

Obecná audio podobnost (NI-VMM)

Popis projektu

Projekt je zaměřen na implementaci podobnostní míry pro audio skladby. Aplikace by měla obsahovat databázi audio souborů. Uživatel se následně může dotázat vlastním audio dotazem (skrze webové rozhraní) do databáze a aplikace vrátí množinu podobných audio souborů v databázi. V rámci projektu je třeba naimplementovat extrakci deskriptorů ze zvoleného typu audio souborů a navrhnout a implementovat na nich vlastní podobnostní míru.

Vstup: Audio soubor.

Výstup: Množina databázových audio souborů podobných vstupnímu audiu setříděných podle podobnosti s možností jejich přehrání.

Aplikace by měla obsahovat části:

- Modul extrakce deskriptorů z audia
- Modul podobnostní míra pro porovnání dvojice audio souborů, tj. jejich deskriptorů
- Modul identifikace podobných databázových audio záznamů s ohledem na vstupní audio
- Webový interface

Způsob řešení

Náš způsob extrakce deskriptorů a jejich následné porovnání je inspirováno aplikací Shazam. Nejprve si přečteme a resamplujeme WAV audio soubor z 44,100 Hz na 11,050 Hz s cílem potlačit vliv šumu. Následně získáme spektrogram pomocí krátkodobé Fourierové transformace (Short-Time Fourier Transformation), která analyzuje signál po krátkých časových úsecích. Vzniklý spektrogram rozdělíme na jednotlivé pásma frekvencí: 0-250 Hz, 250-520 Hz, 520-1,450 Hz, 1,450-3,500 Hz, 3,500-5,512 Hz.

Pro zjednodušení porovnávání chceme ze spektrogramu získat pouze ty frekvence, které jsou pro nás nejdůležitější, tzv. „peak frequencies.“ Pro extrakci jsme implementovali 2 metody:

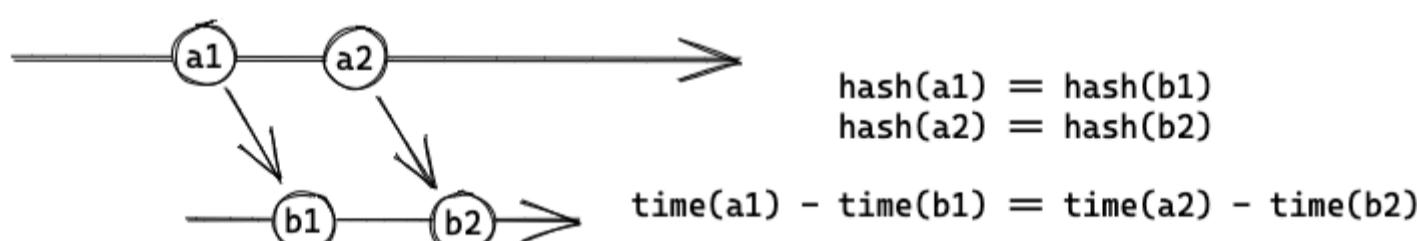
- Metoda [bands](#) : pro každý časový úsek a každé pásmo si vybereme „nejsilnější“ frekvenci. Z těchto frekvencí vybereme pouze ty, které mají větší intenzitu než klouzavý průměr intenzit vybraných frekvencí,
- Metoda [prominence](#) : použili jsme metodu `scipy.signal.find_peaks`, která naleze lokální maxima ve spektrogramu pro každý časový úsek a vrátí prvních N frekvencí s nejvyšší prominencí.

Vzniklé "peak frequencies" poté seřadíme: nejprve vzestupně podle času výskytu a poté vzestupně podle frekvence. Po jejich seřazení jsme schopni seskupit frekvence do pod-posloupnosti (hash) za účelem zvýšení entropie při vyhledávání. V této práci jsme implementovali 2 metody seskupení:

- Metoda [fanout](#) , který kombinatoricky generujeme páry s rostoucím offsetem,
- Metoda [cluster](#) , který posouvá okénko nad seřazenou posloupností.

`(hash, time(hash[0]))` pak tvoří tzv. otisk audio nahrávky, který posléze nahráváme do databáze.

Tento postup provedeme pro celý dataset skladeb při nasazení aplikace. Jakmile přijde nahrávka na vstupu, tak ho zpracujeme stejným způsobem a porovnáme jeho otisky s našimi skladbami v databázi. Získáváme tím páry otisků (otisk nahrávky, otisk skladby z databáze).



Pro příklad, nechť ilustrujme 2 nalezené páry otisků. Každý takto nalezený pár se musí shodovat v posloupnosti. Zároveň platí, že pokud 2 páry pocházejí ze stejné skladby, tak jejich rozdíl mezi časovým výskytem otisku v nahrávce a časovým výskytem otisku ve skladbě, bude stejný. Počet takto shodujících párů pak bereme jako míru podobnosti mezi vstupní nahrávkou a danou skladbou z databáze.

Implementace

Celé porovnávaní je implementováno v Pythonu verze 3.9 za pomocí těchto knihoven:

- [librosa](#) pro práci s audiem,
- [NumPy](#) pro vektorizované matematické operace,
- [SciPy](#) pro konkrétní implementaci metody [prominence](#) .

Spouštění Python skriptu probíhá skrze Node.js funkci [child_process.spawn](#) pro spouštění procesů.

Webové rozhraní je React aplikace využívající [Create-T3-app](#).

Pro ukládání hashů a metadat o skladbách používáme knihovnu [Prisma](#) a [SQLite](#).

Spouštění

Pro zprovoznění doporučujeme použít připravený `docker-compose.yml`, který spolu s aplikací nastaví i databázi. Samotný dataset není součástí zdrojového kódu a bude nutné si jej připravit zvlášť. Aplikace bude vyhledávat skladby a jejich příslušné metadata ve složce `/dataset`.

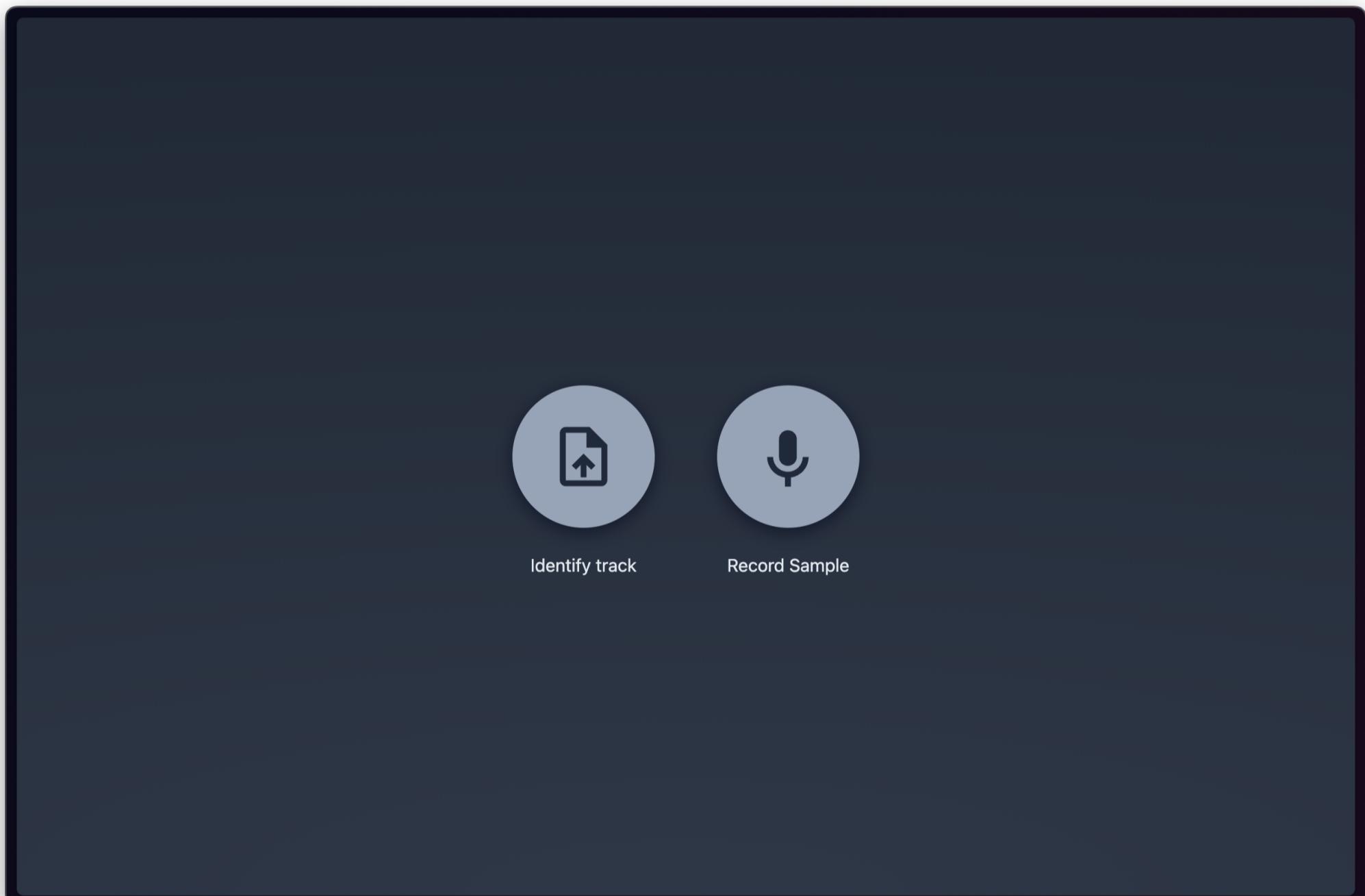
Metadata skladeb (název skladby, autoři, obrázky, apod.) se získavají přečtením souboru `/dataset/index.json` s následující strukturou:

```
[  
  {  
    "filename": "[název souboru]",  
    "title": "REEBOKS OR THE NIKES",  
    "artists": [{ "name": "Jakey" }],  
    "album": { "name": "ROMCOM" },  
    "thumbnails": [  
      {  
        "url": "https://lh3.googleusercontent.com/xm-Ex3VSqwICoAB4gSGNtKiCETfh_lJ6_dYg2CpPqBlrdQjA1Rq8YZLDZUvTeyNfMsbNQTwbrbjivxQsTg  
      }  
    ]  
  }  
]
```

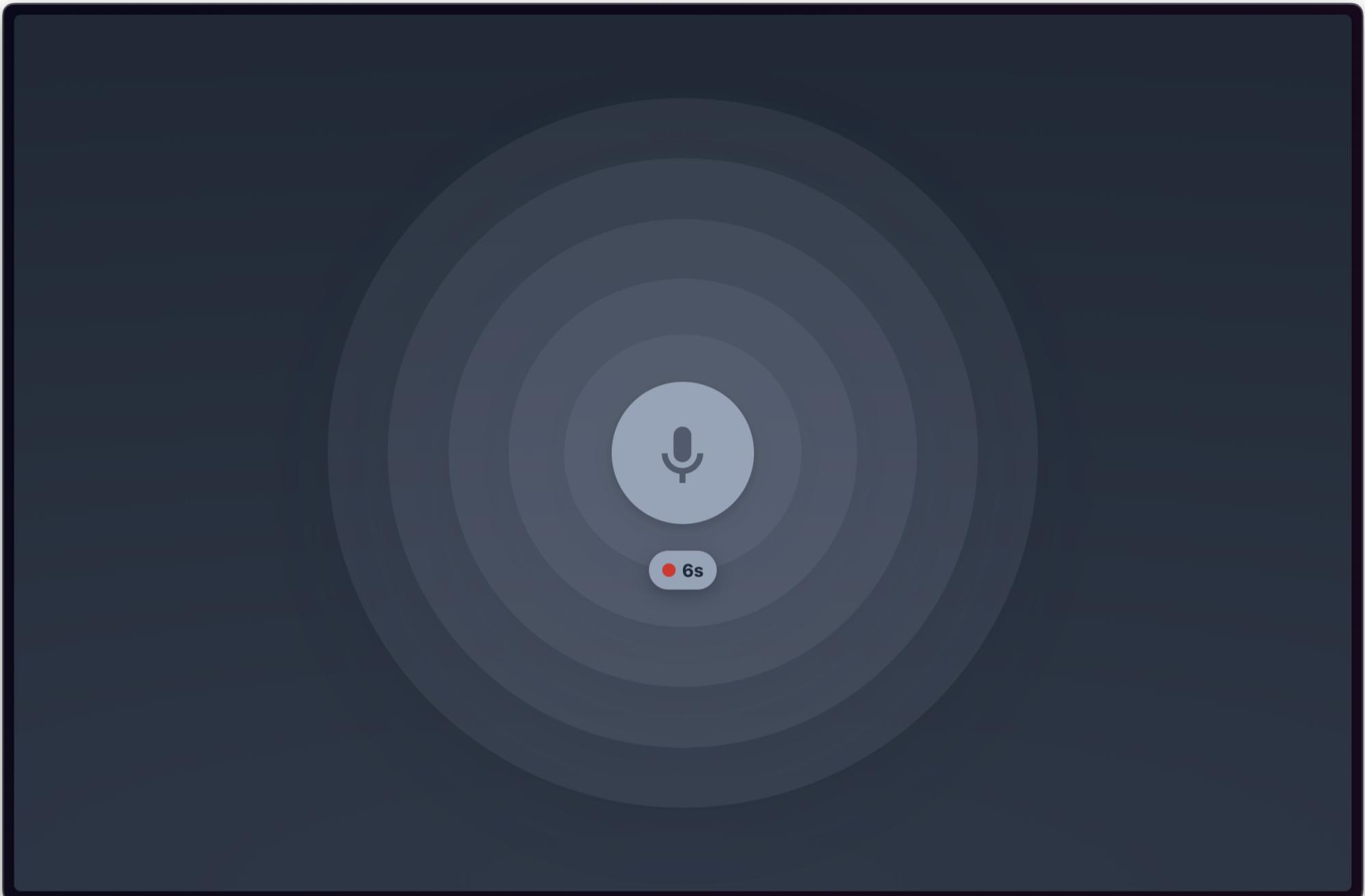
Samotný import se provádí na adrese [/import](#) v prohlížeči.

Příklad výstupu

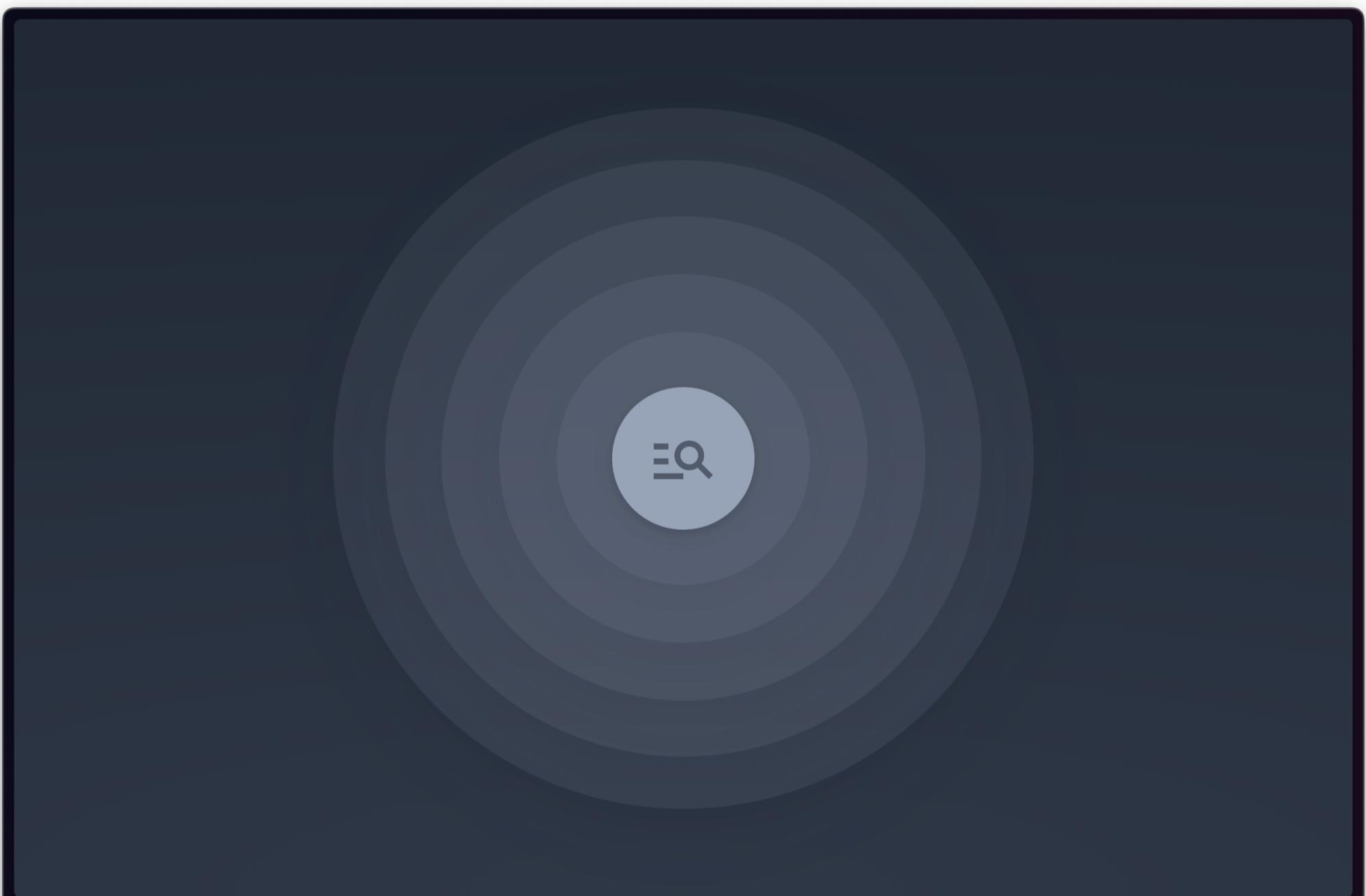
Úvodní stránka:



Nahrávání z mikrofonu:



Porovnávání záznamů:



The screenshot shows a dark-themed mobile application interface. At the top, there are two large circular buttons: one with an upward arrow icon labeled "Identify track" and another with a microphone icon labeled "Record Sample". Below these buttons is a search bar containing the text "REEBOKS OR THE NIKES". Underneath the search bar is a list of search results. Each result item includes a small thumbnail image, the song title, the artist name, the number of matches found, and a "Play" button. The results are as follows:

- 1 REEBOKS OR THE NIKES Jakey 247 matches Play
- 2 Witch Apashe, Alina Pash 48 matches Play
- 3 Tantrum Ashnikko 32 matches Play
- More Than a Friend girli 31 matches Play
- 6.24 Danger 17 matches Play
- DRIVE OFF A BRIDGE Jakey 16 matches Play

Správa skladeb:

The screenshot shows a "Manage songs" page with a grid of song entries. Each entry consists of a checkbox followed by the song title, artist, and additional details. At the top right of the grid are three buttons: "Select all" (blue), "Import" (blue), and "Truncate" (red). The songs listed are:

| <input type="checkbox"/> | I'm Gone | Oliver Tree Ugly is Beautiful With URL Image |
|--------------------------|---------------------------|--|
| <input type="checkbox"/> | Freaks & Geeks | Oliver Tree Cowboy Tears With URL Image |
| <input type="checkbox"/> | Movement | Oliver Tree Movement With URL Image |
| <input type="checkbox"/> | Melancholy (Acoustic) | AViVA Melancholy (Acoustic) With URL Image |
| <input type="checkbox"/> | Sure Sure - Funky Galileo | David Dean Burkhart With URL Image |
| <input type="checkbox"/> | still feel. | half-alive With URL Image |
| <input type="checkbox"/> | Cascade | plenka Cascade |
| <input type="checkbox"/> | Game of Survival | Ruelle Madness With URL Image |
| <input type="checkbox"/> | Shallows | fknsyd Moontower With URL Image |
| <input type="checkbox"/> | Blue Monday | HEALTH Atomic Blonde (Original Motion Picture Soundtrack) With URL Image |
| <input type="checkbox"/> | Oops | Martin Garrix Oops With URL Image |
| <input type="checkbox"/> | Lalala (Official Video) | Y2K, bbno\$ With URL Image |
| <input type="checkbox"/> | Movement | Oliver Tree With URL Image |
| <input type="checkbox"/> | whereareyou | niteboi whereareyou |
| <input type="checkbox"/> | tossing and turning | Arlie Wait With URL Image |
| <input type="checkbox"/> | Can't stand the feeling | Duskus In Retrospect With URL Image |
| <input type="checkbox"/> | Call Me | plenka Pt. One With URL Image |
| <input type="checkbox"/> | LOSER | Neoni LOSER With URL Image |
| <input type="checkbox"/> | グッバイ (feat. Asako Toki) | Asako Toki With URL Image |
| <input type="checkbox"/> | we should try | niteboi we should try With URL Image |
| <input type="checkbox"/> | Falling | REZZ & Underoath Falling |

Deployment

Pro nasazení jsme použili vlastní VPS, na kterém běží Gitlab Runner. Ten se stará o to, aby při každém commitu se sestavil nový obraz a kontejner skrz příkaz docker-compose up --build -d .

Experimentální sekce

V adresáři [/research](#) jsou uloženy ukázky našeho prozkoumávání možností implementace audio podobnosti. Nejdříve jsme začali s vytažením samotných MFCC a následně jejich porovnáváním pomocí algoritmu Dynamic Time Warping (DTW). To se ukázalo jako poměrně výpočetně náročné a tedy i pomalé. Udělali jsme tedy rešerši implementace Shazamu a inspirovali se jejich vytvářením otisků skrz filtrováním spektrogramu. Výsledek je spolehlivější a rychlejší.

Pro testování jsme použili následující audio soubory: [originální skladbu](#) a [nahrávku originální skladby](#).

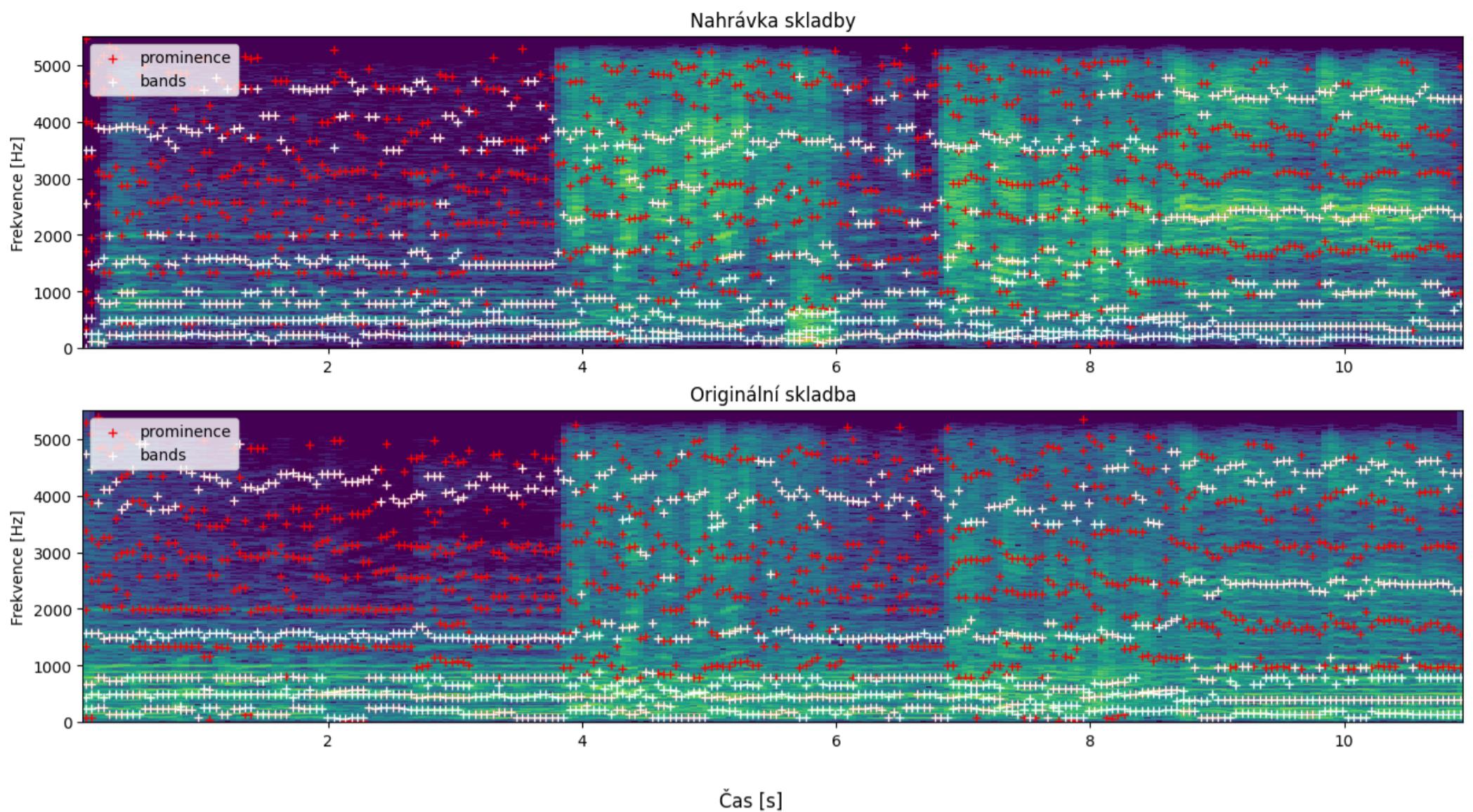
| | | prominence | bands |
|---------|--------------------|-------------------|--------------|
| cluster | Počet shod | 41 | 53 |
| | Procentuální shoda | 2.497% | 4.5377% |
| fanout | Počet shod | 2255 | 2154 |
| | Procentuální shoda | 3.5859% | 6.1009% |

Použili jsme následující parametry:

| Funkce | Parametr | Hodnota |
|------------|-------------|---------|
| bands | avg_window | 10 |
| prominence | num_peaks | 15 |
| | distanace | 100 |
| fanout | fan_out | 50 |
| | tail_size | 1 |
| cluster | window_size | 3 |
| | gap_size | 1 |

Je zjevné, že kombinace metod bands a fanout vede k největší procentuální shodě na testovacích souborech.

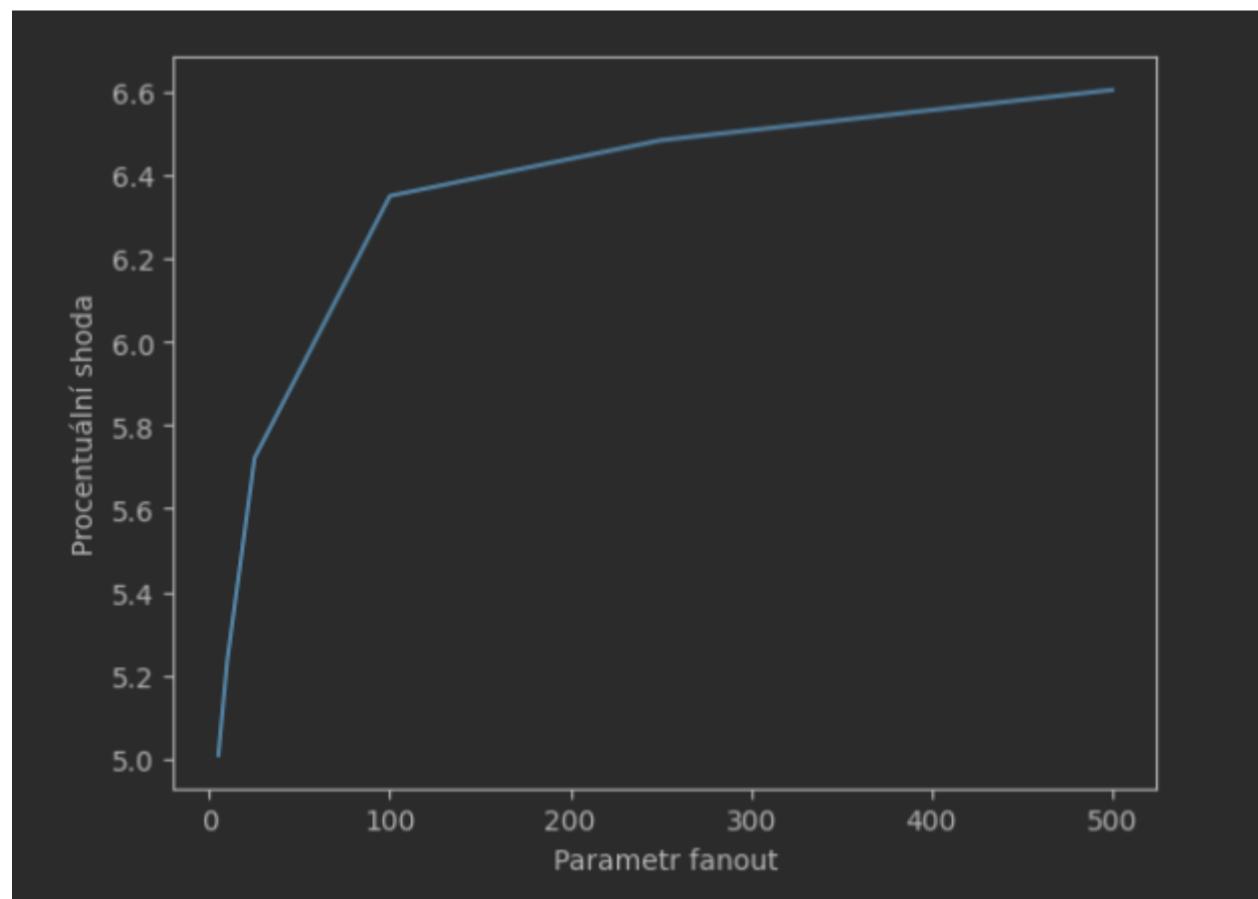
Co se týče tvorby otisků a výpočetní složitosti, tak metoda fanout je zjevně výpočetně náročnější: $O(N \times \text{fan_out})$ u metody fanout , kde N je počet frekvencí oproti $O(N)$ u metody cluster .

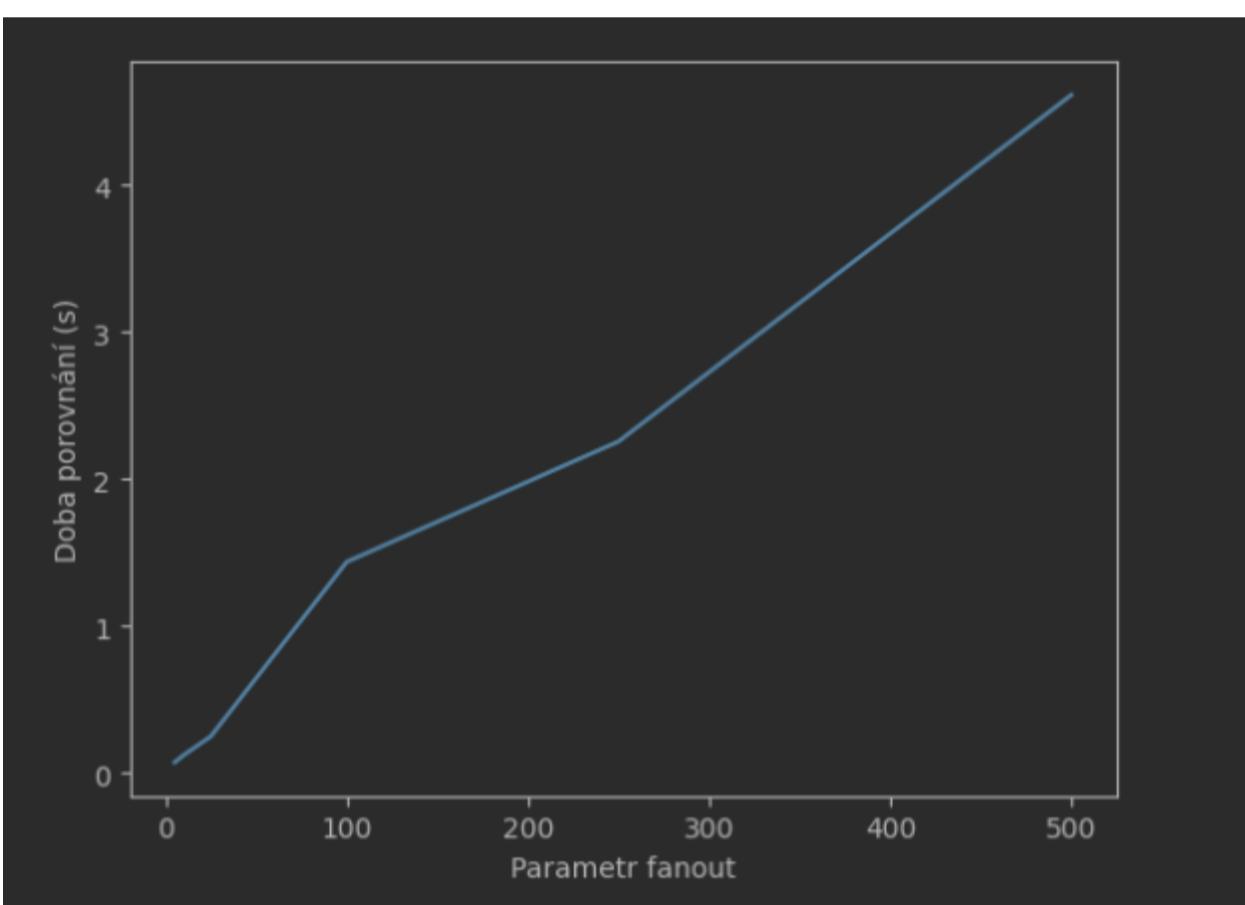
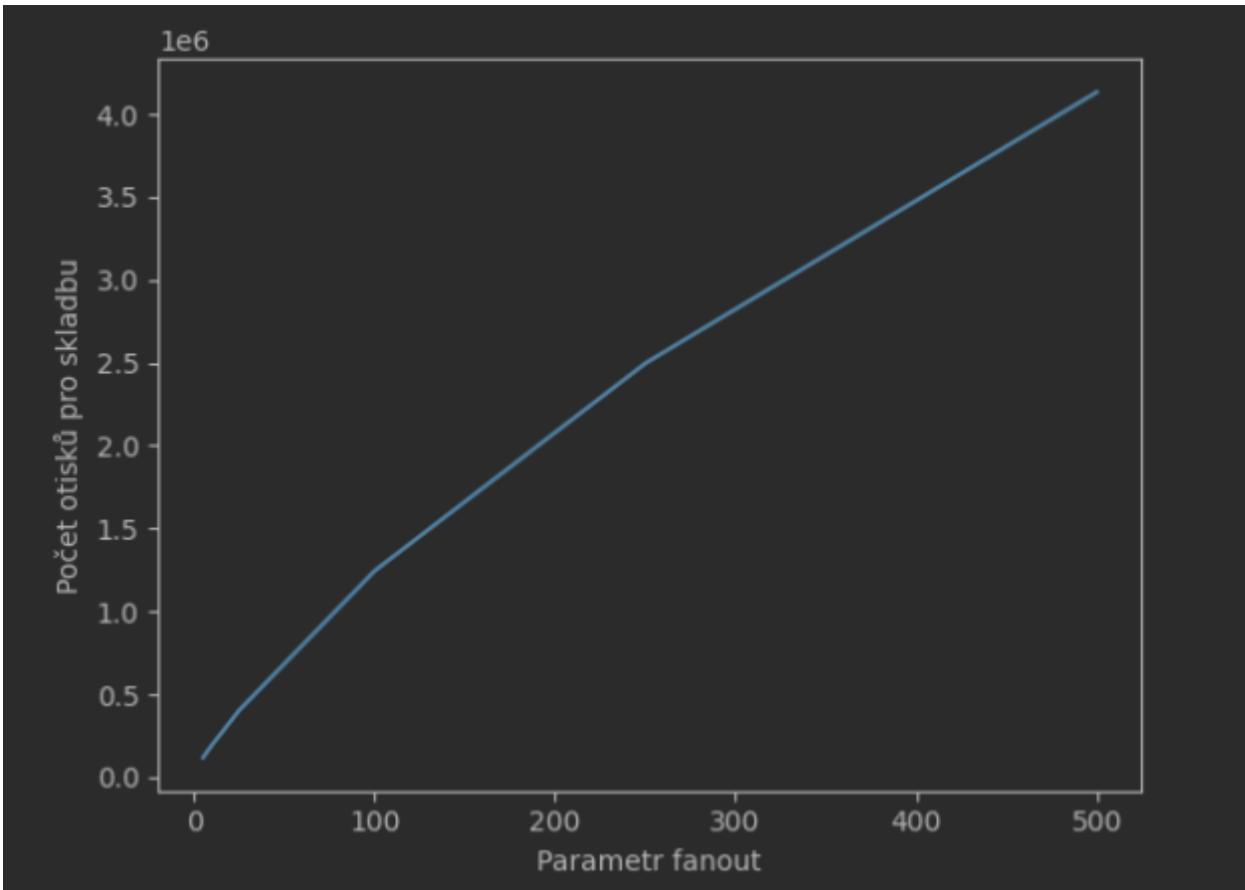


Na spektrogramu obou audio souborů můžeme pozorovat, že metoda `prominence` zachová více frekvencí ze spektrogramu než metoda `bands`, což ovšem nutně nevede k lepší procentuální shodě.

Fanout

Při experimentování s jednotlivými parametry algoritmu se ukázalo, že poměrně důležitý je parametr metody `fanout`, tedy hodnota o kterou se zvyšuje offset použitý při porovnávání otisků metody `fanout`.





Na uvedených grafech můžeme vidět že navýšení parametru `fanout` zvyšuje míru procentuální shody ale také množství vygenerovaných otisků pro skladby a tedy i dobu výpočtu a porovnání.

V praxi by tak jeho navýšení znamenalo neúnosnou zátěž na paměť. Rozhodli jsme se ho tedy ponechat na hodnotě 10. Už při této hodnotě máme při 205 skladbách v knihovně 47 milionů záznamů otisků v databázi.

Diskuze

Jako dataset jsme se rozhodli použít vlastní hudební knihovnu exportovanou ze služby Youtube Music. Kvůli nadměrné velikosti jsme ji však neukládali do repozitáře. Způsob přidání datasetu je tedy skrze lokální složky [/dataset](#). Do budoucna by bylo rozumné rozšířit administraci o možnost nahrání skladby přímo v prohlížeči.

Spouštění Python skriptů jsme zkoušeli původně implementovat jako mikroslužbu komunikující skrz [RabbitMQ](#) a [Celery](#) jobů, to se však ukázalo pro tento projekt zbytečně složité. Oddělením do vlastní mikroslužby dokážeme odstranit cold-start problémy, kdy Python musí načíst všechny závislosti do paměti.

Výsledná implementace ukládá posloupnost vybraných frekvencí jako řetězec oddělený čárkou, což může zabírat zbytečně více místa na disku, než je nutné pro zachování informací. V případě `n_fft = 2048` a `fanout` metody bude pro reprezentaci posloupnosti stačit 20 bitů.

Jako databázi jsme použili SQLite především kvůli přenositelnosti a jednoduchosti importu. S PostgreSQL a MySQL jsme narazili na problémy s nedostatkem paměti při hromadném nahrávání otisků. Finální databáze s 264 skladbami obsahuje přes 47 miliónů otisků.

Závěr

Aplikaci jsme vytvořili dle zadaných požadavků a v některých ohledech je i možná přesahujeme (např. možnost nahrání záznamu skrze mikrofon, hezký design). Výsledek projektu tedy hodnotíme velmi kladně.