ZASHAM

Recreating shazam with audio fingerprinting and matching.

GROUP 8:
Blanca Burgaleta, Spencer Wood, Louis-Esmel Kodo,
Abdallah AlShaarawi, Enrico Tajanlangit

AGENDA FOR TODAY

- THE PROBLEM
- OUR SOLUTION
- OUR APPROACH
- CHALLENGES FACES
- LIVE DEMO
- KEY FINDINGS
- FINAL REFLECTIONS

WHAT is the problem?

“I LOVE THIS SONG!!”

“What is that song called again...?”

“I’ll never hear that remix again!"

“Zasham is a music recognition system inspired by the popular app Shazam, which can identify a song just by "listening" to a short audio clip. The goal of our project is to replicate the core idea behind Shazam and build a **simplified version** of this system **using Python, Librosa, and Apache Spark on Databricks.”**

— Zasham’s founding team

Why did we build ZASHAM?

“What’s That Song?!” Struggle

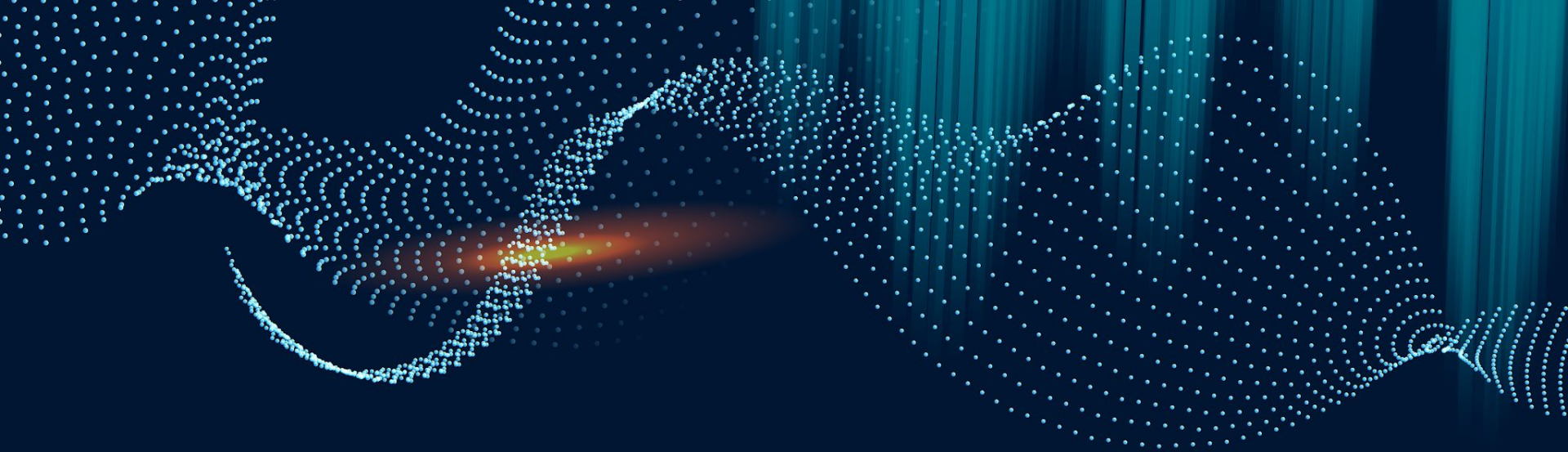
We’ve all been there—you hear a song, love it, but have no idea what it’s called. It’s playing in a café, on the radio, or in a random video, and now it’s **stuck in your head forever.** Frustrating, right?

The Shazam Limitation

Shazam is amazing, but it only recognizes songs from **official music databases and licensing agreements with record labels..** What if your mystery song is a cover or a remix?

Our Solution: ZASHAM

We took things up a notch by using the **YouTube API**, tapping into a massive, diverse music pool. This means we can spot tracks that Shazam might miss—including underground hits and indie gems!



01 CUSTOM DATA CREATION

Instead of using a pre-existing dataset, **we built our own song library using the YouTube API.** By inputting song titles, we retrieved and downloaded audio directly from YouTube, ensuring a diverse and customizable dataset tailored to our project's needs.

02 ENVIRONMENT SET UP

ETL Pipeline Implementation: We structured our workflow using a typical Extract, Transform, Load (ETL) pipeline to efficiently process and manage our audio data.

Bronze, Silver, Gold Architecture: We stored our data using Databricks' Distributed File System (DBFS), which functions similarly to Hadoop, organizing data in three structured layers:

- Bronze Layer: **Raw, unprocessed audio files** and metadata.
- Silver Layer: **Preprocessed, fingerprinted audio data** in a structured format.
- Gold Layer: **Final hashed fingerprints** optimized for fast matching and retrieval.

Spark for Distributed Processing: We used Apache Spark on Databricks to handle large-scale data transformations, including hashing and matching audio fingerprints in a scalable manner.

(Future Enhancement) **Kafka for Streaming**: We planned to incorporate Apache Kafka for real-time streaming and on-the-fly fingerprint matching. However, due to limitations in audio processing libraries, we were unable to implement this in our current version but see it as a valuable future improvement.

DATA PROCESSING: HOW DID WE BUILD ZASHAM?

STEP 1
Connect to the YouTube API

STEP 2
Download original WebM files

STEP 3
Convert WebM files into MP3 files

STEP 4
Convert audio into a spectrogram (Librosa)

STEP 5
Detect spectral peaks (unique fingerprints)

DATA PROCESSING: HOW DID WE BUILD ZASHAM?

STEP 6
Create fingerprint pairs (constellation maps)

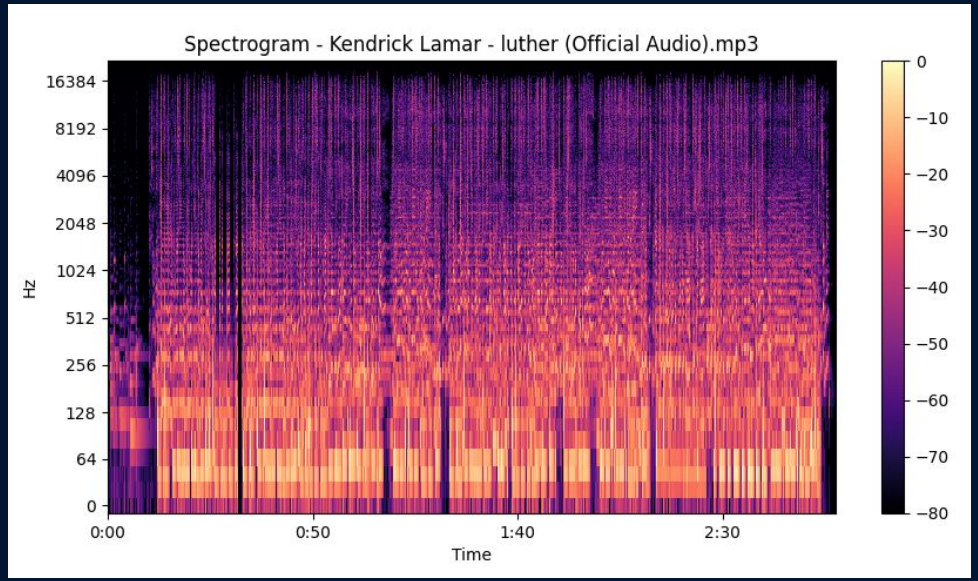
STEP 7
Flattening array of arrays (frequencies & timestamps)

STEP 8 (SPARK)
Convert to hashes for fast database matching

STEP 9 (SPARK)
Compare clip fingerprints to a song database in Apache Spark

STEP 10 (SPARK)
Display top matches with confidence intervals

🔍 **STEP 4:** Converting audio into a spectrogram

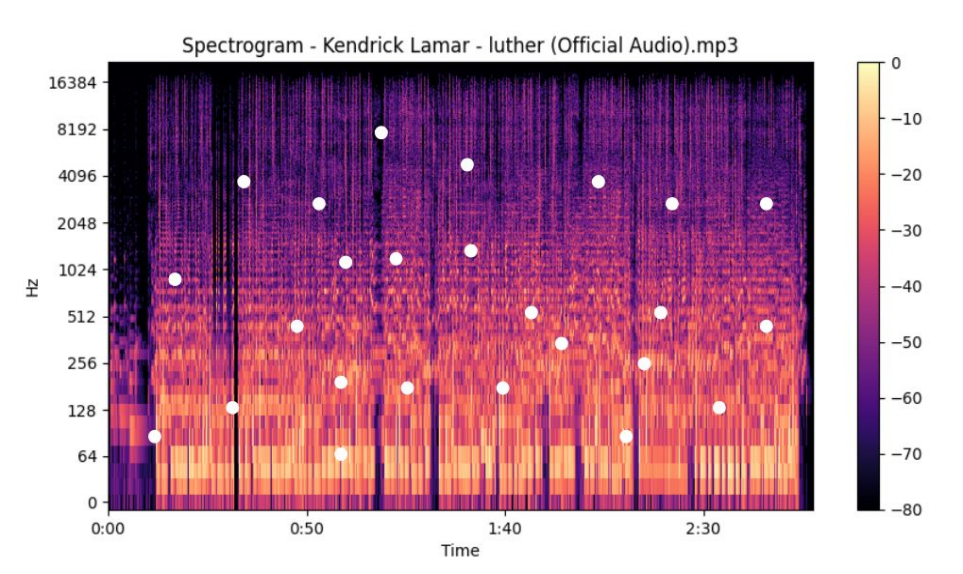


The spectrogram displays:

- Pitch & frequency changes of the song over time
- How loud or strong each frequency is

**Note: Data Pipeline: Local Processing (Librosa) → Spark Processing (Databricks)*

🔍 **STEP 5:** Detecting Spectral Peaks



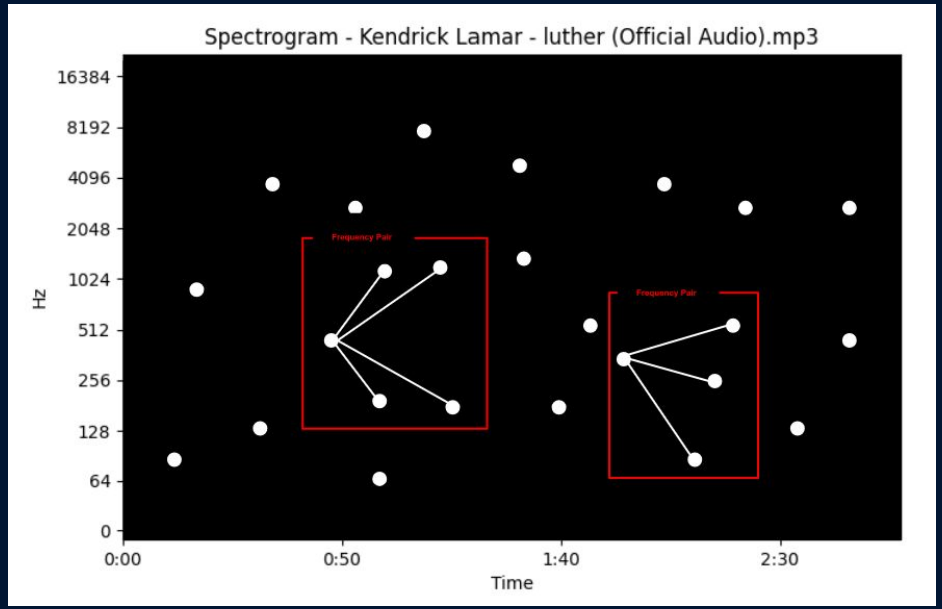
Rather than analyzing the entire spectrogram, Zasham focuses only on the most important parts, called spectral peaks.

These peaks are like the song's fingerprints — unique and robust even if there's noise or distortion (like background chatter or echo).

You can think of them as musical landmarks.

**Note: Data Pipeline: Local Processing (Librosa) → Spark Processing (Databricks)*

🔍 **STEP 6:** Creating fingerprint pairs (Constellation Maps)



Zasham stores the relationships between peaks:

- It picks a “main” peak (anchor)
- Then it finds other nearby peaks (targets) within a small time window
- Stores the frequency of the anchor & target and the time difference between them

These trios (frequency1, frequency2, time difference) become unique combinations that capture specific moments in the song’s structure.

These relationships form a **constellation map, which is a star-like representation of the song's signature moments.**

**Note: Data Pipeline: Local Processing (Librosa) → Spark Processing (Databricks)*

🔍 **STEP 8:** Turning Fingerprints into Hashes

Each (freq1, freq2, time difference) combination is converted into a small, fixed-length code called a hash, which is easy to store, fast to compare, and don't take much space.

Why hash them?

- They're lightweight and searchable.
- You don't need to keep the full audio.
- It makes it very fast to match songs later.

Hashes don't include when exactly the peak happened — just the pattern itself, so that clips can be matched even if they come from a different part of the song.

**Note: Data Pipeline: Local Processing (Librosa) → Spark Processing (Databricks)*

🔍 **STEP 9:** Matching a Clip to a Song

When someone plays a short audio clip:

1. Zasham repeats the same process on the clip: **spectrogram → peaks → peak pairs → hashes.**
2. It then **searches in its database of song hashes to see which ones match**.
3. It checks how closely the timing of matches align (even though we didn't hash time, we kept it separately to calculate offsets).
4. The song with the most matching hashes and consistent timing alignment is considered the **best match**.

**Note: Data Pipeline: Local Processing (Librosa) → Spark Processing (Databricks)*

MAIN CHALLENGES FACED

Limitations with 3rd party libraries

Librosa and yt-dlp are python and command-line based audio libraries, is not designed for distributed computing in Spark.

Hardware and environment constraints

Fingerprinting generates large datasets, leading to memory constraints in a virtual machine (VM) setup

Real-time processing limitations

Apache Kafka had limitations in transmitting full audio clips for live processing.

SO, HOW DID WE SOLVE THEM?

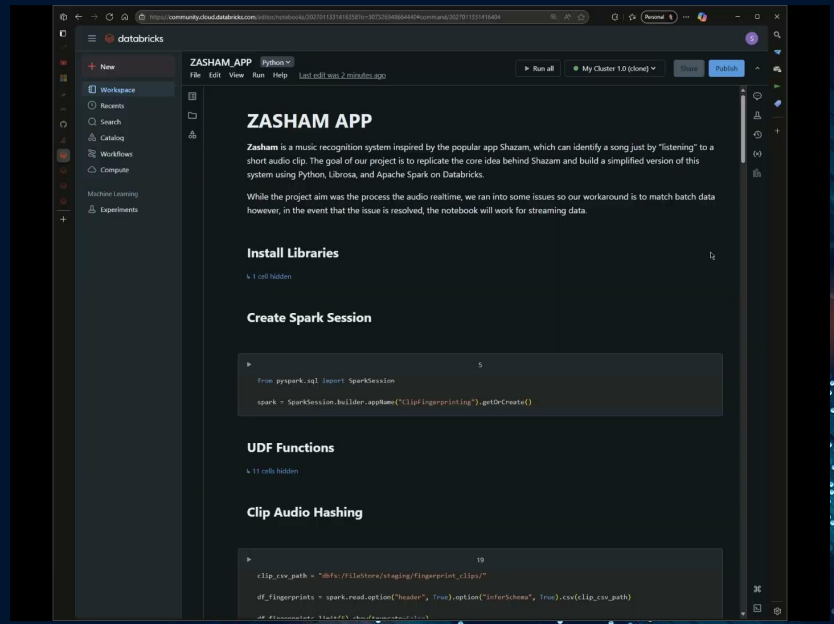
WITH DATABRICKS

1. **We preprocessed audio files outside Spark,** then stored structured fingerprint data in a Databricks File System (DBFS) for distributed matching.
2. **We transitioned to Databricks,** optimized data storage using Parquet format, and processed structured fingerprints instead of raw audio.
3. **We pivoted to an offline batch processing model** but left the door open for future real-time enhancements.

LET'S WATCH A DEMO OF ZASHAM!

Audio Processing (Python + Librosa)

ZASHAM App (Databricks)



FINDINGS SUMMARY

BUSINESS IMPACT

Identifies songs with **70% accuracy**

Finds similar songs (same artist, genre, remixes) → **Opportunity for music recommendations**

Scalable approach → Fast & lightweight storage (hashing)

TECHNICAL TAKEAWAY S

Confidence intervals improve accuracy & ranking

Hashing speeds up queries & reduces storage footprint

False positives due to frequency overlaps

NEXT STEPS

Improve precision with AI-based filtering

Explore real-time matching & recommendations

WHAT IS NEXT IN THE FUTURE OF ZASHAM?

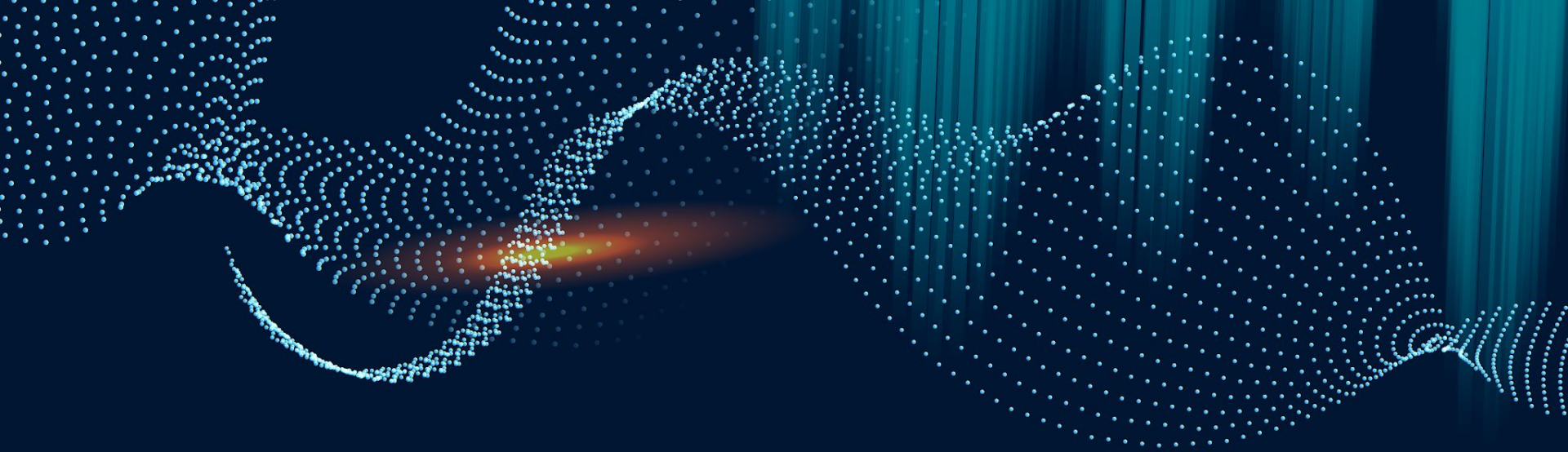
- The next step is to **convert the entire pipeline to Spark** for improved efficiency and scalability.
- We aim to **integrate Kafka streaming to process live MP3 data in real time.** This enhancement would allow us to move beyond batch processing and replicate the real-time functionality of Shazam, making Zasham truly dynamic and responsive.
- **A lightweight web app** could make Zasham accessible to everyday users, providing an intuitive interface for song identification.
- Zasham is just the beginning! We are only scratching the surface of what is possible with the power of Big Data...

IMPORTANCE OF BIG DATA IN THIS PROJECT

Our project highlights how big data solutions can revolutionize how we interact with the world, from something as simple as identifying a song to more complex real-time audio analytics in healthcare, security surveillance, entertainment, etc.

IMPORTANCE OF BIG DATA IN THIS PROJECT

Our project highlights how big data solutions can revolutionize how we interact with the world, from something as simple as identifying a song to more complex real-time audio analytics in healthcare, security surveillance, entertainment, etc.



THANK YOU!

FINDING SUMMARY

Beyond matching: Zasham finds similar songs, making it **useful for recommendations**.

HASHING MAKES IT EFFICIENT

Confidence intervals improve accuracy and help rank possible matches.

Hashing speeds up queries & minimize storage, making the model efficient.

Zasham successfully **recognizes songs**.

70%
Matched Accuracy

IT WORKS !

Solid start, with room for further refinement.

FINDING SUMMARY

Beyond matching: Zasham finds similar songs, making it **useful for recommendations**.

Confidence intervals improve accuracy and help rank possible matches.

Zasham successfully **recognizes songs**, with 70% accuracy.

70%
Matched Accuracy

HASHING MAKES IT EFFICIENT

Hashing speeds up queries & minimize storage, making the model efficient.