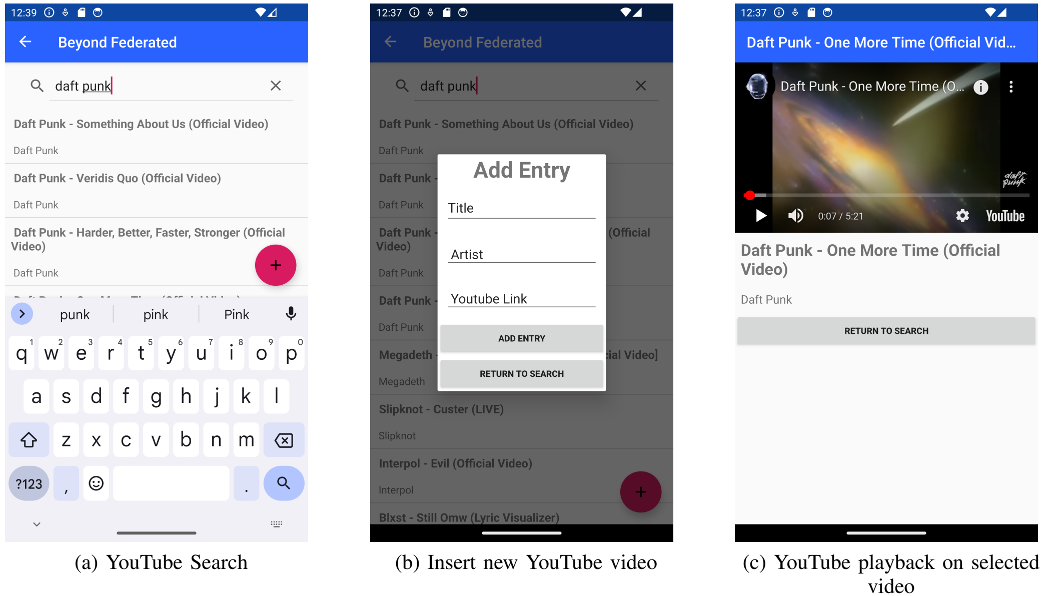
Both the codebook and the clustered partitions are stored in the format of a LevelDB which is a high performant but simple key-value storage. This index is stores inside the metaadata of the model in format of a flatbuffer.

After extensive research the index is build as follows:

By using these diverse datasets, we aim to thoroughly evaluate the system's robustness, scalability, and ability to manage various types of metadata. This approach ultimately contributes to our goal of developing a fully decentralized search and recommendation AI system.The datasets are summarized in Table II.

For training the ScaNN artifacts, we use the following parameters to ensure efficient and accurate similarity search:

- **Number of Clusters/Partitions** (σ): The number of clusters or partitions is determined by $\sigma = \sqrt{N}$, where N is the total number of items in the dataset. This balances search speed and accuracy.
- **Distance Measure**: We use the `dot_product` method to measure the distance between embedding vectors. The negative dot product value is computed to ensure smaller values indicate closer proximity.
- **Tree Structure** (ω): The number of partitions to search through, which reduces computational load while maintaining high search accuracy.
- **Quantization (score_ah)**: Float embeddings are quantized to int8 values, retaining the same dimensionality. This reduces the memory footprint and speeds up the search process without significantly compromising precision.
- **Dimensions per Block**: This parameter specifies the number of dimensions in each Product Quantization (PQ) block. For example, a 12-dimensional vector with



(a) YouTube Search (b) Insert new YouTube video (c) YouTube playback on selected video

Fig. 1: Screenshot of Beyond Federated application userinterface

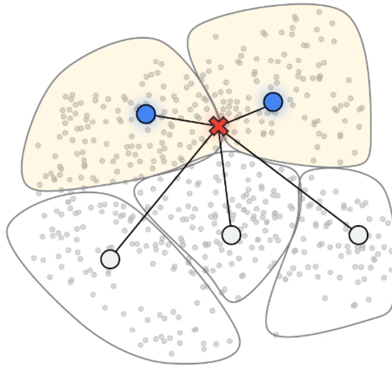


Fig. 2: Visual representation of SCaNN.

dimensions per block set to 2 results in six 2-dimensional blocks. This parameter is set to 1 for all datasets.

- **Anisotropic Quantization Threshold**: This parameter penalizes quantization errors parallel to the original vector differently than orthogonal errors, with a recommended value of 0.2.
- **Training Sample Size**: The number of database points sampled for training the K-Means algorithm for PQ centers.
- **Training Iterations**: Specifies the number of iterations to run the K-Means algorithm for PQ, with a default of 10 iterations for all datasets.

C. TensorFlow Lite Support

TensorFlow Lite Support is a library that enables the deployment of TensorFlow Lite models on edge devices. It provides APIs for loading and running TensorFlow Lite models, allowing for running inference on the models stored locally. The library is written in C++ and uses Bazel for cross-platform building, supporting Java, C++, and Swift. TensorFlow Lite Support is designed to be lightweight and efficient, making it suitable for deployment on mobile and IoT devices with limited computational resources. While ScaNN is actively developed as seperate repository by google it only provides a python interface. However TensorFlow Lite Support has included an older version of ScaNN into their library. As stated by their documentation TensorFlow Lite Support includes a simplified version of ScaNN that requires less resources to run and only for inference. There's no support for K-Means partitioning training and quantization training.

TensorFlow Lite utilizes a pretrained codebook and partitions, which are included in the model as a LevelDB index. This index is mapped directly using the index from the file as an immutable table. Consequently, a LevelDB instance is available in TensorFlow Lite, which is then accessed through SCaNN.

After embeddings are generated, the codebook is used to calculate the closest partitions. The contents of these partitions, which are hashed embeddings, are then fetched. These hashed embeddings are stored as integer values, representing their large floating point values, such as $[0.21021, -0.11321110, \ldots, \text{Embedding Dimension}]$, in a more compact form like $[2, 213, 12, \ldots, \text{Embedding Dimension}]$. This approach reduces storage space and allows for fast hash comparisons to determine the closest matches.

Additionally, LevelDB stores metadata keys, which can be retrieved after finding the closest hashes. This allows the system to fetch the corresponding metadata efficiently. The combination of these techniques ensures that TensorFlow

Lite can perform quick and accurate similarity searches in a resource-efficient manner, leveraging the power of pretrained codebooks and partitions stored in LevelDB.

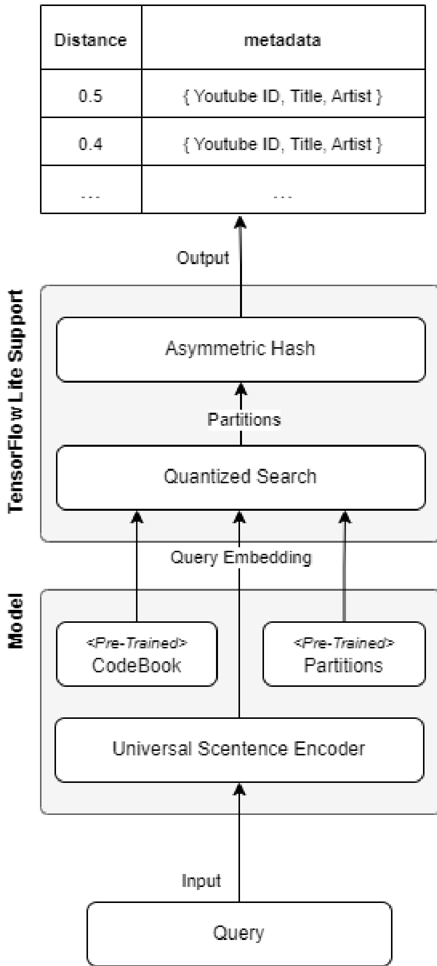


Fig. 3: Beyond Federeated Search model Call Graph.

The limitation of not being able to support K-mean partitioning and quantinazation training limitation occors because when on forehand of training k-means clustering it is required to specify the number of clusters. Inserting new clusters would require re-partitioning possibly multiple partions. This is computationally expensive and would require a lot of resources. Besides that during runtime the LevelDB Table uses an read-only inmutable table for better performance however this makes it not possbile to insert new items.

Since the training of new embeddings is not supported we continue our research by developing a custon a Non-Perfect Insert (NPI) method. The NPI method involves embedding the query, quantizing it using the pre-trained codebook, and appending it to the closest cluster. By doing so, we can simulate an dynamically enviroment. It also explores potentially research areas to improve on for further work. This approach enables us to evaluate the performance of the system under different conditions and analyze the impact of non-perfect inserts on search performance. By testing this method, we aim to understand how the system copes with new data and to ensure it maintains a high level of accuracy and efficiency despite the constraints imposed by non-dynamic scalability.

For handling non-perfect inserts, we employ the same model inference to generate an embedding from the newly inserted title and artist. This embedding is then used to find the closest partition. Given that our table is immutable, direct insertion of new embeddings is not possible. Instead, we follow a specific process to incorporate the new data:

1) **Generate Embedding**: We use the existing model to generate an embedding for the new title and artist.
2) **Find Closest Partition**: This embedding is then used to determine the closest partition based on the pretrained codebook.
3) **Fetch and Update Partition**: Since the table is immutable, we fetch the entire partition that the new embedding would belong to.
4) **Insert New Embedding**: The new embedding is inserted into the fetched partition.
5) **Create New Index**: A new index is created to reflect the updated partition.
6) **Overwrite Mapped File**: The nmapped file is overwritten with the new index and partition data.
7) **Reload File**: The entire file is reloaded to ensure the system can query the model using the newly inserted data.

This process, though requiring the reloading of the whole file, allows the system to update and query new data effectively. By reloading the file after updating the index, we ensure that the system can handle queries with the newly inserted embeddings, maintaining the integrity and accuracy of the decentralized YouTube search system.

D. *Gossip Network Protocol*

Beyond Federated is built on top of the Tribler SuperApp, leveraging its decentralized peer-to-peer network to ensure secure communication between users. This component enables robust, decentralized interactions through the IPV8 protocol, which facilitates secure data exchange.

In our decentralized YouTube search system, each node operates as a self-sovereign entity, maintaining control over its own data and operations. We employ the IPV8 protocol for communication between nodes, which ensures secure and efficient peer-to-peer interactions.

The self-sovereign nature of each node allows for independent operation and decision-making. When nodes disconnect from the network, they simply do not receive model updates. However, the system remains robust and functional due to the decentralized architecture.

Each time a user clicks on a search result, the clicklog entry is created and initially stored new data using our non perfect insert method. The gossip protocol then takes over, periodically sharing this new clicklog entry with neighboring nodes. These neighbors, in turn, propagate the information to their neighbors, and so on, ensuring that the clicklog data eventually reaches all nodes in the network.

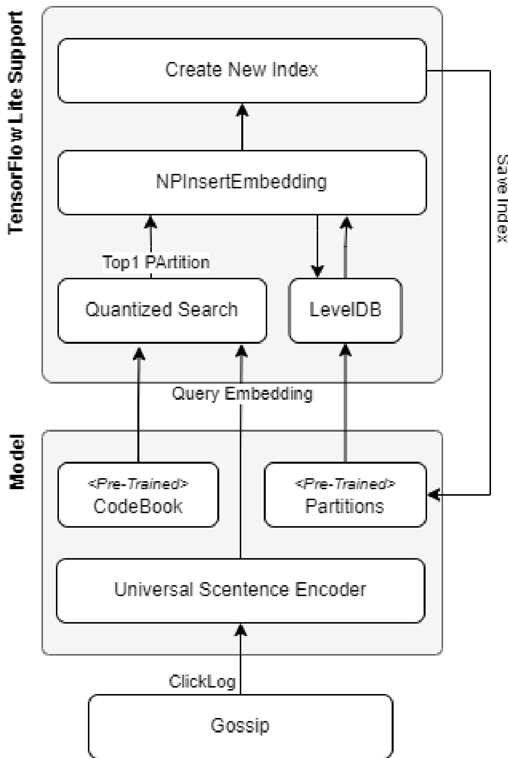


Fig. 4: The model structure in which an NPI is inserted. The NPI is responsible for inserting new items into the index without retraining the entire dataset.

The clicklog data is gossiped around the network, enabling the system to continue sharing information even if some nodes go offline. This gossiping mechanism ensures that the collective knowledge is maintained and disseminated across the network, supporting continuous learning and adaptation of the model.

IV. Experiments and Evaluation

In this section, we present the experiments and evaluations conducted to assess the system's potential as the first decentralized search and recommendation AI system. We will begin by discussing the datasets used for training the pre-trained model, followed by an explanation of the evaluation metrics. Finally, we will present and analyze the experimental results to evaluate the system's performance.

A. Experiment Setup

To ensure the experiments closely mirrored real-world conditions, they were conducted on an Android phone with the following specifications: Processor: Qualcomm Snapdragon 625, Cores 8 cores, Clock speed: 2GHz, Android 8, RAM 4GB, Storage: 64GB. When loading our model we allocate a maximum of 4 threads to perform model inference and ScaNN. For the model pretraining phase, we utilized Kaggle[1] notebook runs which contain the following specifications: CPU: 4 vCPU cores (Intel Skylake), RAM 30GB of RAM, Storage: 73.1GB.

B. Datasets, Embedders and Pre-trained Models

The main dataset used during the experiments is the Spotify and YouTube dataset from kaggle[2], which contains 20,230 songs from 2,079 artists. The dataset is released under the CC0: Public Domain license which is especially important when running our network experiment distributing contents of the dataset across the network. Before using this dataset, it was cleaned to remove redundant YouTube-specific extensions from the titles, such as "Official video," "music video," and "lyrics video," to ensure that our embedding space is not negatively affected. Often the titles of YouTube videos contain the artist's name, so the data was cleaned to remove the artist name from the title. This ensures that the model learns the semantic context of the title and artist separately.

Our second dataset, YouTube-Commons, is a large collection comprising 2,063,066 videos from 411,432 individual channels. These videos are shared on YouTube under a CC-BY license. The dataset predominantly features English-speaking content, accounting for 71% of the original languages. This extensive dataset tests the system's scalability and performance with a vast amount of data, providing insights into how well the system handles large-scale decentralized information.

To embed the datasets and create our vector space, we utilized two pre-trained models, not all embedder models are suitable due to their requirement of running in an on-device scenario. Therefore we train two models on the same Spotify Youtube Dataset using both the Universal Sentence Encoder (USE) and BERT. The basic Universal Sentence Encoder is around 1GB in size and is not optimized for on-device inference therefore we will be using a retrained USE from colab[3]. This results in an 27.3MB model being able to encode embeddings in 6ms. BERT does have a mobile version called MobileBERT we will specifically use mobilebert_qa which has 4.3x smaller and 5.5x faster than BERT-Base while achieving competitive results, suitable for on-device scenario as its 5.83MB.

The details of the pre-trained models and their respective datasets are summarized in Table II.

1) Metrics: The performance of the system will be evaluated using two primary metrics: speed and accuracy.

The speed of the system is assessed through execution time, measured in milliseconds (ms), and memory usage, measured in megabytes (MB). The accuracy of the system is evaluated using the Recall metric. Recall is a crucial metric in the evaluation of information retrieval systems, including search engines and recommendation systems. It is defined as the ratio of relevant items retrieved by the system to the total number of relevant items in the dataset. Mathematically, recall is expressed as:

$$\text{Recall} = \frac{\text{Number of relevant items retrieved}}{\text{Total number of relevant items}} \tag{1}$$

Recall is particularly important in contexts where missing relevant items can have significant consequences resulting in user dissatisfaction due to the inability to find pertinent

Embedder - Dataset	Clusters	Embedding Dimension	Video's	Model Size
BERT - Spotify and Youtube	140	100	20,230	9.21MB
USE - Spotify and Youtube	140	128	20,230	31.3MB
USE - YouTube-Commons	1450	128	2,063,066	458MB

TABLE II: Overview of different pre-trained models and their trained dataset.

YouTube video's. By measuring both speed and recall, we can comprehensively evaluate the performance and effectiveness of the system, ensuring it meets the desired criteria for both efficiency and accuracy.

C. Content Search on the Edge Experiment

The goal of this experiment is to provide a general overview of the capabilities of Beyond Federated. In this initial test, we issue a single query to the system and evaluate its results.

By starting with a query, we aim to understand the basic functionality and effectiveness of our system's search capabilities. This approach allows us to identify any immediate issues and establish a baseline for more complex, multi-query scenarios in subsequent experiments. This preliminary assessment helps in verifying that the core components of the search mechanism are operating correctly and sets the stage for more detailed performance evaluations and refinements in future tests. Our query consists of the band name "red hot chili peppers" to retrieve the top 10 results from the Spotify YouTube Trained Model.

table with results

1) Results: The search query results for the band "Red Hot Chili Peppers" in the Spotify YouTube Trained Model are detailed in Table 1. The search execution time was remarkably efficient, clocking in at just 37 milliseconds. This rapid performance is maintained even with the retrieval of additional items, as the distances are computed within the six closest clusters to the original query. In terms of accuracy, 9 out of the top 10 results are directly relevant to the search query, yielding a recall rate of 90%. However, the tenth result is from a different band, "The Beach Boys," which does not align with the intended query. This discrepancy may be explained by the absence of the "Red Hot Chili Peppers - Snow (Hey Oh)" video, which was located at the 14th place with a distance of -0.68120 to our search query. The sentence encoder model, designed to capture the semantic context of queries, likely associated "The Beach" with "Hot" conceptually, while "Snow," being contradictory to "Hot," in the missing entry's title may have positioned its vector further away in the semantic space. This outcome underscores the system's capability to interpret the semantic context of queries and retrieve items based on the learned embeddings, though it also highlights areas for potential improvement in handling nuanced semantic relationships.

D. Random keyword inference on different Embedders Experiment

In this experiment, we aim to evaluate the impact of different encoder models when our vector space is trained. We will compare the performance of the Universal Sentence Encoder (USE) and BERT models in terms of speed and accuracy. The experiment will involve issuing different type of queries to the system using both encoder models and analyzing the results to determine the most effective model for our decentralized search and recommendation system. We aim to evaluate the effectiveness of the semantic embeddings by measuring the system's performance on a set of seven related queries. These queries are designed to test various levels of specificity and semantic understanding, ranging from exact matches to partial and semantically altered queries. We are searching for the band UB40 which consost of way less than the context given in our first expirement. This band also has 10 YouTube videos learned. The queries include:

1) Results: Analyzing the search times for different queries using the Universal Sentence Encoder (USE) and BERT models reveals a significant difference in performance. The USE consistently outperforms BERT in terms of speed, completing all queries in approximately 38 milliseconds, compared to BERT's 85 milliseconds for the same queries. Despite this difference, both models are capable of handling search queries in real-time, demonstrating their suitability for time-sensitive applications.

To further assess the impact of larger datasets, we trained a model on the YouTube-Commons dataset. Interestingly, the CPU time does not degrade significantly despite the increased number of comparisons, as the system efficiently measures distances to larger clusters. The results of this experiment are summarized in Figure [?].

Assessing the recall in this example is somewhat complex for both embedding models. Both models successfully retrieve the correct video when queries include the full song title or the combination of the artist's name and the song title. However, their performance diverges significantly with other types of queries.

a) Band Name Only: When searching solely for the band "UB40," BERT finds two videos that do not include the title "Red Red Wine," whereas USE fails to retrieve any videos including UB40 as the artist.

b) Partial Song Name: The results for the query "Red" differ between the models. USE predominantly retrieves videos related to the color red, while BERT identifies videos associated with red objects, such as roses. This indicates that USE is more focused on color semantics, whereas BERT captures a broader range of associations with the word "red."

c) Semantic Mistake: For the semantically incorrect query "Green Wine," BERT's results are influenced by the term "poison," and it eventually returns the correct video in

Rank	Distance	Metadata
0	-0.92737	artist: Red Hot Chili Peppers, title: Can't Stop, id: 8DyziWtkfBw
1	-0.92040	artist: Red Hot Chili Peppers, title: Otherside, id: rn_YodiJO6k
2	-0.91266	artist: Red Hot Chili Peppers, title: Californication, id: YlUKcNNmywk
3	-0.90260	artist: Red Hot Chili Peppers, title: Dark Necessities, id: Q0oIoR9mLwc
4	-0.88711	artist: Red Hot Chili Peppers, title: Dani California, id: Sb5aq5HcS1A
5	-0.87163	artist: Red Hot Chili Peppers, title: Give It Away, id: Mr_uHJPUlO8
6	-0.83989	artist: Red Hot Chili Peppers, title: By The Way, id: JnfyjwChuNU
7	-0.83138	artist: Red Hot Chili Peppers, title: Scar Tissue, id: mzJj5-lubeM
8	-0.78570	artist: Red Hot Chili Peppers, title: Under The Bridge, id: GLvohMXgcBo
9	-0.72377	artist: The Beach Boys, title: Good Vibrations, id: apBWI6xrbLY

TABLE III: Top 10-Results searching "Red Hot Chili Peppers" in the 20K Spotify YouTube Trained Model.

Description	Example
Full song name and band name	"UB40 Red Red Wine"
Band name only	"UB40"
Song name only	"Red Red Wine"
Partial song name	"Red"
Semantic mistake in the song name	"Green Wine"
Foreign language translation of the partial song name	"Vino tinto" (Spanish for "Red Wine")

TABLE IV: Examples of Song and Band Name Variations

the fourth position. In contrast, USE accurately returns th correct video as the top result, demonstrating its robustness i handling semantic errors.

d) Foreign Language Translation: The foreign languag query "Vino tinto" does not perform well with either mode which can be attributed to the dataset being in English an the models being trained on English text. However, USE doe return related foreign titles, such as those by Vasco Ross suggesting some degree of cross-linguistic capability.

2) Discussion: The results highlight several key observa tions about the performance and behavior of the USE an BERT models. Both models don't perform on relatively small queries in which no real words are being used. USE's speed advantage makes it more suitable for real-time applications, while BERT's broader semantic understanding can be beneficial in scenarios requiring nuanced context comprehension. Both models show limitations in handling foreign language queries, pointing to a potential area for future improvement. Overall, these findings underscore the importance of selecting the appropriate model based on the specific requirements of the application, whether it be speed, recall, or semantic understanding.

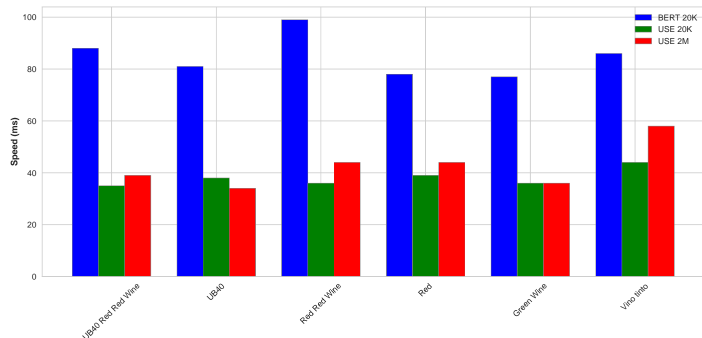


Fig. 5: Measuring the search time for different queries using the Universal Sentence Encoder and BERT models.

E. Non-Perfect Insert Experiment

In this experiment, we evaluate the system's ability to handle the insertion of new entries and measure the execution time required for these operations. This is crucial for ensuring the scalability and efficiency of the system as it updates its database with new information. The experiment is designed to demonstrate the basic functionality of single item insertion, evaluate the system's performance under a moderate load by inserting 10,000 items, and conceptually assess the scalability by estimating the performance for repeated insertions up to one million times. When we add an new song we will be using a random generated list of fantasy arists names and title, inlcusing 2 random english words such as "artist: Eternal Ballad title: Crystal Phoenix".

1) Results: The system successfully inserted a new item into our YouTube Spotify dataset, resulting in a model size increase of 876 bytes. The distance of the new item from the original query was -2.26156, which is relatively uncommon compared to other previously learned videos. The insertion operation took 574 milliseconds to complete, with a RAM usage of 16 MB. These results demonstrate the system's ability to efficiently handle the insertion of new items, even when they do not perfectly match the existing dataset.

However, when inserting into the larger pretrained model, which includes YouTube Common video items, we observed an increase in both insertion time and RAM usage. This bottleneck is due to the need to read, initialize, and overwrite the entire index, which has a size of 458 MB. After writing the new index, it is necessary to reinitialize the database to enable searching.

Inserting approximately 8 new songs results in significant differences, where the YouTube Commons 2M Dataset requires an infeasible amount of time to rebuild the index for each new entry. Since it is not necessary to search all insertions

directly, we also experimented with adding entries first and rebuilding the index after all songs had been added. This approach significantly reduced the insertion time.

We observed that batch processing of insertions followed by a single index rebuild allowed us to manage the database more efficiently. By delaying the rebuild process until after all insertions, we reduced the cumulative time and resource usage. For instance, when inserting 8 new songs in the larger dataset, the rebuild time, though substantial, was offset by the reduced frequency of rebuild operations, leading to a more manageable system load.

This approach, while reducing the frequency of resource-intensive operations, still faced challenges. Specifically, the size and complexity of the dataset necessitated careful management of memory and processing power. Nevertheless, the results indicated a clear benefit in handling multiple insertions as a batch process rather than individually.

When adding 1,600 songs, we eventually observed that even rebuilding a smaller index becomes slower. While the larger dataset takes a bit longer to initialize, it still outperforms the batch insert version of the non-perfect insert.

The performance gap widened as the number of insertions increased. Inserting 1,600 songs individually into the larger dataset required a significant rebuild time for each insertion, cumulatively leading to substantial delays. However, by batch processing these insertions, we managed to reduce the overall time significantly. The larger dataset, with its more extensive index, benefited from fewer rebuild operations, demonstrating the efficiency of handling bulk insertions.

We also noted that the initialization time for the larger dataset, although initially longer, did not scale linearly with the number of insertions. This suggests that the system's architecture is capable of handling large-scale data more effectively when optimized for batch processing. As the batch size increased, the efficiency gains became more pronounced, highlighting the importance of strategic insertion management in maintaining system performance.

When adding 10,000 songs, the index size did not increase substantially. However, the insertion process for 10,000 songs still required 200 and 500 seconds, respectively, for the smaller and larger datasets.

Inserting 10,000 songs into the system provided a comprehensive test of its scalability. Despite the relatively modest increase in index size, the insertion time revealed significant differences between the datasets. For the smaller dataset, the process took approximately 200 seconds, reflecting a relatively efficient handling of the bulk insertion. In contrast, the larger dataset required around 500 seconds, underscoring the additional complexity involved in managing a more extensive index.

These results indicate that while the system can handle a large volume of insertions, the efficiency is heavily dependent on the dataset size. The larger dataset's longer insertion time can be attributed to the increased overhead in managing a more complex index. However, the fact that the index size did not grow significantly suggests that the system's design is effective in maintaining a compact and manageable data structure.

The experiment demonstrates the system's capacity to manage the insertion of up to 10,000 new entries efficiently. However, scalability remains a critical concern as the number of insertions increases. To assess the potential of scaling up to one million items, several factors need to be considered:

Index Management: As the number of entries grows, the complexity and size of the index will increase. Efficient index management strategies, such as hierarchical indexing or partitioned indices, could mitigate the performance bottlenecks observed with larger datasets. Additionally, optimizing the index rebuild process to handle bulk insertions more effectively would be essential.

Memory and Processing Power: Handling a million entries will require substantial memory and processing power. Ensuring that the system can scale its resources dynamically in response to increased load is vital. Implementing distributed processing and leveraging cloud-based solutions could provide the necessary scalability.

Batch Processing: The results indicate that batch processing significantly improves performance. For one million items, batch processing would need to be optimized further. Strategies such as parallel processing of batches and incremental indexing could help manage the load more effectively.

Next if we look at new queries from our updated index we stated the following. The favoring of recently inserted songs can lead to a significant decrease in the overall recall rate of the system. As more items are inserted without retraining the codebook, the likelihood of retrieving correct items diminishes. This trend suggests that the system's effectiveness in retrieving relevant and accurate entries will degrade over time unless periodic retraining is implemented.

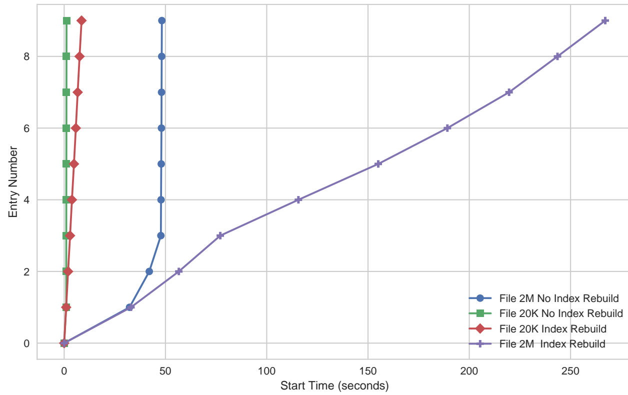
To maintain high recall rates and ensure that the retrieval system remains accurate and effective, it is essential to consider strategies for regular retraining of the codebook. This will help in recalibrating the retrieval process and balancing the influence of newly inserted songs with existing entries.

F. Decentralised content discovery and search Experiment

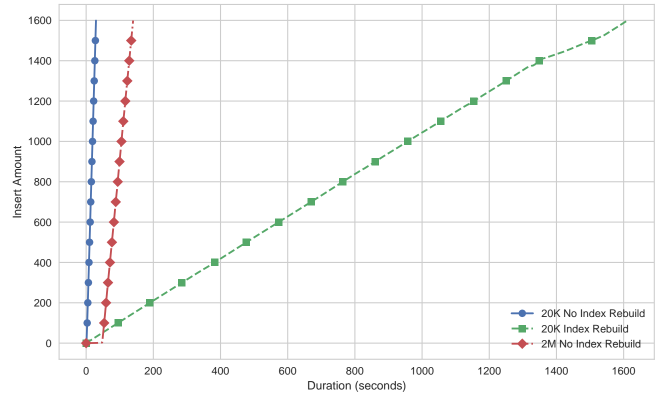
Our final experiment examines the entire end-to-end pipeline of decentralised content discovery and search In this section, we quantify the exact cost of content discovery and decentralized search. Content discovery is based on a gossip protocol, as illustrated by the network view available. We focus on the Creative Commons YouTube dataset, which contains videos that can be freely redistributed.

Our experiment centers on the core primitive of two random peers exchanging discovered content and search results. The search results are organized in a format known as a ClickLog, which consists of pairs of ”Query, Clicked-YouTube-URL.” One device in our experiment generates the ClickLog, while the other device inserts these results into SCANN (Scalable Nearest Neighbors).

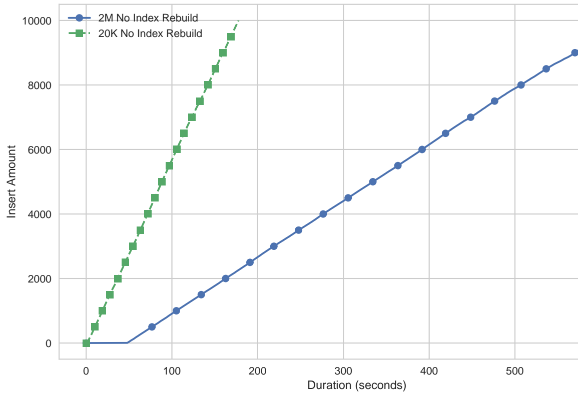
To investigate the data exchange size, we set up two devices to gossip a single ClickLog each second. This setup



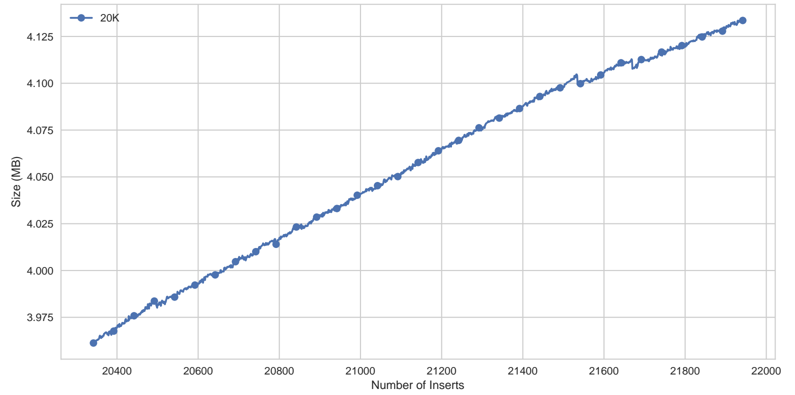
(a) Insert of 8 fantasy Youtube Items



(b) Insert of 1600 fantasy Youtube Items



(c) Insert of 10K fantasy Youtube Items



(d) Index size growht on 10K fantasy Youtube Items

Fig. 6: Results of Non-Perfect Insert

Rank	Distance	Metadata
0	-1.64961	artist: Eternal Ballad title: Crystal Phoenix, id:abc123456
1	-1.53666	artist: Crystal Ballad, title: Phoenix Moonlight, id:abc123456
2	-1.53307	artist: Mystic Symphony, title: Ember Mystic, id: abc123456
3	-1.50501	artist: Moonlight Ballad, title: Mystic Eternal, id:abc123456
4	-1.49494	artist: Phoenix Ballad, title: Eternal Crystal, id:abc123456
5	-1.47767	artist: Whisper Ballad, title: Crystal Mystic, id:abc123456
6	-1.47767	artist: Starlight Ballad, title: Phoenix Eternal, id:abc123456
7	-1.46832	artist: Moonlight Ballad, title: Moonlight Phoenix, id:abc123456
8	-1.45609	artist: Moonlight Ballad, title: Shadow Moonlight, id:abc123456
373	-0.94673	artist: Red Hot Chili Peppers, title: Under The Bridge, id: GLvohMXgcBo

TABLE V: Top 10 and selected additional results for the query ”Red Hot Chili Peppers Under The Bridge” in the Spotify YouTube Trained Model + Fantasy Youtube Entries.

helps us determine the volume of data exchanged and assess the feasibility of implementing Beyond Federated Learning. Understanding the data requirements is crucial for ensuring the efficiency and scalability of the decentralized search process. By quantifying the data exchanged during content discovery and search, we can better understand the implications for network load and system performance.

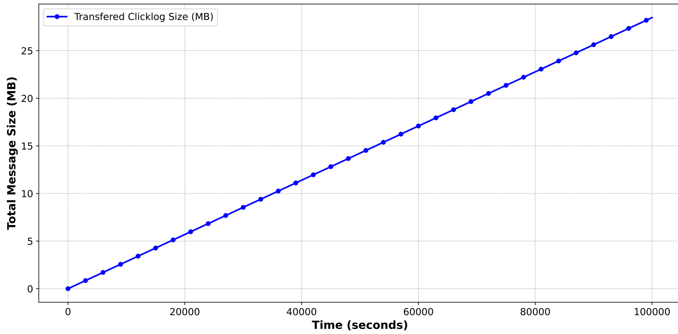


Fig. 7

V. Conclusion

The results demonstrate that the Beyond federated architecture using SCaNN can efficiently retrieve approximate nearest neighbors in a decentralized environment. This system effectively clusters YouTube embeddings using k-means clustering, which enables fast and accurate similarity searches. By testing the insertion of non-perfect entries, we showcase a functional use case of the system, illustrating its capability to learn and retrieve newly added YouTube embeddings even in a non-optimized state.

The decentralized nature of the system allows for data distribution across multiple devices while maintaining performance. Additionally, the system can learn new embeddings based on data from other nodes within the network, leveraging shared clicklog data for collaborative learning. This collaborative approach ensures that user privacy is preserved, as data is not centralized.

The continuous integration of shared clicklog data enables the system to refine its understanding of user preferences and behaviors, leading to constant updates and improvements in the embeddings. This method harnesses the collective knowledge of the network, significantly enhancing the overall performance and accuracy of the decentralized YouTube search system. This research highlights the potential for creating a decentralized web3 YouTube platform that is both efficient and privacy-preserving, paving the way for more robust and user-centric decentralized applications.

A. Future work

Future work could explore k-means clustering methods that do not require access to the entire dataset. This approach would enhance the scalability and efficiency of the system, particularly for large and growing datasets.

Traditional k-means clustering needs the entire dataset to identify optimal centroids, which can be computationally intensive. Developing methods to perform k-means clustering incrementally or on representative samples could reduce this computational load and allow real-time clustering updates as new data is added.

This capability would be especially useful in a decentralized search system, where data is distributed and continuously updated. Incremental k-means clustering would enable the system to adapt to new data without exhaustive recomputation, maintaining performance and accuracy in dynamic environments.

VI. Appendix

References

[1] *KaggleResources*. URL: https://www.kaggle.com/docs/notebooks#the-notebooks-environment.
[2] *spotify-youtube-dataset*. URL: https://www.kaggle.com/datasets/salvatorerastelli/spotify-and-youtube.
[3] *USEREtrained*. URL: https://github.com/tensorflow/tflite-support/blob/master/tensorflow_lite_support/examples/colab/on_device_text_to_image_search_tflite.ipynb.