BeyondFederated

Truly decentralised edge AI

Master Thesis Computer Science

TU Delft

Quinten van Eijs

June 2024

BeyondFederated: Truly decentralised edge AI

– MSc. Thesis –

Quinten van Eijs
Delft University of Technology
Delft, The Netherlands

Johan Pouwelse
Delft University of Technology
Delft, The Netherlands

***Abstract*—Decentralized learning (DL) leverages edge devices for collaborative model training while enhancing data privacy, as training data never leaves the device. In this paper, we present "BeyondFederated: Truly Decentralised Edge AI," a novel approach to overcoming the limitations of traditional Federated Learning (FL) by eliminating reliance on centralized servers. Traditional FL, despite its decentralization benefits, still poses significant privacy and security risks due to its central control points. Our system leverages edge AI with TensorFlow Lite and a peer-to-peer gossip network to ensure fully decentralized learning and data processing. We developed a proof-of-concept to demonstrate a decentralized alternative to Spotify, incorporating Web3 YouTube playback, and enabling efficient and accurate Scalable Nearest Neighbor (ScaNN) searches using vector embeddings incorporated into an actual TensorFlow model running on edge devices. Through various experiments, we evaluated the system's performance in real-time search capabilities, embedding model comparisons, and new data insertion handling. The results confirm that BeyondFederated maintains high efficiency, scalability, and privacy. This study underscores the potential for truly decentralized machine learning and sets the stage for more robust and user-centric decentralized AI applications.**

***Index Terms*—Decentralised learning, Semantic Search, Peer-2-Peer, Gossip Protocol, Information Retrieval, Distributed systems.**

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[20], 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.

To map queries and database items into a common vector embedding space, we can train machine learning models that learn the semantic relationships between the query and the database items. The distance between embeddings reflects their semantic similarity, with similar items positioned closer together. The above mentioned is a well-studied issue in computer science called the nearest neighbor(NN) problem, which involves identifying the closest item to a given query within a dataset. It can be computationally expensive and impractical for large datasets to find the exact nearest neighbor for a given point. This resulted in the rise of approximate nearest neighbor(ANN)[2] search algorithms which finds a point that is close enough (approximate) to the nearest neighbor, trading off some accuracy for significant improvements in speed and resource usage. This makes ANN algorithms feasible for handling the vast and complex datasets typical in modern AI applications.

Modern AI applications which are known for similarity search through their complex datasets, such as Google, Facebook, and Spotify, have developed their own ANN algorithms. Google has developed ScaNN(Scalable Nearest Neighbor)[24], Facebook has created FAISS[19], and Spotify has implemented Annoy[21]. The ANN-benchmarks[4] show results of these different ANN algorithms. However, training ANN based search models often requires the collection of vast amounts of personal data, raising significant privacy concerns. As a response to the challenge, decentralised learning paradigms are emerging as a promising alternative. By distributing the learning process across multiple devices, these paradigms offer superior privacy, security, and scalability. Thereby the paradigms reduce the reliance on centralized data collection and processing. For example, a widely implemented decentralised learning

approach is Federated Learning(FL)[17], which enables the training of AI models across multiple devices while keeping data localized.

Besides the growing rise of decentralised learning, the computational capabilities of mobile devices are enhancing. Platforms such as Google's TensorFlow Lite, now enable running FL models directly on mobile devices. This allows mobile phones to collaboratively learn AI models while keeping all training data on the device. Despite these advancements, the current FL model retains some drawbacks of centralized learning, primarily due to its reliance on central control. The defining feature of the internet to fully decentralise central control can *not* be achieved with FL[16]. This research aims to explore the feasibility of creating a decentralised Spotify alternative with real Web3 YouTube playback. By leveraging a ScaNN-powered AI model and a peer-to-peer architecture, we propose "BeyondFederated", a solution in which all participants are autonomous and self-sovereign, collaborating as equals. In BeyondFederated, each participant possesses equal power and supports all features, with no participant temporarily acting as a leader or coordinator. This decentralised approach ensures that 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 FL and homomorphic encryption to allow peers to contribute to the learning process without directly sharing sensitive data, thereby addressing privacy concerns inherent in decentralised learning.
4) **Robustness to node failure:** Gracefully handles the failure of neighboring peers.

The paper is organized as follows: Section 2 outlines the main problems addressed in this research. Section 3 describes the design and implementation of the proposed system. Section 4 presents the experimental results and evaluation. Finally, Section 5 concludes the paper and discusses future work directions.

II. Problem Description

Traditional FL frameworks consist of a select set of individually connected devices refered to as clients. Each client processes its own raw data locally, ensuring the data remains private and is not shared with other entities. While processing the local data, each client trains a model and then shares only the model parameters with a central server. The central server aggregates these parameters to integrate the learning results from all individually connected clients, thereby creating a global model. Despite its decentralised training approach, FL remains susceptible to various privacy and security threats[28]. Attacks such as the gradient inversion attack, where attackers reconstruct user data from gradient updates, highlight a significant privacy risk. Additionally, the frequent communication of model updates between clients and the central server can leak sensitive information, even when raw data remains on local devices. The centralized server itself poses a security threat, as it can be targeted by attackers to compromise the entire system. Moreover, the diverse and heterogeneous nature of client data and resources complicates the model aggregation process, potentially leading to inconsistent and insecure outcomes. These issues collectively undermine the safety and effectiveness of FL in preserving user privacy and ensuring robust security.

Fully Decentralised Federated Learning(DFL) is gaining significant attention as a method to enhance the robustness, privacy, and scalability of machine learning systems. Numerous studies[29, 11, 27] are actively exploring DFLs potential and addressing the various challenges it presents. DFL demands extensive peer-to-peer communication, leading to increased bandwidth consumption and latency. Efficient communication protocols are vital to manage the heightened data exchange without degrading system performance. Additionally, each node must independently handle the computational load and storage requirements for local data and model updates. This can place a significant burden on devices with limited resources, complicating scalability and overall system efficiency. Ensuring data privacy and security also becomes more challenging in a decentralised environment. Sophisticated encryption and privacy-preserving techniques are necessary to protect against data leakage and malicious attacks.

From a logistical perspective, managing heterogeneous resources across different nodes presents another challenge. Nodes in a decentralised system often have varying computational capabilities, network bandwidths, and storage capacities, making balanced resource management complex. Effective coordination and orchestration are essential for maintaining efficient communication and model updates across the network. This includes peer discovery and task scheduling to ensure fair and efficient use of resources. Additionally, setting up and maintaining the infrastructure for a decentralised network involves considerable logistical efforts. Monitoring performance and managing the health of a decentralised system are more complex compared to centralized systems, requiring robust tools and techniques.

It shows that there is insufficient research done to support the implementation and adoption of DFL techniques while using mobile devices. Presenting these challenges, decentralised learning remains a nascent field with many open

research questions. None of the implementations found have yet achieved the full potential of decentralised learning, and much work is needed to realise the vision to fully decentralise FL.

III. Architecture of BeyondFederated

To demonstrate the feasibility of decentralised ANN based search models that respect user privacy, we present an overview of the BeyondFederated system architecture in this section. BeyondFederated is a proof of concept designed. This proof of concept includes a functional android application creating the first decentralised Spotify alternative, featuring Web3 YouTube playback integration. Although the implementation of the architecture focusses on YouTube content to define the scope, the underlying architecture is versatile enough to handle any type of content. The proof of concept consists of four components which include the *user interface* to let the user interact with BeyondFederated, 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 BeyondFederated user interface features a simple and practical design, allowing users to search for YouTube videos given a query text, 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. The video player library used in our application utilizes YouTube's own web player to ensure 100% compliance with YouTube's terms of service[3]. 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 1 illustrates the three different screens of the BeyondFederated application.

B. TensorFlow Model

The use of TensorFlow is beneficial due to the release of TensorFlow Lite, which enables TensorFlow models to be run on devices which are computationally limited. Each distribution of BeyondFederated contains a pretrained version of our TensorFlow model to allowing the user to already start searching for content when the application is installed. The model has two important responsibilities. The first one is that it will transform the search query into a high dimensional vector which is called an embedding. This is done through using pretrained models specifically designed for this task. Secondly the model contains our pretrained ScaNN artifacts to perform ANN.

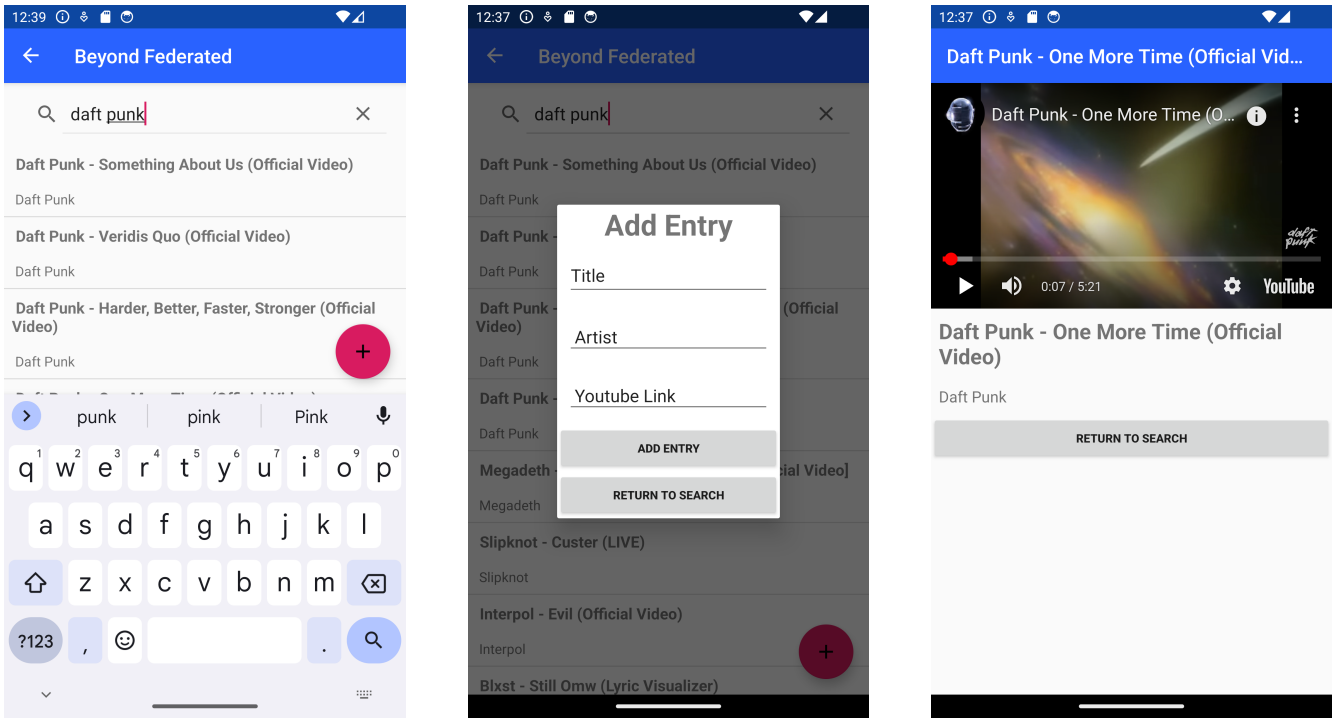
Careful consideration must be given to the choice of text embedding model, as it significantly impacts the accuracy of the search model. The text embeddings define the vector space where closely positioned vectors are identified, directly influencing the relevance and precision of search results. Given 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 including the YouTube video ID 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[8] and Universal Sentence Encoder (USE) models[6]. 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. 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 and at least one output embedding tensor. 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.

To find the YouTube songs that are considered to be closest to our search query, our generated embedding is used during the ScaNN approximate nearest neighbor search. Therefore, the model contains the pre-trained ScaNN artifacts required to eventually perform approximate nearest neighbor search. ScaNN operates by combining vector quantization[9] and asymmetric hashing[25] to perform efficient and accurate nearest neighbor searches in high-dimensional datasets. At the core of ScaNN's functionality is vector quantization, which reduces the complexity of the search space. This involves partitioning the high-dimensional space into smaller subspaces and representing these subspaces with centroids generated



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

Fig. 1: Screenshot of the BeyondFederated application user interface

using k-means clustering. These centroids are organized into a pretrained codebook, created during the initial training phase. The codebook acts as a compact representation of the dataset, enabling quick approximations of data points within each subspace. In addition to vector quantization, ScaNN employs asymmetric hashing to enhance search efficiency. Asymmetric hashing creates compact binary codes for data points, allowing rapid comparisons during the search process. A visual representation of ScaNN is shown in figure 2.

Both vector quantization and asymmetric hashing significantly reduce the computational load by enabling quick distance approximations between the query and data points stored inside the retrieved hash codes in the corresponding partition. During a search, ScaNN first leverages the pretrained codebook to quickly identify the most relevant subspaces that likely contain the nearest neighbors. By focusing on these subspaces, the number of distance computations is drastically reduced. Within the identified subspaces, asymmetric hashing is used to efficiently compare the query with the data points, ensuring both speed and accuracy. To enable our TensorFlow model to interact with the pretrained codebook and subspaces, these values are stored in a ScaNN configuration file after training. The configuration file created and the hashes of each embedding are organized into their respective partitions and comprehensive metadata of the YouTube item. Each embedding is collectively stored inside the LevelDB index(.ldb). LevelDB[14] is just like ScaNN developed by Google and provides a high perfromant Key-Value storage providing ordered mapping from string keys to string values. The LevelDB index is then converted into a FlatBuffers[1] binary and added to the metadata of the TensorFlow model.

C. TensorFlow Lite Support

TensorFlow Lite Support(TFLite Support) [26] is a library that enables the deployment of TensorFlow Lite models on edge devices. TFLite Support provides APIs for loading and running inference on our TensorFlow Lite model, allowing for running inference on the distributed model. The library is written in C++ and uses Bazel for cross-platform building, supporting Java, C++, and Swift. TFLite 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 being updated and improved by Google[12], TFLite Support includes a simplified version of ScaNN that requires less resources to run and only for inference.

Given our TensorFlow Lite model, TFLite Support initializes the metadata FlatBuffer (.ldb) index file through memory mapping (mmap)[18]. Memory mapping (mmap) is a technique that maps a file into the memory space of a process, allowing the file to be accessed as if it were part of the memory. This approach facilitates efficient and quick read and write operations, which are crucial for executing the ScaNN algorithm. After the closest subspaces are found given an embedding query, TFLite Support retrieves the hashed embeddings from the LevelDB table by requesting

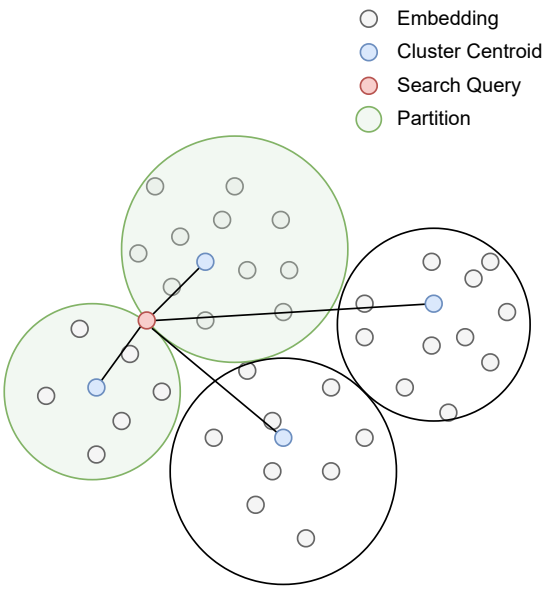


Fig. 2: Visual representation of SCaNN searching through the embeddings of the top 2 sub spaces(partitions).

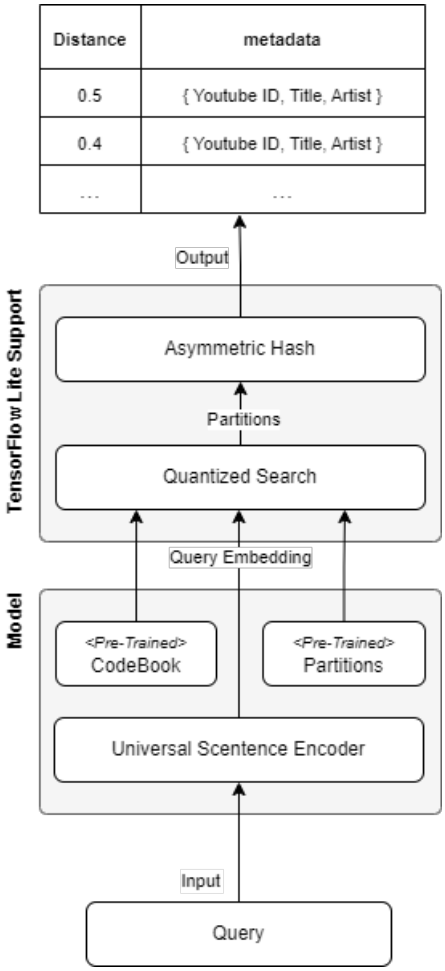


Fig. 3: Overview of the different components when perfoming a search operation.

the subspace key 'E_{subspace}'. The subspaces are often refered to as partitions, scann only calculates the distance between the query and the embeddings in the closest partition instead of the whole dataset. After the closest embeddings are calculated the metadata is also requested by 'M_{metadata}' from the LevelDB table and returning them for display in our user interface. An overview of how the different components opereate when searching is shown in figure 3

The current implementation of TFLite Support does not support on-device K-Means partitioning and quantization training. This limitation arises primarily because K-Means clustering requires a specified amount of clusters before training. Inserting new clusters post-training would necessitate re-partitioning, potentially affecting multiple partitions. This re-partitioning process is computationally expensive and demands significant resources, making it impractical for on-device execution. Additionally, during runtime, the LevelDB table employs a read-only immutable table for enhanced performance. While this design choice improves the efficiency of read operations, it inherently prevents the insertion of new items into the database. Consequently, the immutable nature of the LevelDB table further contributes to the limitation of not supporting dynamic K-Means partitioning and quantization training on-device.

Since the training of new embeddings is not supported by default we continue our research by developing a custom 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 a dynamical environment. The method also explores the potential research areas to improve on for further work. The NPI approach enables us to evaluate the performance of the system under different conditions and analyze the impact of NPI on search performance. Here, 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 the NPI approach, 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 subspace using the pretrained codebook. Unfortunately when adding the embedding to the closest partition we have tothe LevelDB table is immutable since data is directly being loaded through mmap. Therefore the creation of a new LevelDB index means overwriting the existing index file which is stored in the metadata of the model. This overwrite then requires the whole index to be loaded into RAM and written to disk. This process is computationally expensive and time-consuming, making it impractical for real-time insertion of

new embeddings. To address this limitation, we also developed a Batch Insert function that allows for the insertion of multiple embeddings without reloading the entire index. This function is designed to improve the efficiency of the system by reducing the computational load and time required for inserting new embeddings. The Batch Insert function is implemented as a separate module that can be called from the main application, enabling the system to handle multiple insertions simultaneously. By batching the insertions, the system can efficiently manage the addition of new embeddings without overloading the memory or disk resources. This approach enhances the scalability and performance of the system, ensuring that it can handle a large number of insertions while maintaining high accuracy and efficiency.

The whole process incorporate a new YouTube songs:

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) **Reload the index**: The nmapped file is overwritten with the new index and partition data.

The process of adding a new YouTube song is also described in figure 4. When the batch insert function is used, the process does not directly create and reload this new index, instead it temporarly stores all the newly added embeddings after which the batch save function is called. The batch save function then creates a new index and reloads the index file when newly added items needs to be searchable.

D. Gossip Network Protocol

BeyondFederated is built on top of the open source Tribler SuperApp [1]. This software features a wide range of integrated functionality such as recommendation, trust to identity [15, 5, 10]. The Tribler superapp enables robust, decentralised interactions through the IPV8 protocol, which facilitates secure data exchange.

In our decentralised 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, ensuring 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 decentralised architecture.

[1]https://github.com/Tribler/trustchain-superapp

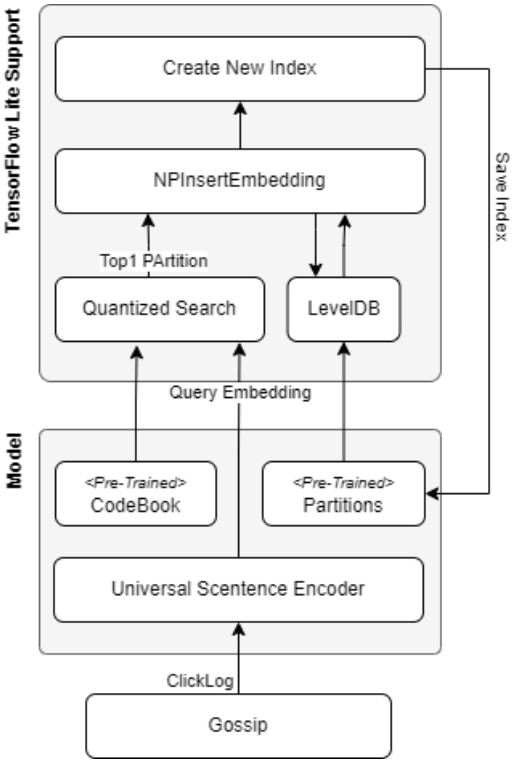


Fig. 4: An overview of inserting a new items into the model without retraining the entire dataset.

BeyondFederated Content discovery is based on the gossip protocol, each time a user clicks on a search result, a clicklog entry is created and initially stored using our non-perfect insert method. The ClickLog currently contains the following data of the clicked YouTube video: Query, Title, Author, YouTube video ID such that no personal data is shared. 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.

The clicklog data is gossiped around the network, enabling the system to continue sharing information even if some nodes go offline. The gossiping mechanism ensures that collective knowledge is maintained and disseminated across the network, supporting continuous learning and adaptation of the model. The standard implementations of Gossip protocols assume that nodes are non-malicious. This reliance on the integrity of nodes poses significant challenges in a decentralized environment where malicious nodes may exist. This is considered a disadvantage of the standard Gossip implementation, further research is needed to prevent continuous learning from malicious nodes.

The primary advantages of using gossip protocols for disseminating YouTube ClickLog data include scalability, robustness, and simplicity. The system can efficiently manage communication in networks with a large number of nodes, as

each node only interacts with a few other nodes at a time, thus reducing overall communication overhead. The decentralised nature of gossip protocols makes them resilient to node failures and network partitions, ensuring that the dissemination of ClickLog data continues smoothly even under adverse conditions. Additionally, the straightforward implementation and minimal overhead of gossip protocols make them easy to manage and integrate into existing systems.

Despite this challenge, gossip protocols are a valuable tool for ensuring efficient and resilient dissemination of YouTube ClickLog data in distributed systems. By balancing communication overhead, speed of information dissemination, and robustness, gossip protocols can significantly enhance the performance and reliability of BeyondFederated that rely on timely and accurate ClickLog data.

IV. Experiments and Evaluation

In this section, we present the experiments and evaluations conducted to assess the system's potential as the first decentralised 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[13] 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[22], 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 decentralised 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[7]. 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:

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

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 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 BeyondFederated. 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.

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.

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.

1) Results: The search query results for the band "Red Hot Chili Peppers" in the Spotify YouTube Trained Model are detailed in Table III. 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 decentralised 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 shown in table IV. 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 5].

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."

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

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

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

2) Discussion: The results highlight several key observations about the performance and behavior of the USE and 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.

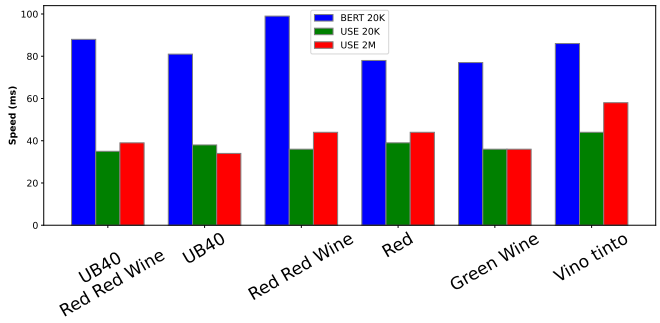


Fig. 5: Measuring the search time for different queries using the Universal Sentence Encoder and BERT models.

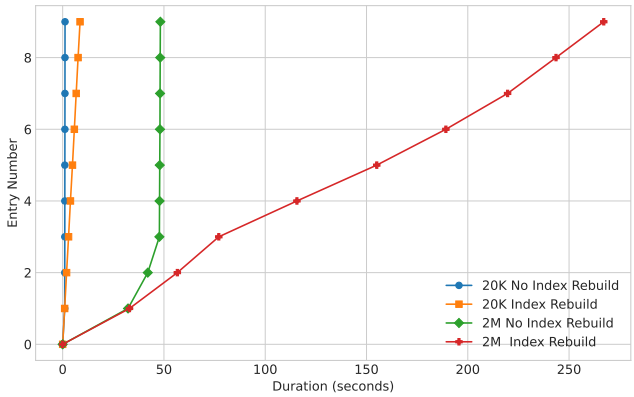
E. Non-Perfect Insert Experiment

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 YouTube item insertion, evaluate the system’s performance under a moderate load by inserting a batch of 10,000 items, and assess the scalability and performance for repeated insertions up to one million times. When we add a new song, we will be using a randomly generated list of random words for artist names and titles, including two random English words such as *”artist: Eternal Ballad title: Crystal Phoenix YouYubeId: 123”*.

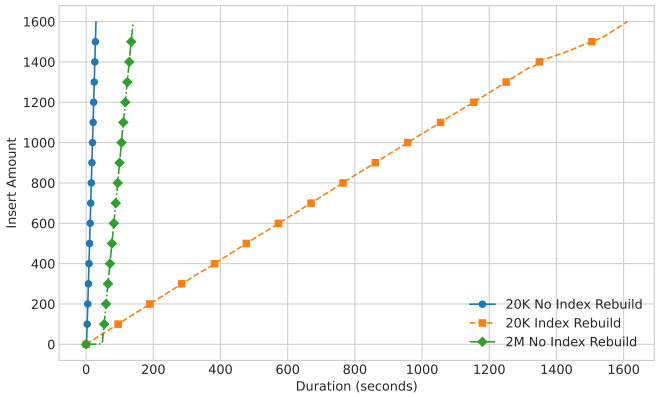
1) Results: The system successfully inserted a new item into our YouTube Spotify dataset, resulting in a model size increase of 876 bytes. When the new model is queried, including the same embedding as the one we just added, we notably receive our added YouTube item as the first result. However, we noticed the extremely close distance of -1.46156, which is uncommon compared to learned embeddings during the pre-trained phase. Furthermore, 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. In Figure 6 the different insertion times are shown for the different model sizes and the cost of rebuilding the index each time.

Inserting the same YouTube item into the larger pre-trained model, we observed an increase in both insertion time and RAM usage compared to our smaller model. This bottleneck is due to the need to read, initialize, and overwrite the entire index, which consists of a large size of 458 megabytes (MB). After writing the new index, it is necessary to reload the index and initialize a new LevelDB table to enable searching. The insertion process took 1.5 seconds, with a RAM usage of 458 MB. Despite the increased time and resource usage, the system successfully added the new item to the dataset.

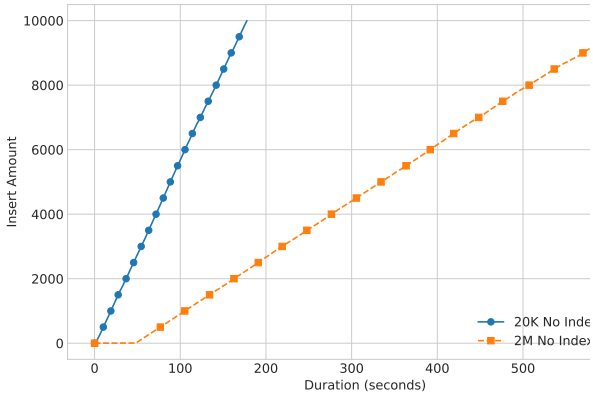
Inserting approximately 8 new songs results in significant differences, where the YouTube Commons 2M Dataset requires an infeasible amount of time to rebuild the index for each new entry. Since it is not necessary to search all insertions directly, we also experimented with adding entries first and rebuilding 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



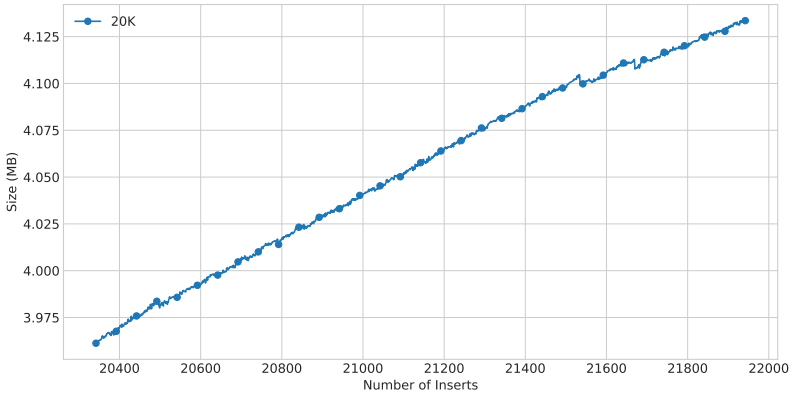
(a) Insert of 8 random Youtube items



(b) Insert of 1600 random Youtube items



(c) Insert of 10K random Youtube items



(d) Index size growth on 10K random Youtube Items

Fig. 6: Non-perfect item insert experiment.

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 each time on insertion 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 comparing more centroids to find the closest partition. 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.

Finally, adding 1 million YouTube items to the system, we observed a total insertion time of 7 hours, and the index size increased by 62 MB. Queries to the system were still processed really fast differencing from the original 37ms to 90ms. The difference dependents on the total number of items in the partition. Hashing the items in the partition and then comparing the distance to the centroid of the partition still performs extremely fast but the more items in the partition the slower the search time will be. In terms of recall, adding more songs to the system will quickly decrease the recall rate since all the newly added embeddings suffer from the codebook and the k-mean centroids not being updated, meaning their distance is placed closer to the centroid of the cluster they are placed in, overflowing the partition. None of the earlier queried songs are found in the top 10 results, and the recall rate drops to 0%. This is shown in tableV. This is a critical issue that needs to be addressed to maintain the system's effectiveness in retrieving relevant and accurate entries. The system's ability to learn and retrieve newly added YouTube embeddings, even in a non-optimized state, demonstrates its potential for handling large-scale data. However, facing the challenge of decreasing recall rates, the system must be periodically retrained to recalibrate the retrieval process and balance the influence of newly inserted songs with existing entries.

2) Discussion: The results of the non-perfect insert experiment provide valuable insights into the system's scalability and efficiency in handling new entries. The system demonstrates the ability to efficiently insert new items, even when they do not perfectly match the existing dataset. By managing the insertion process strategically, the system can optimize resource usage and reduce the overall time required for bulk insertions. The batch processing approach, which delays the index rebuild until after all insertions, proves to be an effective strategy for managing large-scale data. This method allows the system to handle a high volume of insertions while maintaining performance and efficiency. However, the system faces challenges in maintaining recall rates as the dataset grows, highlighting the need for periodic retraining to ensure the accuracy and relevance of search results. By addressing these challenges and optimizing the insertion process, the system can enhance its scalability and performance, enabling it to handle large-scale data effectively and efficiently.

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 decentralised search. Content discovery is based on a gossip protocol, as illustrated by the network view available. We focus on the Creative Commons Spotify-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 helps us determine the volume of data exchanged and assess the feasibility of implementing BeyondFederated Learning. Understanding the data requirements is crucial for ensuring the efficiency and scalability of the decentralised search process. By quantifying the data exchanged during content discovery and search, we can better understand the implications for network load and system performance.

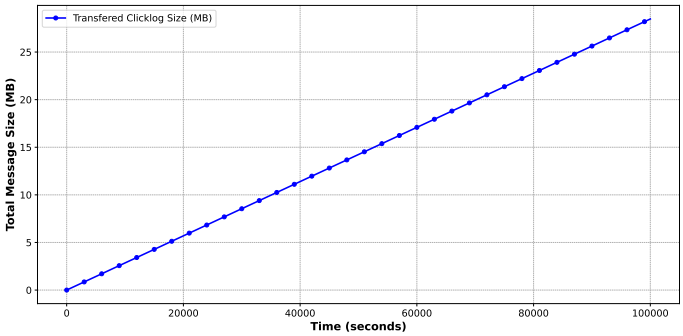


Fig. 7: Two random peers exchanging clicklog data

1) Results: The results of the content discovery and search experiment provide valuable insights into the data exchange process between peers. The data exchanged between the two devices is minimal, with each ClickLog containing only a fesingle YouTube item. This low volume of data ensures that the gossip protocol is efficient and does not overload the network with unnecessary information. Figure 7 show the steady increase of exchanged ClickLogs between the two devices, highlighting the continuous flow of information in a decentralised environment. This experiment demonstrates the feasibility of implementing content discovery and search using a gossip protocol, with minimal data exchange and efficient communication between peers. The results indicate that the system can effectively handle the exchange of search results and ClickLogs, enabling decentralised content discovery and search in a scalable and efficient manner.

V. Conclusion

In this research is shown that machine learning can be fully decentralised. Our on-device federated learning realisation is one of the first to explore the *share nothing* architecture [23]. Our open source prototype is capable of operating on any supported smartphone [2].

[2]https://github.com/Tribler/tribler/issues/7254

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 + Random Youtube Entries.

Query	Youtube ID	Artist name	Song title
Red Red Wine	zXt56MB-3vc	UB40	Red Red Wine
Red Hot Chili Peppers	8DyziWtkfBw	Red Hot Chili Peppers	Can't Stop
Red Hot Chili Peppers Otherside	rn_YodiJO6k	Red Hot Chili Peppers	Otherside

TABLE VI: ClickLog format example which is exchanged between smartphones using our IPv8 gossip overlay.

The results demonstrate that our *BeyondFederated* architecture using SCaNN can efficiently retrieve approximate nearest neighbors in a decentralised environment. This system effectively clusters YouTube embeddings using k-means clustering, which enables accurate real-time 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 decentralised 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 decentralised YouTube search system. This research highlights the potential for creating a decentralised web3 YouTube platform that is both efficient and privacy-preserving, paving the way for more robust and user-centric decentralised applications.

REFERENCES

[1] *Adding metadata to TensorFlow Lite models*. URL: https : / / www . tensorflow . org / lite / models / convert / metadata.

[2] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. *Approximate Nearest Neighbor Search in High Dimensions*. 2018. arXiv: 1806.09823 [cs.DS].

[3] *AndroidYoutubePlayer*. URL: https : / / github . com / PierfrancescoSoffritti/android-youtube-player.

[4] *ANN-Benchmarks is a benchmarking environment for approximate nearest neighbor algorithms search*. URL: https://ann-benchmarks.com/.

[5] Joost Bambacht and Johan Pouwelse. *Web3: A Decentralized Societal Infrastructure for Identity, Trust, Money, and Data*. 2022. arXiv: 2203.00398 [cs.DC].

[6] Daniel Cer et al. *Universal Sentence Encoder*. 2018. arXiv: 1803.11175 [cs.CL].

[7] *Custom trained Bert Scentence Encoder model*. URL: https : / / github . com / tensorflow / tflite - support / blob / master / tensorflow_lite_support / examples / colab / on_device_text_to_image_search_tflite.ipynb.

[8] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[9] Ruiqi Guo et al. *Accelerating Large-Scale Inference with Anisotropic Vector Quantization*. 2020. arXiv: 1908.10396 [cs.LG].

[10] J. S. Hammudoglu et al. *Portable Trust: biometric-based authentication and blockchain storage for self-sovereign identity systems*. 2017. arXiv: 1706.03744 [cs.CR].

[11] Jialiang Han et al. *DeFL: Decentralized Weight Aggregation for Cross-silo Federated Learning*. 2022. arXiv: 2208.00848 [cs.LG].

[12] *Introducing ScaNN vector indexing in AlloyDB, bringing 12 years of Google research to speed up vector search*. 2024. URL: https : / / cloud . google . com / blog / products/databases/understanding-the-scann-index-in-alloydb.

[13] *KaggleResources*. URL: https://www.kaggle.com/docs/notebooks#the-notebooks-environment.

[14] *LevelDB is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.* URL: https://github.com/google/leveldb.

[15] Rohan Madhwal and Johan Pouwelse. *Web3Recommend: Decentralised recommendations*

with trust and relevance. 2023. arXiv: 2307.01411 [cs.DC].

[16] Enrique Tomás Martínez Beltrán et al. "Decentralized Federated Learning: Fundamentals, State of the Art, Frameworks, Trends, and Challenges". In: *IEEE Communications Surveys & Tutorials* 25.4 (2023). ISSN: 2373-745X. DOI: 10.1109/comst.2023.3315746. URL: http://dx.doi.org/10.1109/COMST.2023.3315746.

[17] H. Brendan McMahan et al. *Communication-Efficient Learning of Deep Networks from Decentralized Data*. 2023. arXiv: 1602.05629 [cs.LG].

[18] *mmap*. URL: https://man7.org/linux/man-pages/man2/mmap.2.html.

[19] Facebook AI Research. *FAISS: A library for efficient similarity search and clustering of dense vectors*. https://faiss.ai/. Accessed: 2024-05-29. 2024.

[20] Shaped.ai. *The Secret Sauce of TikTok's Recommendations*. https://www.shaped.ai/blog/the-secret-sauce-of-tik-toks-recommendations. Accessed: 2024-05-29. 2024.

[21] Spotify. *Annoy: Approximate Nearest Neighbors in C++/Python optimized for memory usage and loading/saving to disk*. https://github.com/spotify/annoy. Accessed: 2024-05-29. 2024.

[22] *spotify-youtube-dataset*. URL: https://www.kaggle.com/datasets/salvatorerastelli/spotify-and-youtube.

[23] Michael Stonebraker. "The Case for Shared Nothing Architecture". In: *Database Engineering* 9.1 (1986). http://db.cs.berkeley.edu/papers/hpts85-nothing.pdf.

[24] Philip Sun. *Announcing ScaNN: Efficient Vector Similarity Search*. URL: https://research.google/blog/announcing-scann-efficient-vector-similarity-search/.

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems*. 2014, pp. 3104–3112. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/310ce61c90f3a46e340ee8257bc70e93-Paper.pdf.

[26] *TensorFlow Lite Support*. URL: https://github.com/tensorflow/tflite-support/tree/master/tensorflow_lite_support.

[27] Martijn de Vos et al. *Decentralized Learning Made Practical with Client Sampling*. 2024. arXiv: 2302.13837 [cs.DC].

[28] Yuxin Wen et al. *Fishing for User Data in Large-Batch Federated Learning via Gradient Magnification*. 2022. arXiv: 2202.00580 [cs.LG].

[29] Liangqi Yuan et al. *Decentralized Federated Learning: A Survey and Perspective*. 2024. arXiv: 2306.01603 [cs.LG].