

# Audio fingerprinting in WebAssembly per l'esecuzione in browser web

*Candidato:* Davide Pisanò    *Relatore:* Antonio Servetti

Collegio di Ingegneria Informatica, del Cinema e Meccatronica  
Politecnico di Torino

Laurea Magistrale in Ingegneria Informatica, Luglio 2023



# Fingerprinting Audio nel Browser

- ▶ Per molti anni capacità dei browser limitate
  - ▶ Scarse performance con Javascript
  - ▶ Niente operazioni in real time
  - ▶ Computazioni a carico del server
- ▶ Ora browser più avanzati
  - ▶ Computazioni parzialmente sul client
  - ▶ Server più scarichi

# Fingerprinting Audio nel Browser

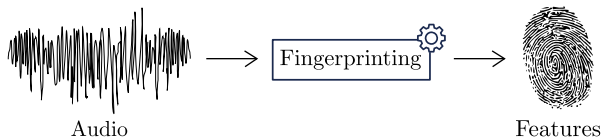
- ▶ Per molti anni capacità dei browser limitate
  - ▶ Scarse performance con Javascript
  - ▶ Niente operazioni in real time
  - ▶ Computazioni a carico del server
- ▶ Ora browser più avanzati
  - ▶ Computazioni parzialmente sul client
  - ▶ Server più scarichi

# Il Fingerprinting Audio

- ▶ Il **fingerprint audio** è l'impronta digitale di un audio
  - ▶ Ogni fingerprint identifica univocamente un audio
- ▶ Il **fingerprinting audio** crea il fingerprint dell'audio
  - ▶ L'algoritmo estrae delle features significative dell'audio
  - ▶ Un brano può essere ricercato note *alcune* sue features

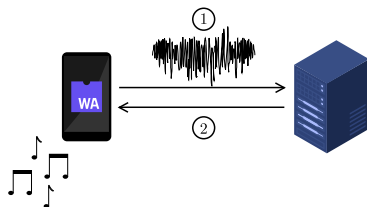
# Il Fingerprinting Audio

- ▶ Il **fingerprint audio** è l'impronta digitale di un audio
  - ▶ Ogni fingerprint identifica univocamente un audio
- ▶ Il **fingerprinting audio** crea il fingerprint dell'audio
  - ▶ L'algoritmo estrae delle features significative dell'audio
  - ▶ Un brano può essere ricercato note *alcune* sue features



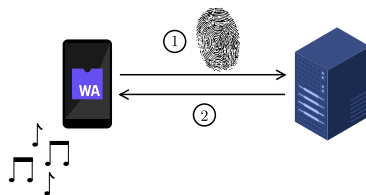
# Rivedere il modello C/S

## C/S classico



1. Client invia audio
2. Server risponde

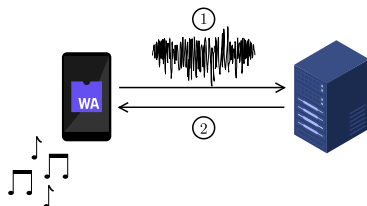
## C/S rivisto



0. Client calcola fingerprint
1. Client invia fingerprint
2. Server risponde

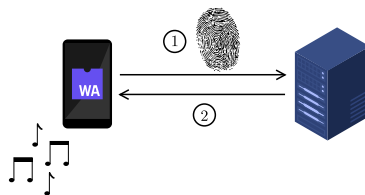
# Rivedere il modello C/S

## C/S classico



1. Client invia audio
2. Server risponde

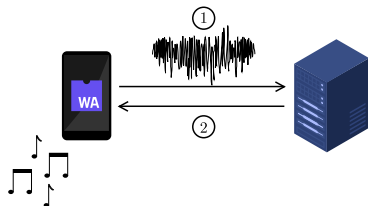
## C/S rivisto



0. Client calcola fingerprint
1. Client invia fingerprint
2. Server risponde

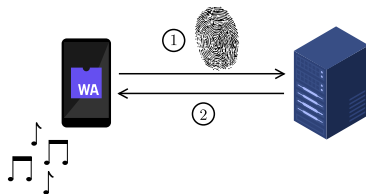
# Rivedere il modello C/S

## C/S classico



1. Client invia audio
2. Server risponde

## C/S rivisto



0. Client calcola fingerprint
1. Client invia fingerprint
2. Server risponde



## ▶ **Riconoscimento audio**

- ▶ Audio simili condivideranno molte features
- ▶ Es: *Shazam*, *ACRCloud*

## ▶ **Sincronizzazione** di fonti multimediali

- ▶ È possibile sincronizzare due stream audio distinti
- ▶ Es: second screen application

## ▶ **Pubblicità** personalizzata

- ▶ Inserimento di un segmento audio personalizzato

## ▶ **Riconoscimento audio**

- ▶ Audio simili condivideranno molte features
- ▶ Es: *Shazam*, *ACRCloud*

## ▶ **Sincronizzazione** di fonti multimediali

- ▶ È possibile sincronizzare due stream audio distinti
- ▶ Es: second screen application

## ▶ **Pubblicità** personalizzata

- ▶ Inserimento di un segmento audio personalizzato

## ▶ **Riconoscimento audio**

- ▶ Audio simili condivideranno molte features
- ▶ Es: *Shazam*, *ACRCloud*

## ▶ **Sincronizzazione** di fonti multimediali

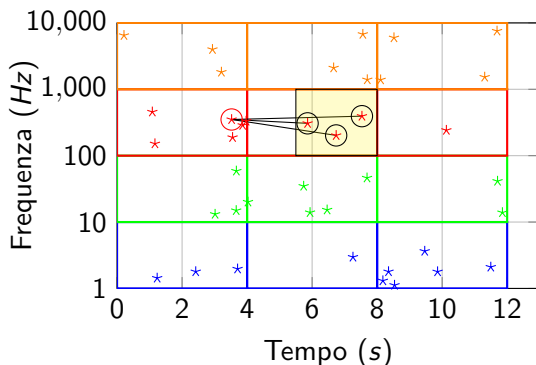
- ▶ È possibile sincronizzare due stream audio distinti
- ▶ Es: second screen application

## ▶ **Pubblicità** personalizzata

- ▶ Inserimento di un segmento audio personalizzato

# Estrazione delle Features

- L'estrazione delle features consiste in:
  1. Individuare i picchi nel brano
  2. Correlare un picco *anchor* con gli altri picchi
  3. Per ogni coppia *anchor-picco* calcolare l'hash



# Wasm e WebAudio

È possibile eseguire l'algoritmo nel browser grazie a:

## Wasm



- ▶ Formato per codice binario
- ▶ Eseguitibile nel browser
- ▶ Target per linguaggi di alto livello

## WebAudio



- ▶ API Javascript
- ▶ Manipolazione audio

# Wasm e WebAudio

È possibile eseguire l'algoritmo nel browser grazie a:

## Wasm



- ▶ Formato per codice binario
- ▶ Eseguitibile nel browser
- ▶ Target per linguaggi di alto livello

## WebAudio



- ▶ API Javascript
- ▶ Manipolazione audio

È possibile eseguire l'algoritmo nel browser grazie a:

## Wasm



- ▶ Formato per codice binario
- ▶ Eseguitibile nel browser
- ▶ Target per linguaggi di alto livello

## WebAudio

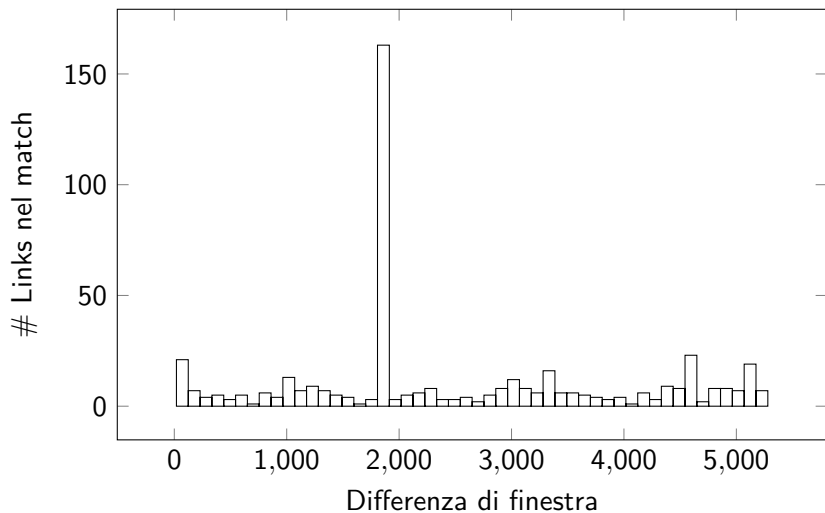


- ▶ API Javascript
- ▶ Manipolazione audio

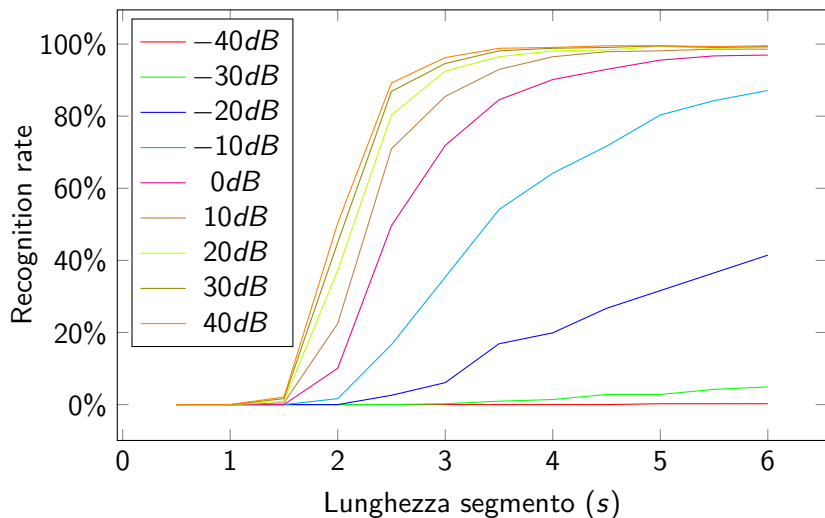
- ▶ Basato su tre idee:
  1.  $\Delta T \geq 0$  tra features brano originale e registrazione
  2.  $\Delta T$  costante tra features brano originale e registrazione
  3. Gli hash delle features devono coincidere
- ▶ Quindi bisognerà:
  1. Raggruppare per  $\Delta T$  e per id brano originale
  2. Ogni gruppo da  $n$  features
  3. Ordinare per  $n$  crescente
  4. **Miglior match**: gruppo con  $n$  maggiore



# Riconoscimento: Visualizzazione



# Recognition rate



# Second screen application

