# Aucher: Multi-modal Queries on Live Audio Streams in Real-time

Zeyi Wen[†1], Mingyu Liang[‡2], Bingsheng He[†3], Zexin Xia[‡4], Bo Li[§5]

[†]*National University of Singapore,* [§]*Hong Kong University of Science and Technology*
{[1]wenzy,[3]hebs}@comp.nus.edu.sg, [5]bliar@connect.ust.hk

[‡]*Shanghai Jiao Tong University, China*
{[2]liangmingyu,[4]xiazexin}@sjtu.edu.cn

*Abstract*—**This paper demonstrates a real-time search system called *Aucher* for live audio streams. Audio streaming services (e.g., Mixlr, Ximalaya, Lizhi and Facebook Live Audio) have become increasingly popular with the wide use of smart phones. Because of the popularity of audio broadcasting, the data volume of live audio streams is also ever increasing. Searching and indexing these audio streams is an important and challenging problem. Aucher is a system prototype which can support both voice search and keyword search on audio streams. We achieve the real-time response for queries by our novel index which exploits log structured merge-trees and supports multi-modal search. Moreover, our system can handle insertion about four times faster and more memory efficient than the state-of-the-art solution. We plan to demonstrate searching live audio streams by keywords and voice, illustrate the trade-off of freshness, popularity and relevance on query results, perform searching hot terms, and show the ability of searching live audio streams in real-time.**

## 1. Introduction

Audio streaming platforms (e.g., Ximalaya and Facebook Live Audio) have attracted a large number of users. More and more people are enjoying live audio broadcasting. Searching and indexing these audio streams is an important problem but is also challenging because: (i) queries on the large number of audio streams need to be answered in real-time; (ii) a live audio stream is inserted into the index continuously to enable live audio streams to appear in query results, and the number of insertions is large which often becomes a performance issue. Existing studies [1], [2] on audio search either oversimplify the problem (e.g., only consider the title of an audio stream) or simply ignore searching live audio streams. Moreover, they do not explore the multi-modal property (i.e., both sound and transcribed text) of audio streams.

In this demonstration paper, we leverage recent successful technologies from speech recognition and databases (e.g., log structured merge-tree indexing) to address the problem of the emerging application of live audio streaming. Our system called *Aucher* (short for Audio Searcher) is powered by a multi-modal and unified log structured merge-tree (LSM-tree) based index to support intensive insertions and real-time search on live audio stream applications. Aucher supports two major types of indexing techniques in audio search: text based indexing and sound based indexing. Hence, Aucher allows both keyword and voice search using the indices on the transcribed text and extracted sound features (e.g., MFCC [3]). Aucher has three key technical challenges: (i) a live audio stream may appear in multiple inverted indices due to the streaming nature; (ii) relevance, popularity and freshness of each audio stream need to be maintained in a way that allows fast accesses; (iii) massive insertions are happening alongside with queries. In Aucher, we exploit various techniques to address the technical challenges (more details presented in Section 3.2).

Aucher is built on top of our recent work which introduces RTSI [4] to support live audio stream indexing and search. In this demonstration paper, we make the following additional contributions: (i) we build a complete system prototype to demonstrate our techniques for live audio search, (ii) we develop techniques to search hot terms in a given time frame, and (iii) we further improve RTSI by combining two LSM-trees into one to reduce the memory consumption.

In the demonstration, we will interact with the audience to demonstrate the following three scenarios. First, we will show the audience searching live audio streams in real-time by keywords and by voice, and the audience can propose their queries. We will demonstrate the comparison with the state-of-the-art approach named LSII [5]. Our experimental results show that our index outperforms LSII by four times in insertions and is more memory efficient, while retaining similar query response time. Second, we like to show the audience searching hot terms in a given time frame, and together with the audience, we will try to identify emerging events at the time frame based on the hot terms. All the demonstration studies will be conducted in two real data sets (i.e., Ximalaya and VOA), both of which are from popular platforms for live audio streams. In the demo, users can also play with the system prototype with their own queries.

## 2. Background and related work

In this section, we first present some background on live audio streaming services, and audio indexing and search. Then we discuss the related work on indexing techniques for real-time search.

### 2.1. Live audio streaming and audio indexing

Live audio streaming services have become very popular due to the wide use of smart phones and better network connection. Some examples of live audio streaming services are Mixlr [6], Ximalaya [7], Lizhi [8] and Facebook Live Audio. Searching for relevant live audio streams in real-time is very important, because the listeners are interested in some real-time hot events and topics that are being emerged in the live audio streams. Existing audio search studies are based on one of the two mainstream methods: sound based search and text based search.

Nagano et al. [9] proposed a purely sound based method to retrieve audio streams. Their key idea is to compare the query voice with the audio streams using the similarity of sound signals. Other studies (e.g., [10]) propose to convert audio streams into phonetic lattices (i.e., sound units) and build indices on the lattices. These techniques are not designed for real-time live audio stream search, because they consider the set of audio streams static and no more audio streams are coming in after the index is built.

Audio streams can be represented using simple text such as titles and categories [11], and then the index is built on the text. The disadvantage is simple text is not informative to represent the audio streams, and hence many relevant audio streams are not retrieved when answering queries. Some studies (e.g., [12]) use full text (transcribed from audio) based approaches to audio search. Transcribed text becomes more popular due to advancement of speech recognition.

### 2.2. Indexing techniques for real-time search

There are several existing studies [13], [14] for supporting real-time search for microblogs in particular. The recent and more related work is *LSII* [5] which is built on top of LSM-trees for real-time search. These existing real-time search approaches discussed above are for indexing microblogs or short text where a microblog appears in only one inverted index.

In live audio stream indexing problems, data arrives in a long stream manner and an audio stream may appear in multiple inverted indices to enable live audio streams to appear in query results. The issue of an audio stream appearing in multiple inverted indices makes those existing studies not applicable to live audio indexing. Our recent work proposed RTSI [4] to support live audio stream indexing and search. In this demo paper, we further improve RTSI in order to reduce the memory consumption by combining two LSM-trees into one. We also develop techniques to search hot terms in a given time frame, and we build a complete system prototype to demonstrate our techniques.

## 3. The Aucher system

### 3.1. Overview of our system

We implement the system in a client-server architecture. Figure 1 shows the search interface of Aucher, which supports both keyword search and voice search. In the voice search, users say the query keywords as input similar to Google voice search. The system allows users to set the weights of freshness, significance and relevance via "Advanced Setting". The scoring function is shown below.

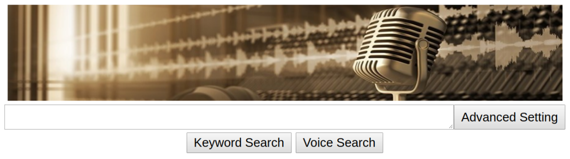
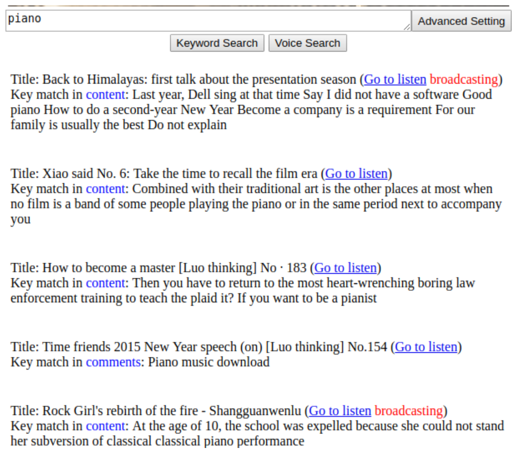


Figure 1. Search interface



piano

Advanced Setting

Keyword Search Voice Search

Title: Back to Himalayas: first talk about the presentation season (Go to listen broadcasting)
Key match in content: Last year, Dell sing at that time Say I did not have a software Good piano How to do a second-year New Year Become a company is a requirement For our family is usually the best Do not explain

Title: Xiao said No. 6: Take the time to recall the film era (Go to listen)
Key match in content: Combined with their traditional art is the other places at most when no film is a band of some people playing the piano or in the same period next to accompany you

Title: How to become a master [Luo thinking] No · 183 (Go to listen)
Key match in content: Then you have to return to the most heart-wrenching boring law enforcement training to teach the plaid it? If you want to be a pianist

Title: Time friends 2015 New Year speech (on) [Luo thinking] No.154 (Go to listen)
Key match in comments: Piano music download

Title: Rock Girl's rebirth of the fire - Shangguanwenlu (Go to listen broadcasting)
Key match in content: At the age of 10, the school was expelled because she could not stand her subversion of classical classical piano performance

Figure 2. Search result presentation

$$f(q,p) = \omega_p \cdot pop(p) + \omega_r \cdot rel(q,p) + \omega_f \cdot frsh(p) \quad (1)$$

where $q$ is a query and $p$ is an audio stream; $pop(p)$, $rel(q,p)$ and $frsh(p)$ are the scores of popularity, relevance and freshness, respectively; $\omega_p$, $\omega_r$ and $\omega_f$ are the weights of popularity, relevance and freshness, respectively.

After issuing a query, the query results appear right below the query box. Figure 2 shows the query results. From the figure, each result consists of a title, a snapshot and an external link called "Go to listen". Some of the results are highlighted with "broadcasting" which are the live audio streams, and the other are historical audio streams. Moreover, the snapshots are mainly from audio content and user's comments.

### 3.2. System internals

Figure 3 shows the key components of audio indexing and query answering in Aucher. The top part is the indexing process and the bottom part is the query answering process. When building the index, the live audio streams are converted into text by speech recognition (e.g., Google Speech Recognition) and also converted into phonetic lattices using techniques proposed in [10]. The phonetic lattices are then represented using MFCC. The index is built for text and phonetic lattices to support keyword and voice search.

**Freshness, significance and relevance**: The users of audio streaming services tend to be interested in more recent

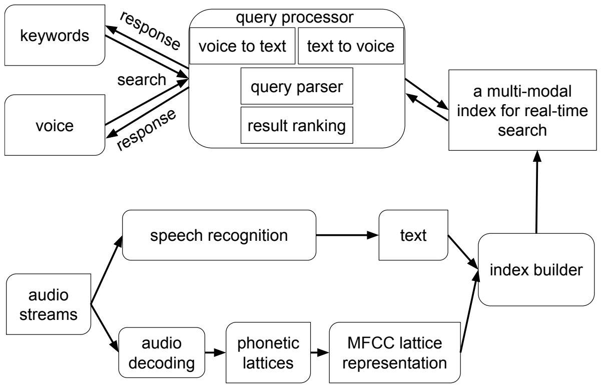


Figure 3. Indexing and querying overview

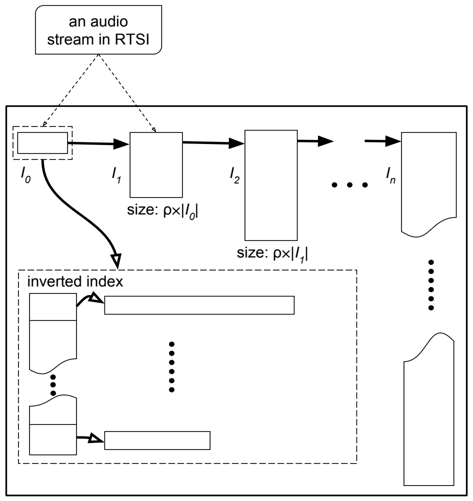


Figure 4. The index for real-time search

or more popular content. For example, they are interested in the audio on the more recent events. Moreover, the audio streams should be relevant (to some extent) to the query. So, freshness, popularity as well as relevance need to be considered and be accessed efficiently. We maintain information for relevance, popularity and freshness in the inverted indices of the LSM-tree with three inverted lists for each term for real-time search. Given a query, the information of freshness, significance and relevance of a term in an audio stream is obtained through intersecting the three inverted lists. As a result, we address Challenge (i).

**Query answering**: When handling queries, users can use either keywords or voice as input. Our query processor converts keywords into voice or voice to keywords for tolerance of transcription or phonetic lattice presentation inaccuracy. Then, the voice is decoded into phonetic lattices. We retrieve from the indices all the audio streams which contain the keywords or phonetic lattices. Finally, we compute the score for each audio stream and obtain the top-$k$ audio streams and respond the user queries.

The weights can be customized by users. More specifically, the results shown in Figure 2 are from a query that uses 20, 20 and 60 for the weights of freshness, popularity and relevance, respectively. Through "Advanced Setting" in Figure 1, we can change the weights of them to 80, 10 and 10, respectively, to give higher ranks for fresher streams.

When answering the query for hot terms in a given time frame, we obtain all the inverted list windows which are within the time frame. Then, we rank the terms based on the frequencies and select the terms with the highest frequencies. To improve the efficiency, we exploit pruning techniques to reduce the number of terms to evaluate.

**The multi-modal index**: The index we build on the transcribed text and phonetic lattices is based on our proposed RTSI [4]. The RTSI index is based on log structured merge-tree (LSM-tree) indexing. The LSM-tree has multiple inverted indices. The second inverted index $I_1$ is $\rho$ times larger than the first inverted index $I_0$ and is $\frac{1}{\rho}$ of the third inverted index $I_2$. This log structured amortizes the cost of merging indices and improves the overall insertion and search efficiency. The insertion cost is approximately the cost of inserting a term to $I_0$. Hence, RTSI can support massive insertions together with queries. Figure 4 gives an example of the LSM-tree in RTSI. Here, we like to point out that RTSI allows one audio stream to appear in multiple inverted indices, which enables live audio streams to be searched.

In our recent work on the RTSI index [4], we used two individual LSM-trees: one for indexing transcribed text and the other for indexing phonetic lattices. Here we further improve RTSI, and implement the multi-modal functionality by using only one LSM-tree: each inverted list is associated with an entry for text dictionary and an entry for phonetic lattices dictionary. Thus, we achieve a more memory friendly index for real-time search.

To allow fast access to the audio information for computing score, we store the term frequency into the inverted list which is sorted in descending order. We also maintain a hash table for popularity and freshness of each audio stream. This hash table of the index is small because the number of keys in the hash table equals to the number of audio streams.

With the hash table for popularity and freshness of each audio streams in the index, we can easily obtain popularity and freshness. For computing the score of an audio stream using Equation 1, we also need to obtain the total frequency of a term that matches the query. However, obtaining the total term frequency of a term may requires accessing multiple inverted indices, because an audio stream may appear in multiple inverted indices. To avoid accessing multiple inverted indices for improving query efficiency, we maintain another small hash table which keeps track of the existing term frequency of a term. This hash table is small because it only stores the live audio streams which consists of only a small proportion of all the audio streams in the audio streaming platform. Consequently, we address Challenge (ii). Furthermore, our proposed index exploits LSM-trees which efficiently support massive insertion together with queries for addressing Challenge (iii).

**Merge indices**: We merge two inverted indices if the size of index $I_0$ exceeds the memory limit. To support queries while merging, we create mirrors for the indices. When creating a mirror for the inverted index, we can create the mirror for $I_i$ or $I_{i+1}$. In our implementation, we create the mirror for $I_i$, because $I_i$ is smaller and hence more efficient for the creation. Furthermore, if we merge $I_0$ and $I_1$, we need to create two extra sorted inverted lists for $I_0$ since $I_0$ only has one sorted inverted list for freshness. Readers may refer to our paper [4] for more details.

## 4. Demonstration plan

We have conducted the evaluations on a workstation running Linux with two Xeon E5-2640v4 CPUs (totally 20 CPU cores at 2.40GHz). We plan to demonstrate Aucher with remote access to the workstation. We obtained 6,000 audio streams from Voice of America (VOA) and 80,000 audio streams from Ximalaya [7]—an audio streaming service. The total length of the audio streams is about 2,133 hours, and the average length of an audio stream is about 16 minutes. We will show case that Aucher can work with different types of audio streams. We aim to demonstrate three scenarios.

**(i) Keyword search and voice search**: In this scenario, we aim to compare Aucher with the state-of-the-art techniques. We demonstrate searching for the audio streams of interest using keywords (e.g., "piano"). In the demonstration, we will highlight to the audience what are the live audio streams and what are the historical audio streams among the query results. For voice search, we plan to bring a microphone to the venue to accept query terms. We also prepare some audio queries stored locally as query samples for demonstrations. Moreover, we will demonstrate that Aucher is able to retrieve audio streams similar to the query in terms of the audio content. The existing audio search algorithms based on titles, tags or comments are unable to achieve this result. We will also demonstrate that Aucher is around four times faster than LSII on insertion, and reduces memory consumption by half.

**(ii) Searching hot terms in a time frame**: We will show audience what are the hot terms. The supported queries include "hot-term: today", "hot-term: this week" and "hot-term: this month". For example, "hot-term: today" retrieves the hot terms in the audio streams of today. Based on the retrieved terms, we can infer what events occur on the time frame. For example, in the demo, the audience may make interesting findings for Ximalaya: "eat" is the second mostly mentioned term; people discuss much on education, because "kid" and "teacher" are among the hot terms.

**(iii) Adding multiple audio streams and search**: We show the audience that Aucher can handle adding multiple audio streams transparently, while supporting search. This is an important experience for users. We will demonstrate two cases on adding multiple audio streams: 1) adding audio streams that does not result in index merge, and 2) adding audio streams that results in index merge. We will show the audience that Aucher is able to retrieve the audio streams that are in broadcasting.

## 5. Conclusion

We show that live audio streams can be indexed and searched in a real-time manner using proper indexing techniques. This study demonstrates that the success of machine learning (in speech recognition) and indexing techniques enables real-time and multi-modal search on the live audio stream applications. The proposed techniques are able to improve the timeliness and user experience of live audio stream applications over the state-of-the-art approach.

## Acknowledgements

This work is supported by a MoE AcRF Tier 1 grant (T1 251RES1610) and Tier 2 grant (MOE2017-T2-1-122) in Singapore. We also thank Xingyang Liu and Hongjian Cao for the initial development of the Aucher system prototype.

## References

[1] G. Richard, S. Sundaram, and S. Narayanan, "An overview on perceptually motivated audio indexing and classification," *Proceedings of the IEEE*, vol. 101, no. 9, pp. 1939–1954, 2013.

[2] A. Wang *et al.*, "An industrial strength audio search algorithm." in *Ismir*, vol. 2003, 2003, pp. 7–13.

[3] F. Zheng, G. Zhang, and Z. Song, "Comparison of different implementations of mfcc," *Journal of Computer Science and Technology*, vol. 16, no. 6, pp. 582–589, 2001.

[4] Z. Wen, X. Liu, H. Cao, and B. He, "Rtsi: An index structure for multi-modal real-time search on live audio streaming services," in *International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 1495–1506.

[5] L. Wu, W. Lin, X. Xiao, and Y. Xu, "Lsii: An indexing structure for exact real-time search on microblogs," in *International Conference on Data Engineering (ICDE)*, 2013, pp. 482–493.

[6] Mixlr, "Broadcasting live audio made simple," 2018. [Online]. Available: http://mixlr.com/

[7] Ximalaya, "Enabling users to share audio and personal radio stations," 2018. [Online]. Available: http://www.ximalaya.com/

[8] Lizhi, "Lizhi FM: a Chinese podcast platform," 2018. [Online]. Available: http://www.lizhi.fm/

[9] H. Nagano, R. Mukai, T. Kurozumi, and K. Kashino, "A fast audio search method based on skipping irrelevant signals by similarity upper-bound calculation," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 2324–2328.

[10] V. Gupta, J. Ajmera, A. Kumar, and A. Verma, "A language independent approach to audio search," in *Annual Conference of the International Speech Communication Association*, 2011.

[11] R. Typke, F. Wiering, and R. C. Veltkamp, "A survey of music information retrieval systems," in *Proc. 6th International Conference on Music Information Retrieval*, 2005, pp. 153–160.

[12] R. Shadiev and Y.-M. Huang, "Facilitating cross-cultural understanding with learning activities supported by speech-to-text recognition and computer-aided translation," *Computers & Education*, vol. 98, pp. 130–141, 2016.

[13] Y. Li, B. He, R. J. Yang, Q. Luo, and K. Yi, "Tree indexing on solid state drives," *Very Large Data Bases (VLDB) Conference*, vol. 3, no. 1-2, pp. 1195–1206, Sep. 2010.

[14] A. Magdy, L. Alarabi, S. Al-Harthi, M. Musleh, T. M. Ghanem, S. Ghani, and M. F. Mokbel, "Taghreed: a system for querying, analyzing, and visualizing geotagged microblogs," in *SIGSPATIAL*. ACM, 2014, pp. 163–172.