# Beyond Federated Learning decentralised learning on the edge

Quinten van Eijs Delft University of Technology The Netherlands Johan Pouwelse Delft University of Technology The Netherlands

***Abstract*—**

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

## I. Introduction

In recent years, the power of Artificial Intelligence (AI) has been increasingly leveraged by search and recommender systems to enhance user experiences and drive engagement. These systems use AI to analyze vast amounts of data, uncover patterns, and make personalized content suggestions to users. This capability is particularly crucial for content-rich platforms, such as social media and streaming services, where users are continuously exposed to new and diverse content. AI-driven search and recommender systems can deliver highly relevant and timely recommendations, significantly improving user satisfaction and retention.
Training AI models for these search and recommender systems involves processing large datasets to identify and predict user preferences accurately. Traditional approaches often rely on centralized models, where data is collected and processed in a central location. While this method can be effective, it introduces several challenges, including scalability issues, latency, and significant privacy concerns. Centralized models require aggregating vast amounts of user data, raising the risk of data breaches and unauthorized data usage.

Privacy concerns have become a major issue in centralized AI systems. Users are increasingly wary of data breaches and unauthorized data usage. Centralized models typically aggregate vast amounts of user data, raising significant privacy and security risks. Federated learning has emerged as a solution to some of these issues by enabling the training of AI models across multiple devices while keeping data localized. This approach enhances privacy, as raw data never leaves the local devices, and only aggregated model updates are shared. However, federated learning still relies on a central server for aggregating these updates, which introduces bottlenecks and potential points of failure.

Decentralized systems offer a promising alternative by ensuring that data is processed locally and only aggregated insights are shared. This approach not only enhances user privacy but also complies with stringent data protection regulations, such as the General Data Protection Regulation (GDPR). Decentralizing AI systems can overcome the limitations of both centralized and federated models by distributing computational tasks and data storage across multiple nodes, thereby enhancing scalability, reducing latency, and eliminating single points of failure.

This research is driven by the significant advancements in the computational capabilities of mobile devices, which now rival traditional computing systems, and the promising potential breakthroughs in distributed learning methodologies. These developments open up new possibilities for creating more efficient, scalable, and privacy-preserving AI systems that can operate seamlessly in a decentralized environment.

To enhance the effectiveness of the search and recommendation system, this research employs advanced techniques such as vector search, k-means clustering, and approximate nearest neighbor (ANN) search. Vector search allows for efficient retrieval of similar items in high-dimensional space, essential for content recommendation. K-means clustering helps in organizing content and user behaviors into meaningful clusters, facilitating targeted recommendations. ANN search techniques enable scalable and real-time retrieval of nearest neighbors, critical for maintaining the performance of the system under dynamic conditions.

Decentralizing these techniques presents several interesting challenges and opportunities. By distributing the computation and data storage across a P2P network, the system can achieve greater scalability and fault tolerance. Local processing of data can significantly enhance user privacy, as sensitive information does not need to be transmitted to centralized servers. Additionally, decentralization can reduce latency and improve the responsiveness of the system by leveraging the computational resources available at the network's edge.

## II. Problem Description

The significance of this problem lies in the absence of high-performing decentralized search and recommendation AI systems. Currently, the most widely used content-sharing platforms, such as YouTube, TikTok, and Instagram Reels, rely on centralized architectures for their recommendation engines. These centralized systems, while efficient in controlled environments, face significant challenges when it comes to scalability, privacy, and real-time data processing, especially as the volume of content and user interactions continues to grow exponentially.

The rise of frameworks like TensorFlow Lite has enabled machine learning capabilities on local mobile devices, allowing for more personalized and efficient processing without the need for constant communication with central servers. This advancement is crucial for enhancing user experiences through faster, on-device computations and reduced latency. However, despite these advancements, there has been a lack of exploration into leveraging these capabilities within a distributed or decentralized setting. This gap highlights the need for research into decentralized AI systems that can harness the power of local learning while also facilitating robust, scalable, and privacy-preserving recommendation systems.

## III. Design and Implementation

In this section, we present a comprehensive overview of the Beyond Federated system architecture. We will begin with a high-level description of the overall architecture, highlighting the key components and their interactions. Following this, we will delve into detailed descriptions of each component, explaining their roles, functionalities, and how they collectively contribute to creating a decentralized search engine built on top of the Tribler SuperApp.

**App Screenshots x2**

### A. High-Level Overview of the Architecture

Beyond Federated represents an initial, modest step toward creating the first decentralized search engine. The system is built on top of the Tribler SuperApp and its architecture consists of four distinct components. These components work together to provide a decentralized, secure, and efficient search experience. The key components are:

- **User Interface**: Facilitates user searches and data entry.
- **IPV8 Protocol**: Leverages decentralized peer-to-peer networking for secure communication.
- **Search and Embedding Model**: Converts queries into embeddings and retrieves similar items.
- **Beyond Federated TensorFlow Lite Support**: Provides APIs for model interaction and customization of the model.

### B. System Model and Assumptions

### C. User Interface

This component allows users to search for content, returning the top results most relevant to the query. The interface is designed to be simple and practical, enabling efficient and intuitive searches. Content is retrieved as metadata and can include various types of data such as compressed files or references to file locations. Our specific focus is on YouTube music and video search; therefore, our metadata consists of the Title, Author, and YouTube video ID. The interface also allows users to insert new entries into the system by providing the Title, Artist, and YouTube video ID for new YouTube items.

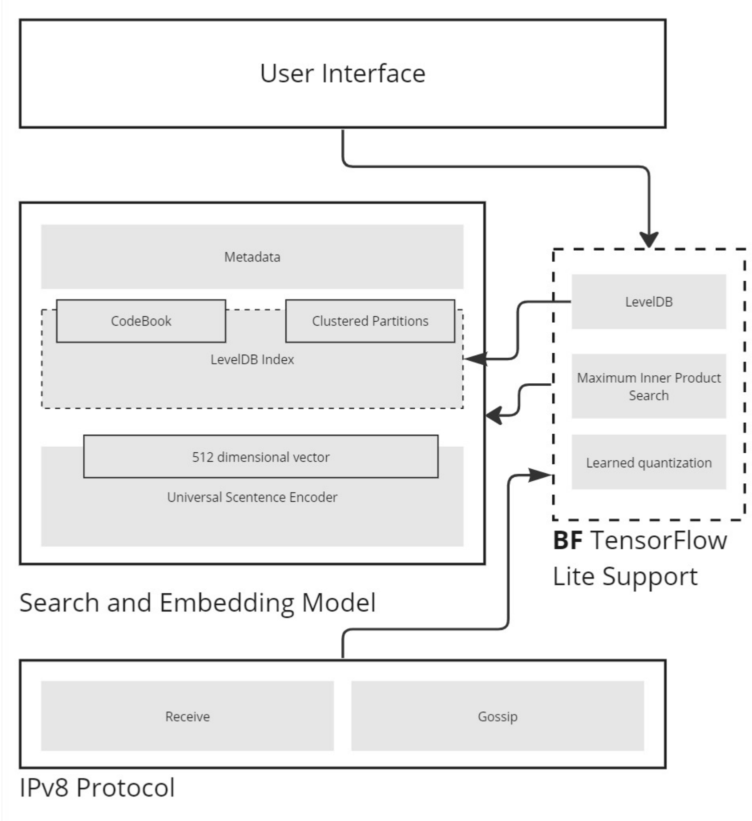

Fig. 1. The PeerAI system model. The system consists of three components: the P2P network, the model, and the collaborative learning algorithm. The P2P network is responsible for sharing the data, the model is responsible for encoding the data into a vector space, and the collaborative learning algorithm is responsible for updating the embedding model based on the data received from the P2P network.

### D. IPV8 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.

The clicklog data is gossiped around the network using a gossip protocol. Gossip protocols are decentralized communication methods where nodes periodically share information with a subset of other nodes. This approach ensures that data is disseminated efficiently across the network without requiring a central coordinator.

Each time a user clicks on a search result, the clicklog entry is created and initially stored on the local node. 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.

This decentralized distribution method enhances the resilience and fault tolerance of the network. Since the data is not reliant on a single central server, the system can continue to function and share information even if some nodes go offline.

The clicklog is an essential component for understanding user interactions and improving the relevance of search results. Each entry in the clicklog consists of three key attributes: Title, Author, and YouTube video ID.

### *E. Search and Embedding Model*

The third component is the search model, which converts input queries into embeddings and searches for the most similar items within the dataset. This model ensures that the search results are highly relevant to the user's query, providing accurate and efficient retrieval of information.

To make Beyond Federated capture semantics in our search terms and content, we utilize the Universal Sentence Encoder (USE) to transform text into high-dimensional vectors suitable for various natural language processing tasks. The USE is trained and optimized for processing text longer than individual words, such as sentences and phrases. The model accepts variable-length English text as input and produces a fixed-length 512-dimensional vector as output. The USE is built upon a Deep Averaging Network (DAN) encoder architecture, training on tasks that necessitate understanding the meaning of word sequences rather than individual words.

In addition to the USE, we employ the ScaNN algorithm for similarity search in 512-vector dimension space. ScaNN is based on Product Quantization (PQ) and Asymmetric Hashing, compressing database embeddings into a compact form for fast retrieval resulting in state-of-the-art performance in similarity search tasks. Training a codebook for 512-dimensional vectors in ScaNN involves subdividing the vectors 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. This process enables efficient approximate nearest neighbor searches by reducing the computational complexity and leveraging the compact quantized representations. Next all the quantized representations are then clustered also using K-mean to form partitions of the dataset. For optimal performance the number of partitions is suggested by the square root of the dataset size.

*Assumption 1: The Universal Sentence Encoder significantly enhances the decentralized search model's capability to handle fuzzy search queries by providing semantically rich embeddings that ScaNN can efficiently process and search within a large and diverse dataset.*

Despite the ”S” in ScaNN standing for scalable, ScaNN does not dynamically support the insertion of new items into the index after initial training. This limitation arises because the process of learning the codebook and forming partitions is computationally expensive, making it infeasible to run on edge devices or in real-time scenarios. To address this, we developed 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 imbalanced tree structure within the dataset. 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.

*Assumption 2: Although the NPI method allows for the practical insertion of new data without the computational expense of re-learning the entire codebook and forming new partitions, the resulting search performance will be somewhat lower. This is because the new data points are not perfectly integrated into the index structure, potentially leading to less optimal clustering and retrieval accuracy.*

### *F. Beyond Federated TensorFlow Lite Support*

The final component consists of TensorFlow Lite APIs that enable users to interact with the search model. This component allows for customization of model inference and modification of the model's metadata, including all YouTube entries. The TensorFlow Lite Support library is written in C++ and uses Bazel for cross-platform building. It is supported on Java, C++, and Swift. The library provides APIs for loading and running TensorFlow Lite models on edge devices, enabling efficient and personalized processing without the need for constant communication with central servers.

## IV. Experiments and Evaluation

In this section, we will 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. Datasets and Pre-trained Models*

The datasets for this experiment have been selected based on the specific use case of focusing on audio and video content. Consequently, the embeddings consist of the artist's name and the title of the video or audio content. Additionally, the metadata includes a YouTube video ID to enable retrieval and display of the actual video within our user interface.

To highlight the diversity of the metadata, we began with the PandaCD dataset, which contains 700 songs featuring legally distributed music under Creative Commons licenses and artist permissions. Although the videos are accessed through YouTube, indicating that the system is not fully decentralized due to the necessity of sending requests to Google's central authority, this setup demonstrates that the metadata does not have to be limited to a specific domain. To explore full decentralization, we included a dataset that consists of magnet links.

The second dataset, sourced from Kaggle, is the Spotify and YouTube dataset, released under the CC0: Public Domain license. This dataset includes 20,230 songs from 2,079 artists. Before utilizing 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 model remains unaffected by these extraneous details.

Our final 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

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** ($\sigma$): 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** ($\omega$): 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 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.

## *B. Experiment I — Pretrained Model Search and Insert*

In this experiment, we aim to test the system's ability to perform searches and insert new items. The experiment will utilize all the pretrained models. The testing environment consists of a single node with a single peer running inside an Android emulator, specifically the PIXEL API LEVEL 34. The host machine is equipped with an 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz processor and 16GB of RAM.

*1) Metrics:* A common metric for testing the accurcy of the search is the Recall@k metric. This metric measures the percentage of relevant items that are retrieved in the top-k search results. The formula for Recall@k is as follows:

$$Recall@k = \frac{|\{relevant\} \cap \{retrieved\}|}{|\{relevant\}|} \tag{1}$$

where $\{relevant\}$ is the set of relevant items and $\{retrieved\}$ is the set of items retrieved by the search algorithm. The Recall@k metric is used to evaluate the search accuracy of the system. In addition to measuring recall, we will also evaluate the system's performance in terms of the time it takes to insert and search for items. This will provide a comprehensive assessment of the system's efficiency and effectiveness. Tensorflow is capable of running CPU and GPU operations, but we will focus on CPU performance for this experiment.

*2) Searching results:* In this section, we present the results of evaluating multiple queries containing parts of the original embeddings learned by the model. An immediate observation is that all queries take roughly the same amount of processing time, even though the partitions in the 2M model are much larger. This consistency in query processing time is attributed to the parallel nature of the search and its logarithmic time complexity. On average, all queries return the top 8 results in approximately 500ms, as shown in Table **??**.

When querying using the full embedding as stored in the model, the results consistently include the correct items within the top 3, yielding a recall of 100

Additionally, when we examine the query results for the band "UB40," we observe that while the correct song is returned, the band has multiple songs in the dataset. Therefore, it is logical that the results would include more of the songs by UB40, demonstrating the model's ability to handle queries for artists with multiple entries in the dataset.

Another notable finding is the model's semantic understanding of the queries. For instance, querying "green wine" successfully returns the correct song, showcasing the model's capability to comprehend and process the semantic context of the query accurately.

However, we also observed that queries in languages other than English are less effective, with a recall@k score of 0

These findings highlight the efficiency of the system in handling large datasets and its robustness in returning relevant results even with partial queries. The consistency in processing time across different query types and dataset sizes underscores

| Embedding | | Metadata |
|---|---|---|
| $\{artist\}$ | $\{title\}$ | $JSON\{artist : \{artist\}, title : \{title\}, id : \{youtubeID\}\}$ |

TABLE I
OVERVIEW OF DIFFERENT PRE-TRAINED MODELS AND THEIR TRAINED DATASET.

| Dataset | Model Params | Entries | Model Size | Dataset Size |
|---|---|---|---|---|
| PandaCD | $\sigma$ = 30, $\omega$ = 6 | 700 | 500KB | 657KB |
| Spotify and Youtube | $\sigma$ = 140, $\omega$ = 6 | 20,230 | 4MB | 30.78MB |
| YouTube-Commons | $\sigma$ = 1450, $\omega$ = 3 | 2,063,066 | 458MB | 427.27MB |

TABLE II
OVERVIEW OF DIFFERENT PRE-TRAINED MODELS AND THEIR TRAINED DATASET.

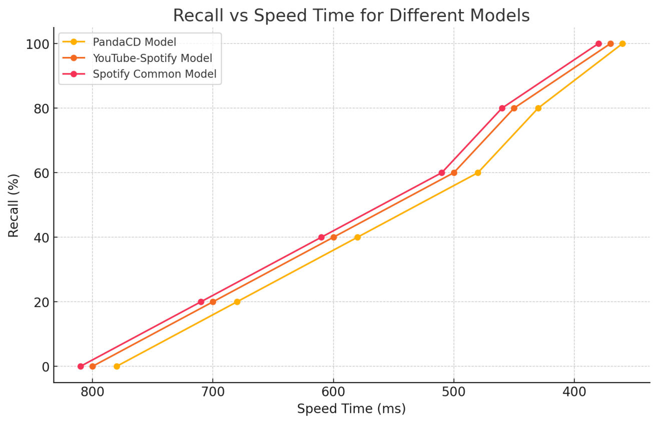

Fig. 2.

the effectiveness of the parallel search mechanism employed by the model.

## C. Experiment II — Partition Overflow

In this experiment, we aim to test the system's ability to handle partition overflow scenarios. The experiment will utilize the Spotify and YouTube dataset, which contains 20,230 songs from 2,079 artists. Our primary interest is to observe when the system's performance begins to degrade as the number of inserted items increases, potentially leading to partition overflow.

The testing environment consists of a single node with a single peer running inside an Android emulator, specifically the PIXEL API LEVEL 34. The host machine is equipped with an 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz processor and 16GB of RAM. This setup provides a controlled environment to accurately measure the system's performance under varying conditions of data load and insertion rates.

We will incrementally insert new items into the dataset and monitor the system's response time and accuracy. By analyzing these metrics, we aim to identify the threshold at which partition overflow occurs and evaluate the impact on the system's overall efficiency and search accuracy. This experiment will help in understanding the robustness of the system in real-world scenarios where data continuously grows and new items are frequently added.

*1) Metrics:* The metrics during the partition overflow experiment will primarily focus on CPU performance and the time required to search for items after numerous inserts. This will provide a comprehensive assessment of the system's efficiency and effectiveness under increased data load. Although TensorFlow is capable of running both CPU and GPU operations, we will focus on CPU performance for this experiment to maintain consistency and accurately gauge the impact of partition overflow on the system.

*2) Results:* To test the effects of overloading a single partition in the model, we repeatedly inserted the same embedding until the partition contained 1,000,000 identical items. Since the LevelDB storage overwrites the same key that already exists, the model size was not affected.

As a result of this overload, the search time increased significantly to around 1300 ms, compared to the 500 ms observed before the overload. Despite this increase, the system still performs reasonably fast and continues to handle a large volume of data effectively. Importantly, the rest of the queries were not affected by this overload, as only the partition with the repeated embedding experienced the slowdown.

For this specific overloaded embedding, recall reached 100

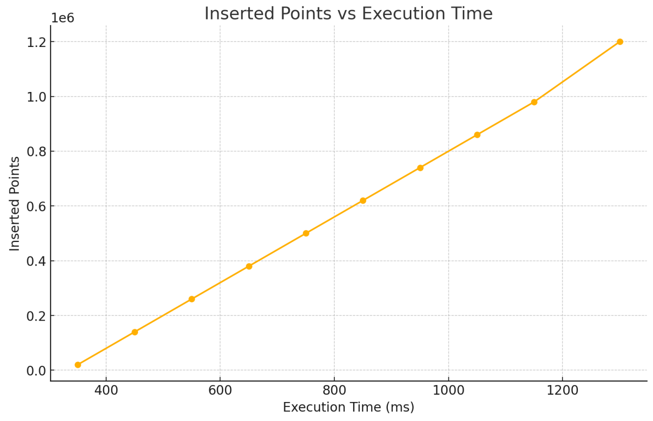

Fig. 3.

To conduct this analysis, we repeated our previous experiment but this time inserted 1,000,000 different items into their respective closest partitions. This deliberate overloading aimed to assess how the system handles significant data influxes across multiple partitions.

The results showed that the search time increased to 2300 ms, which, while higher than previous measurements, is still relatively fast given the volume of data. This demonstrates the system's robustness and ability to manage large datasets effectively.

For single word queries, the recall remained at 100

To ensure a thorough evaluation, we used a text file containing 1,000,000 random sentences to insert new items into the model. This diverse set of entries provided a rigorous test of the system's capacity to handle extensive and varied data inputs. However, unlike our previous single partition overflow experiment, we observed a notable decrease in recall scores when multiple partitions were overloaded. This indicates that the system's accuracy is compromised when several partitions are simultaneously burdened with excessive data.

For single word queries, the recall was notably lower than 100

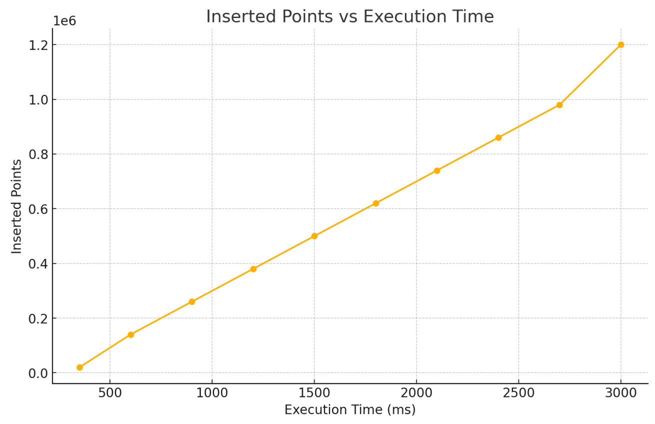

Fig. 4.

### D. Experiment III — Network Level

This network experiment aims to investigate the efficiency, latency, and scalability of communication protocols within a decentralized system by focusing on the distribution of clicklog data. The experiment involves continuously inserting random n clicklog entries per second into the network and recording the resulting traffic. The primary objectives are to measure the network's ability to handle high-throughput data insertion and to monitor how traffic evolves over time.

To achieve this, we will simulate a scenario where clicklog data is generated at a constant rate, with random entries being inserted into the system continuously. Throughout the experiment, we will meticulously record the network traffic, capturing both the total amount of data transferred (in megabytes) and the incremental increases over time.

The collected data will then be plotted to visualize the total traffic volume and its growth over the duration of the experiment. This visualization will help us understand the network's capacity to handle continuous data streams and identify any potential bottlenecks or inefficiencies in the communication protocols.

By analyzing the traffic patterns and latency, we aim to gain insights into the performance of the network under high-load conditions. This will contribute to developing more efficient and scalable communication protocols for decentralized systems, ensuring they can effectively manage large volumes of distributed data in real-time.

### E. Metrics

The primary metrics for this experiment will focus on network performance, including throughput, latency, and scalability. By measuring the total amount of data transferred and the rate of data insertion, we can assess the network's capacity to handle high-throughput data streams. Additionally, monitoring the latency of data transmission will provide insights into the system's responsiveness and efficiency in processing incoming data.

### F. Results

## V. Conclusion

The results show that the Beyond federated architecture can effectively support decentralized k-means clustering, enabling nodes to collaboratively learn and refine models without a central coordinator. Now tackle assumptions based on results from the experiments.

### 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

| CPU Speed | Searched Partition size | Rank | Distance | Metadata |
|---|---|---|---|---|
| 434.396 ms | P1= 9 | 1 | -0.52592 | artist: Cullah, title: Firebird, magnet: magnet:?xt=urn:..tr=.. |
| | P2= 28 | 2 | -0.42465 | artist: Hatemagick, title: Hellfire EBM,magnet: magnet:?xt=urn:..tr=.. |
| | P3= 13 | 3 | -0.40696 | artist: Greendjohn, title: Loophole,magnet: magnet:?xt=urn:..tr= |
| | P4= 29 | | | |

TABLE III
RESULTS OF THE SEARCH QUERY FOR THE QUERY "FIREBIRD" IN THE PANDACD TRAINED MODEL.

| CPU Speed | Searched Partition size | Rank | Distance | Metadata |
|---|---|---|---|---|
| 477.54 ms | P1= 86 | 1 | -0.56265 | artist: UB40, title: Red Red Wine, url: zXt56MB-3vc |
| | P2= 108 | 2 | -0.51885 | artist: Cream, title: Strange Brew (1967), url: hftgytmgQgE |
| | P3= 139 | 3 | -0.51441 | artist: Owl City, title: Vanilla Twilight , url: pIz2K3ArrWk |
| | P4= 132 | 4 | -0.51441 | artist: Pouya, title: Death by Dishonor, url: otl8yjZcg2Y |
| | P5= 110 | 5 | -0.51220 | artist: Duster, title: Chocolate And Mint, url: t4NcmzFx2tk |
| | P6= 63 | 6 | -0.51164 | artist: Dominic Fike, title: Vampire, url: SUjD_nwooTg |
| | | 7 | -0.50942 | artist: Saweetie, title: Tap In , url: jUIrolORx6M |
| | | 8 | -0.49944 | artist: Night Lovell, title: Dark Light, url: HTp5PH8ot6Q |

TABLE IV
RESULTS OF THE SEARCH QUERY FOR THE QUERY "RED RED WINE UB40" IN THE SPOTIFY YOUTUBE TRAINED MODEL.

| CPU Speed | Searched Partition size | Rank | Distance | Metadata |
|---|---|---|---|---|
| 469.873 ms | P1= 190 | 1 | -0.71566 | artist: Red Hot Chili Peppers, title: Cant Stop, url: 8DyziWtkfBw |
| | P2= 92 | 2 | -0.70641 | artist: Red Hot Chili Peppers, title: Give It Away , url: Mr_uHJPUlO8 |
| | P3= 106 | 3 | -0.51441 | artist: Red Hot Chili Peppers, title: Otherside , url: rn_YodiJO6k |
| | P4= 149 | 4 | -0.65953 | artist: Red Hot Chili Peppers, title: Californication , url: YlUKcNNmywk |
| | P5= 127 | 5 | -0.64781 | artist: Red Hot Chili Peppers, title: Dark Necessities , url: Q0oIoR9mLwc |
| | P6= 129 | 6 | -0.64226 | artist: Red Hot Chili Peppers, title: Dani California , url: Sb5aq5HcS1A |
| | | 7 | -0.59723 | artist: Red Hot Chili Peppers, title: Snow (Hey Oh) , url: yuFI5KSPAt4 |
| | | 8 | -0.57441 | artist: Jonas Brothers, title: Marshmello x - Leave Before You Love Me , url: hmUyEDG7Jy0 |

TABLE V
RESULTS OF THE SEARCH QUERY FOR THE QUERY "RED HOT CHILI PEPPERS" IN THE SPOTIFY YOUTUBE TRAINED MODEL.

| CPU Speed | Searched Partition size | Rank | Distance | Metadata |
|---|---|---|---|---|
| 477.235 ms | P1= 1032 | 1 | -0.88953 | title: how to pronounce fallible, author: Proper English, id: 8ZxQ57QpoYk |
| | P2= 687 | 2 | -0.88628 | title: How to pronounce August Ames, author: Pronunciation Guide USA, id: foQaI7gbIuc |
| | P3= 13 | 3 | -0.87587 | title: How to pronounce larynx., author: Pronunciation Guide USA, id: rx5WeqUXeeU |
| | P4= 1533 | 4 | -0.86677 | title: How to pronounce jeopardize., author: Pronunciation Guide USA, id: DadHqPqJs1I |
| | | 5 | -0.85506 | title: how to pronounce fallers, author: Proper English, id: 8Z8WpaYn-vI |
| | | 6 | -0.85051 | title: How to pronounce Aaliyah Love, author: Pronunciation Guide USA, id: 5X10rDWRxlE |
| | | 7 | -0.84725 | title: How to pronounce Maxon, author: Pronunciation Guide USA, id: AAiUt_ADZRQ |
| | | 8 | -0.84075 | title: how to pronounce juxtaposition, collocation and suggestion, author: English Perfect Pronunciation |

TABLE VI
RESULTS OF THE SEARCH QUERY FOR THE QUERY "HOW TO PRONOUNCE AUGUST" IN THE YOUTUBE COMMON TRAINED MODEL.