Università degli Studi di Torino
Dipartimento di Informatica
Corso di Laurea in Informatica

Tesi di Laurea Triennale

Raccomandazione di contenuti musicali: un sistema intelligente basato sulla combinazione di concetti

RELATORE
Prof. Gian Luca Pozzato

CANDIDATO
Alberto Marocco
947841

Anno Accademico 2024/2025

Dichiarazione di Originalità

Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata.

Abstract

La seguente tesi di laurea presenta **DEGARI-Music**, un sistema di raccomandazione musicale basato sulla combinazione concettuale e sulla tipicità. I testi e le caratteristiche stilistiche dei brani, raccolti e arricchiti attraverso un crawler automatico di Genius.com, vengono preprocessati per estrarre feature lessicali e segnali di ripetizione; da essi si costruiscono prototipi di genere con proprietà **rigide** e **tipiche**. La combinazione HEAD/MODIFIER è realizzata con il framework **TCL** e il tool **CoCoS**, che generano scenari ponderati e prototipi ibridi impiegati per classificazione, ranking e spiegazioni. Un classificatore spiegabile, basato su «anchors» e soglie adattive, filtra e ordina i brani; le raccomandazioni risultano interpretabili sulla base dei tratti e dello scenario selezionato. L'approccio coniuga trasparenza, riproducibilità e potenziale estendibilità a multilingua, feature audio e apprendimento dei pesi.

Indice

1 Introduzione

La raccomandazione musicale è ormai comune nelle varie piattaforme di streaming, ma molti sistemi restano opachi, ossia suggeriscono contenuti ma senza chiarirne i criteri alla base delle loro scelte. In ambito accademico e industriale vi è un crescente interesse verso soluzioni content-based e spiegabili, in grado di operare anche in assenza di segnali utente e di giustificare le scelte in modo trasparente. Questa tesi adatta e applica il framework **DEGARI** *(tipicità + combinazione di concetti)* al dominio musicale, mostrando come prototipi e scenari possano guidare raccomandazioni interpretabili.

Questo lavoro presenta **DEGARI-Music**, un prototipo di sistema di raccomandazione che sfrutta la combinazione di concetti e la tipicità per costruire prototipi di generi (e ibridi *cross-genere*) a partire dai testi e caratteristiche stilistiche dei brani. L'approccio si fonda sul framework logico *TCL (Typicality-based Compositional Logic)* e sul tool *CoCoS*, adattati al dominio musicale. In sintesi, proprietà *rigide* e *tipiche* dei generi vengono estratte dai testi (Genius) e normalizzate; la logica seleziona scenari coerenti e non banali per la combinazione HEAD/MODIFIER; il risultato è un prototipo concettuale impiegato per riclassificare i brani e suggerire contenuti affini, con spiegazioni *ancorate* ai tratti ereditati e alla *forza* (punteggio) dello scenario selezionato.

1.a Obiettivi

Adattare il framework **DEGARI** al dominio musicale, proponendo un sistema *content-based* e spiegabile basato sui testi delle canzoni, che impiega la combinazione di concetti e la tipicità per costruire prototipi di genere e ibridi *cross-genere*.

Realizzare una pipeline riproducibile per l'estrazione dei testi e delle caratteristiche da **Genius**, il pre-processing linguistico e la normalizzazione dei dati lessicali necessari all'elaborazione logica.

Costruire prototipi di genere che rappresentino le proprietà **rigide** e **tipiche** di ciascun genere musicale, da utilizzare per classificazione e raccomandazione.

Fornire spiegazioni sintetiche e leggibili, basate su tratti lessicali e proprietà ereditate dai prototipi.

Validare il comportamento del sistema tramite verifiche automatiche e analisi qualitative su combinazioni di generi.

Rilasciare una pipeline modulare e integrabile con basi di conoscenza diverse.

1.b Contributi

Adattamento del framework DEGARI. Il lavoro estende l'uso del sistema originario, basato su **TCL** e **CoCoS**, dal dominio concettuale generale a quello musicale. Sono stati definiti formati di input compatibili con i testi delle canzoni e procedure di generazione dei prototipi.

Pipeline testi → prototipi di genere. Implementata una catena completa per la raccolta dei testi da **Genius**, la pulizia e lemmatizzazione, la selezione dei lemmi informativi e la costruzione dei prototipi di genere con proprietà **rigide** e **tipiche** pronte per l'elaborazione logica.

Pre-processing per CoCoS. Sviluppati gli script per la preparazione automatica delle coppie **HEAD/MODIFIER** e per la produzione dei file di input necessari alla combinazione concettuale, con gestione delle inclusioni rigide e dei conflitti di tipicità.

Generazione dei prototipi ibridi. Utilizzato **CoCoS** per combinare i prototipi di genere, ottenendo nuovi generi **cross-genere** caratterizzati da tratti coerenti e plausibili, successivamente riutilizzati per classificazione e raccomandazione.

Classificatore e sistema di raccomandazione. Implementato un modulo che impiega i prototipi generati per riclassificare i brani e produrre raccomandazioni spiegabili, basate sui tratti lessicali ereditati e sugli scenari selezionati dal framework logico.

Riproducibilità e validazione. Forniti script, configurazioni e documentazione per eseguire l'intera pipeline, con valutazioni automatiche e analisi qualitative su combinazioni di generi rappresentative.

1.c Metodologia in breve

Estrazione e pre-processing (Genius). Raccolta dei testi musicali tramite crawler su **Genius**; rimozione di **stopwords**, gestione delle forme flesse, filtraggio di lemmi poco informativi; lemmatizzazione e normalizzazione lessicale.

Generazione dei prototipi di genere. Per ciascun genere: conteggio dei lemmi caratteristici, assegnazione di punteggi di frequenza; applicazione di una

soglia di significatività; **rescaling** dei punteggi nel range compatibile con il framework **TCL**.

TCL / CoCoS – combinazione concettuale. Annotazione delle proprietà tipiche con grado p; generazione di scenari coerenti per ciascuna coppia **HEAD/MODIFIER**; selezione non banale con euristica HEAD preferenziale; le inclusioni risultanti assumono la forma $p :: T(C) \sqsubseteq D$.

Classificazione e ranking. Un brano è considerato compatibile con un prototipo se soddisfa i vincoli **rigidi** e possiede una sufficiente parte delle proprietà tipiche attive; il punteggio per il ranking deriva dall'allineamento tra i tratti del brano e quelli del prototipo, e viene utilizzato per ordinare raccomandazioni.

1.d Perimetro e limiti

Il sistema illustrato opera esclusivamente in ambito **content-based** sui testi: non utilizza (al momento) feature audio o metadati strutturati (anno, artista, popolarità). Questa scelta delimita il perimetro del lavoro, impedendo l'uso di segnali complementari che potrebbero migliorare il ranking.

Le assunzioni di indipendenza tra proprietà tipiche e la scelta delle soglie e della normalizzazione, pur compatibili con il framework teorico, possono introdurre distorsioni nel calcolo del punteggio finale e influenzare l'ordine delle raccomandazioni. In particolare, valori soglia troppo bassi possono generare ambiguità (inclusione di generi "vicini" ma non pertinenti), mentre soglie troppo elevate rischiano di escludere generi validi o far fallire l'assegnazione.

Aspetti linguistici avanzati, come polisemia ed espressioni multi-parola, sono gestiti in modo conservativo e non completamente disambiguati, il che può limitare la pertinenza delle proprietà estratte in contesti complessi.

Infine, il sistema è limitato dal punto di vista linguistico: l'estensione multilingua è prevista solo come sviluppo futuro; attualmente funziona solamente su contenuti in lingua inglese.

1.e Sintesi

DEGARI-Music dimostra che tipicità e combinazione concettuale possono supportare raccomandazioni musicali robuste e interpretabili. I prototipi di genere e gli ibridi **cross-genere** forniscono tratti leggibili e riutilizzabili lungo la pipeline (riclassificazione $\rightarrow$ ranking $\rightarrow$ spiegazioni). Il sistema è progettato per future estensioni (multilingua, audio, metadati) e valutazioni su scala con utenti reali, mantenendo la trasparenza come fondamento del metodo.

2 Fondamenti teorici: tipicità e combinazione di concetti (TCL)

2.a Tipicità e chiusura razionale

Le **Logiche Descrittive** (DL) forniscono un formalismo per rappresentare concetti, ruoli e individui. Nelle DL classiche (ad es. **ALC**) il ragionamento è **monotono**: aggiungere nuova informazione non invalida ciò che è già derivabile. Questo comportamento è inadeguato per modellare la conoscenza "di buon senso", spesso caratterizzata da regole generali con eccezioni.

Per modellare regolarità con eccezioni si introduce l'operatore di **tipicità** $T(c\cdot)$: un'assioma del tipo $T(C) \sqsubseteq D$ indica che "tipicamente i C sono D". Le inclusioni **rigide** conservano la forma classica $C \sqsubseteq D$.

La **rational closure** estende alle DL la nozione di chiusura razionale: i concetti vengono ordinati per grado di eccezionalità e si adottano **modelli minimi** che minimizzano i ranghi di anomalia (**rank**). In questo modo le inferenze che coinvolgono $T(c\cdot)$ risultano conservative ma flessibili, consentendo di sospendere premesse tipiche quando emergono informazioni più specifiche (**ereditarietà difettibile**).

2.a.I Notazione di base

- **Inclusioni rigide**: $C \sqsubseteq D$.
- **Inclusioni tipiche**: $T(C) \sqsubseteq D$.
- **Preferenza semantica**: si privilegiano interpretazioni che minimizzano l'eccezionalità (modelli minimi) e rispettano la gerarchia di specificità.

2.b TCL: Typicality-based Compositional Logic

TCL combina tipicità, probabilità e combinazione concettuale in stile cognitivo. Le proprietà tipiche sono annotate con un grado: $p :: T(C) \sqsubseteq D$ con $pin(0.5, 1]$, che si legge: "con grado p, i C tipici sono D".

Dato un **HEAD** C_H e un **MODIFIER** C_M, la combinazione non è la semplice intersezione, ma il risultato di una selezione coerente delle proprietà:

1. **Generazione di scenari**: si costruiscono insiemi compatibili di inclusioni tipiche (rispettando tutte le rigide).
2. **Valutazione**: ad ogni scenario si associa un punteggio/probabilità ottenuto combinando le annotazioni delle tipicità considerate attive (ipotesi d'indipendenza).
3. **Selezione**: si scelgono gli scenari ammessi e meglio valutati; le proprietà del risultato sono quelle presenti nello/gli scenario/i selezionato/i.

2.b.I Ruolo HEAD / MODIFIER

- L'**HEAD** fornisce la struttura concettuale di base; in caso di conflitti tra tipicità,hanno priorità le rigide e, tra le tipiche, si privilegiano scelte coerenti con la "fisionomia" dell'HEAD.
- Il **MODIFIER** introduce restrizioni/aggiustamenti che possono soppiantare tratti non essenziali dell'HEAD se ciò aumenta la coerenza globale dello scenario.

2.b.II Perché serve TCL (il "pet–fish")

La combinazione concettuale umana non è additiva: esistono congiunzioni in cui la tipicità del composto supera quella dei componenti (es. **pet fish**). Approcci puramente fuzzy o intersezioni rigide non catturano questi effetti; l'uso di tipicità difettibili, modelli minimi e scenari pesati consente invece di selezionare combinazioni plausibili senza contraddizioni.

2.c Semantica probabilistica delle tipicità

Le annotazioni p sulle inclusioni tipiche sono lette in modo "distribuzionale": uno scenario eredita un punteggio combinando (sotto ipotesi d'indipendenza) i pesi delle tipicità attive e penalizzando quelle escluse o in conflitto. Questa lettura permette di ordinare gli scenari e di spiegare perché alcune combinazioni risultano più plausibili di altre, sia qualitativamente sia quantitativamente.

2.d Il sistema CoCoS (cenni)

CoCoS implementa il calcolo di scenari di **TCL**. Dati **HEAD** e **MODIFIER** con le rispettive proprietà rigide e tipiche, costruisce gli scenari ammissibili, ne calcola il punteggio e restituisce il/i prototipo/i del concetto composto come **insieme di proprietà con il relativo grado**, insieme alle informazioni sullo scenario selezionato. (Questa sezione introduce l'idea operativa; i dettagli implementativi saranno trattati nei capitoli successivi.)

3 Probabilità e strumenti per la combinazione concettuale (TCL e CoCoS)

In questo capitolo approfondiamo (i) la componente probabilistica di **TCL** e (ii) il tool **CoCoS** che implementa la combinazione concettuale basata su tipicità, presentando anche sistemi affini che lo riutilizzano.

3.a Inclusioni probabilistiche e scenari in TCL

In TCL le proprietà tipiche sono annotate con un grado di credenza: $p :: T(C) \sqsubseteq D$ con $pin(0.5, 1]$. L'idea è che le tipicità valgano "per i C normali", ma possano essere sospese di fronte a informazione più specifica. La semantica probabilistica (stile **DISPONTE**) assume indipendenza fra tipicità e definisce una distribuzione sugli **scenari**, cioè sulle scelte di quali inclusioni tipiche sono attive.

Definizione operativa. Data una base con tipicità $p_i :: T(C_i) \sqsubseteq D_i$:

- uno **scenario** è un sottoinsieme consistente delle tipicità;
- allo scenario si associa un punteggio/probabilità ottenuto combinando i p_i (attive) e $(1 - p_i)$ (escluse); in termini generali, il punteggio di uno scenario si ottiene come prodotto dei gradi p_i delle tipicità attive e dei complementi $(1 - p_i)$ di quelle escluse.
- si selezionano gli scenari **ammessi** dalle rigide e **coerenti** con l'euristica HEAD/MODIFIER; il/i prototipo/i del concetto composto eredita/no le proprietà presenti nello/gli scenario/i selezionato/i.

Questa lettura si integra con la **rational closure** di **ALC+T_R**: gli scenari sono valutati solo sui modelli che minimizzano l'eccezionalità (modelli minimi), preservando specificità e irrilevanza.

3.b CoCoS: input, algoritmo, output

Scopo. CoCoS implementa **TCL**: dato un concetto **HEAD** C_H e un **MODIFIER** C_M, costruisce gli scenari ammissibili e restituisce il prototipo del concetto composto $C_H l \wedge C_M$.

Input.

- **Proprietà rigide** di C_H e C_M (vincoli non derogabili).
- **Proprietà tipiche** con grado p per ciascun concetto.
- (Opzionale) limiti al numero massimo di proprietà da ereditare.

Algoritmo (sketch).

1. **Genera scenari**: combina le tipicità in insiemi consistenti con le rigide.
2. **Valuta**: assegna a ogni scenario un punteggio/probabilità (ipotesi di indipendenza).
3. **Filtra HEAD/MODIFIER**: scarta scenari incompatibili con la dominanza dell'HEAD.
4. **Seleziona**: sceglie lo/gli scenario/i migliore/i; produce un **insieme di proprietà con il relativo grado** per il prototipo composito e annota il punteggio dello scenario.

Output. Un prototipo composto (insieme di proprietà con grado) e le informazioni sullo scenario selezionato.

3.c Estensioni e contesto d'uso

Il formalismo **TCL** e il tool **CoCoS** possono essere riutilizzati in domini diversi da quello musicale, ogni volta che è utile combinare concetti o categorie basate su proprietà tipiche. In questa tesi vengono applicati come motore di combinazione per la costruzione di prototipi di genere e ibridi **cross-genere**.

3.d Collocazione nella pipeline di DEGARI-Music

Nei capitoli successivi **TCL/CoCoS** è usato come "motore" di combinazione: dopo la costruzione dei prototipi di base (proprietà rigide/tipiche), combineremo **HEAD/MODIFIER** per ottenere concetti ibridi; questi sprototipi composti alimenteranno classificazione e ranking.

4 Estrazione e pre-processing dei dati (Genius)

In questo capitolo si descrivono la raccolta e la preparazione dei dati testuali (testi e metadati) da **Genius**, l'uso del client Python `lyricsgenius` per l'accesso all'API e uno script di **enrichment** per stimare indici di ripetizione e derivare tag ausiliari.

4.a Accesso all'API Genius e gestione del token

Per il recupero o l'integrazione dei testi si utilizza `lyricsgenius`, un client Python che interfaccia la **Developer API** (`api.genius.com`) e la **Public API**. L'accesso autenticato richiede un **access token** impostato come variabile d'ambiente `GENIUS_TOKEN` (token non versionato nel codice). In fase di istanziazione (`Genius(...)`) sono configurati timeout e rate-limit; il codice gestisce retry su errori temporanei (ad es. HTTP 429) ed evita duplicati per brano/ID.

4.b Raccolta dei brani: criteri e formato di output

La raccolta è guidata da una lista di brani per genere (rap, metal, rock, pop, trap, reggae, rnb, country). Per ciascun brano si persiste, quando disponibile, un record nel JSON unico `descr_music_GENIUS.json` con i campi principali: { «ID»: «rap_eminem_rap-god_2013_235729», «genius_id»: 235729, «source»: «genius», «source_url»: «https://genius.com/Eminem-rap-god-lyrics», «title»: «Rap God», «artist»: «Eminem», «album»: «Curtain Call 2», «year»: «2013», «lyrics»: «Testo integrale …», «tags»: [«high_repetition», «rap»], «moods»: [ ], «instruments»: [ ], «subgenres»: [«rap»], «contexts»: [ ], «repetition»: { «rep_ratio»: 0.576, «has_chorus_like»: 0, «has_hook_like»: 0 } }

I campi `title/artist/album/year` provengono da **Genius**; `lyrics` contiene il testo integrale (se presente). La chiave `tags` è usata anche per i segnali di ripetizione derivati nel pre-processing (sezione seguente). Sono previsti controlli

di **deduplicazione** per `genius_id` e normalizzazione di `title/artist` (minuscolizzazione, trimming) per gestire alias e varianti.

4.c Enrichment della ripetizione e tag derivati

Lo script `enrich_repetition.py` elabora il testo e aggiunge al record un blocco `repetition` con:

- **Indice di ripetizione** `rep_ratio`, compreso tra 0 e 1, calcolato come **1 meno il rapporto tra il numero di lemmi distinti e il numero totale di lemmi** dopo normalizzazione del testo.
- **n-gram più frequenti** (`top_terms`, `top_bigrams`, `top_trigrams`).
- **Flag euristici** `has_chorus_like` / `has_hook_like`, basati su densità di n-gram e su eventuali etichette testuali come `[Chorus]` o `[Hook]`.

In base a questi indici si arricchisce `tags` con:

- `high_repetition` (se `rep_ratio ≥ 0.25`);
- `catchy_chorus` e/o `hook_repetition` quando i flag sono attivi.

La soglia `0.25` è scelta in modo conservativo dopo ispezioni qualitative su più generi: valori inferiori generano troppi falsi positivi su testi prolissi, mentre valori troppo alti escludono casi con ritornelli brevi. L'arricchimento è **idempotente** e il JSON viene riscritto con **commit atomico** (scrittura temporanea e rinomina del file finale). Questi segnali alimentano le **proprietà tipiche** utilizzate da **TCL/CoCoS** nei capitoli successivi.

4.d Nota sulla variante "extended"

È stata sperimentata una variante **extended** che, a partire dai prototipi per brano e dai profili **typical/rigid**, genera per ogni canzone un JSON dedicato (piano di struttura/strumentazione, focus lirico, **lyrics**). Per semplicità di integrazione nella pipeline, nella versione finale è stato adottato il JSON unico; la variante è mantenuta come utilità per analisi qualitative.

5 Creazione dei prototipi (Modulo 1)

In questo capitolo si descrive come, a partire dal JSON `descr_music_GENIUS.json`, vengono generati i prototipi testuali per ciascun brano. Il modulo produce un file `.txt` per ogni istanza (artwork/brano) con coppie `parola: punteggio` normalizzate nell'intervallo `[0.6, 0.9]`, compatibile con il range atteso dalle proprietà **tipiche** in **TCL**.

5.a Input e obiettivo

Input. Lista di record con campi minimi: `ID`, `title`, `artist`, `album`, `year`, `lyrics`, `tags`, `moods`, `instruments`, `subgenres`, `contexts`. **Output.** Per ogni istanza `ID`, un file `/<cartella>/<ID>.txt` con feature lessicali pesate, destinato ai moduli successivi (**CoCoS** e Recommender).

5.b Pre-processing linguistico

- **Tokenizzazione.** `nltk.tokenize.word_tokenize` (Treebank + Punkt).
- **Stopword.** Rimozione delle stopword inglesi (corpus NLTK).
- **Lemmatizzazione & POS.** Quando disponibile, **TreeTagger** via `treetaggerwrapper` (lemma + POS); in fallback, lowercasing senza lemma.
- **Regola leggera di co-occorrenza.** Se in bigramma compare un **verbo (lemma)** seguito da un **sostantivo/parola di contenuto**, oltre al lemma del sostantivo si incrementa anche il conteggio del verbo (cattura associazioni **verbo → tema**).

5.c Estrazione e pesatura delle feature

Per ogni istanza si costruisce una "descrizione" concatenando i campi configurati (`title`, `artist`, `lyrics`, `tags`, …); il testo è tokenizzato/lemmatizzato e si contano le occorrenze grezze. I conteggi sono trasformati in punteggi tramite **normalizzazione lineare** sull'intervallo `[0.6, 0.9]`:

$$\text{score} = 0.6 + 0.3 * \frac{\text{freq} - \text{min\_freq}}{\text{max\_freq} - \text{min\_freq}} \tag{1}$$

dove

$$\text{freq} = \frac{\text{count}(w)}{\text{tot\_words}} \tag{2}$$

. Se

$$\text{max\_freq} = \text{min\_freq} \tag{3}$$

, si assegna

$$\text{score} = 0.9 \tag{4}$$

(tutti i termini al valore massimo nel caso degenerato).

5.d Formato dell'output

Un file per istanza, una riga per parola (spaziatura allineata per leggibilità): rap: 0.645 lookin: 0.627 god: 0.624 feel: 0.615 …

L'`ID` dell'istanza determina il nome del file (sanificato per il filesystem).

5.e Esempio sintetico

Estratto (rap, **Rap God**): `rap: 0.645 · lookin: 0.627 · god: 0.624 · feel: 0.615 · ...` (La lista completa è nel file generato in `music_for_cocos`.)

5.f Considerazioni implementative

- **Robustezza.** Se **TreeTagger** non è disponibile, il modulo prosegue con tokenizzazione + stopword (senza lemma/POS).
- **Pulizia.** Normalizzazione `title/artist` (minuscolizzazione, trimming) e deduplicazione per `genius_id`.
- **Riproducibilità.** Parametri e soglie sono fissati nel codice; i file sono scritti in modo idempotente.

6 Combinazione di generi musicali con CoCoS (Modulo 2)

In questo capitolo si descrive come, a partire dalle proprietà **rigide** e **tipiche** per macro-genere, **CoCoS** costruisce scenari di combinazione **HEAD/MODIFIER** e seleziona i prototipi dei concetti ibridi (es. pop–rap).

6.a Input, preprocessing e formato dei file

Input. Le directory `typical/` e `rigid/` prodotte dal modulo di costruzione dei profili (cap. 8: **BuildTypicalRigid**), più la cartella dei file `.txt` per ciascuna coppia `H_M` in `prototipi_music/` (creata da `cocos_preprocessing.py`).

Ogni file `H_M.txt` contiene:

- intestazione con **Head Concept Name** e **Modifier Concept Name**;
- elenco delle **rigide** di **head** e **modifier**;
- inclusioni **tipiche** annotate come $p :: T(C) \sqsubseteq D$ con $p \in (0.5, 1]$.

Esempio sintetico (pop–rap). Head Concept Name: pop Modifier Concept Name: rap

head, pop head, high_repetition

modifier, high_repetition

T(head), pop, 0.95 T(head), high_repetition, 0.629 T(head), hook_repetition, 0.629 T(head), catchy_chorus, 0.6

T(modifier), high_repetition, 0.8

6.b Vincoli e ruolo HEAD/MODIFIER

- Le **rigide** di entrambi i concetti devono sempre essere rispettate.
- Tra le **tipiche**, **CoCoS** privilegia scenari coerenti con la fisionomia dell'**HEAD**; il **MODIFIER** introduce aggiustamenti compatibili (cfr. TCL nei capp. 2–3).
- Un limite al numero di tipiche selezionabili (`MAX_ATTRS` in `cocos_config.py`, default 2) evita combinazioni "sovraccariche".

6.c Generazione e valutazione degli scenari

Dalle tipicità disponibili si costruiscono sottoinsiemi ammissibili (rispetto a rigide e al limite `MAX_ATTRS`). A ciascuno scenario si associa uno **score** in stile **DISPONTE** (indipendenza dei pesi tipici): lo **score** è il **prodotto** dei gradi p_i delle tipicità attive (oppure un'eventuale trasformazione monotona equivalente implementata nel codice per stabilità numerica). Gli scenari sono ordinati per **score**; in caso di parità si mantengono più **best**.

6.d Selezione e output

Per ciascun file `H_M.txt`, **CoCoS**:

- stampa a console gli scenari consigliati;
- appende al file due righe riassuntive:

Result: {«pop»: 0.95, «high_repetition»: 0.80, «hook_repetition»: 0.629, «@ scenario_score»: …} Scenario: [1, 1, 0, 1, 0, …]

(Opzionale) Con l'opzione `-o` salva un JSON pulito in `prototipi_music/scenarios_json/` con tutti gli **best**. Esempio di risultato (**pop–rap**):

```
Result: {"pop": 0.95, "high_repetition": 0.8, "hook_repetition": 0.629,
"@scenario_probability": 7.0941136, "@scenario_score": 7.0941136}
Scenario: [1, 1, 0, 1, 0, 7.0941136]
```

6.e Parametri rilevanti

- `MAX_ATTRS` (default 2): massimo numero di tipiche ereditabili.
- **CLI (`cocos.py`):**
  - esecuzione su tutta la cartella configurata: `python cocos.py`
  - esecuzione su una coppia specifica: `python cocos.py path\H_M.txt 3 -o path\scenarios_json`

6.f Casi senza scenario

Se nessuno scenario supera i vincoli (rigide, coerenza **HEAD/MODIFIER**, limite attributi), **CoCoS** segnala **"NO recommended scenarios!"** e non modifica il file `H_M.txt`. Il comportamento è utile per individuare coppie poco informative o profili troppo scarsi.

6.g Collegamento ai moduli successivi

I prototipi compositi (insieme di proprietà con il relativo grado + **scenario score**) alimentano il classificatore e il recommender: sono riutilizzati per ran-

king e spiegazioni, mantenendo trasparenza sui tratti ereditati e sullo scenario selezionato.

7 Preprocessing CoCoS: costruzione dei file H_M (Modulo 3a)

In questo capitolo si descrive il preprocessing che, a partire dai profili per genere (cartelle `typical/` e `rigid/` prodotte dal Modulo 2), costruisce i file di input per **CoCoS** in `prototipi_music/`, uno per ogni coppia **Head–Modifier** (`H_M.txt`).

7.a Scopo, input e percorsi

Script. `Sistema di raccomandazione/cocos_preprocessing.py` **Config.** `cocos_config.py` (percorsi `TYPICAL_PROP_DIR`, `RIGID_PROP_DIR`, `COCOS_DIR`).

Input.

- `typical/<genere>.txt`: righe nel formato `prop: peso` (es. `high_repetition: 0.80`).
- `rigid/<genere>.txt`: una proprietà per riga (vincoli non derogabili).

Output.

- `prototipi_music/H_M.txt` con:
  - intestazione (titoli **Head/Modifier**);
  - rigide di **head** e **modifier**;
  - tipiche annotate come `T(head), <prop>, <p>` e `T(modifier), <prop>, <p>`.

7.b Algoritmo (`cocos_preprocessing.py`)

1. Legge i file `rigid` e `typical` per **HEAD** e **MODIFIER**.
2. Scrive `H_M.txt` in `COCOS_DIR` con il seguente ordine:
   - intestazione: `Title`, `Head Concept Name`, `Modifier Concept Name`;
   - blocco **rigid** dell'**head** (`head, <prop>`), poi del **modifier** (`modifier, <prop>`);
   - blocco **typical** del **modifier** (`T(modifier), <prop>, <peso>`);
   - blocco **typical** dell'**head** (`T(head), <prop>, <peso>`).

L'ordine `rigid → T(modifier) → T(head)` è coerente con il ruolo **HEAD/MODIFIER** usato da **CoCoS** nel modulo successivo.

7.c Esecuzione

- **Singola coppia:** `python cocos_preprocessing.py <HEAD> <MODIFIER>`
- **Batch su tutte le coppie (H $\neq$ M) – PowerShell:**

```
genres = @("rap","metal","rock","pop","trap","reggae","rnb","country")
foreach (h in genres) {
  foreach (m in genres) {
    if (h -ne m) {
      python cocos_preprocessing.py h m
    }
  }
}
```

7.d Formato del file `H_M.txt` (esempi)

country–metal Title: country-metal Head Concept Name: country Modifier Concept Name: metal

head, country head, high_repetition

modifier, metal modifier, high_repetition

T(modifier), metal, 0.95 T(modifier), high_repetition, 0.6

T(head), country, 0.95 T(head), high_repetition, 0.6

metal–country (simmetrico, ruoli invertiti) Title: metal-country Head Concept Name: metal Modifier Concept Name: country

head, metal head, high_repetition

modifier, country modifier, high_repetition

T(modifier), country, 0.95 T(modifier), high_repetition, 0.6

T(head), metal, 0.95 T(head), high_repetition, 0.6

7.e Note pratiche

- Le **rigid** sono riportate come vincoli duri e saranno sempre rispettate da **CoCoS**.
- I pesi delle **typical** sono copiati dai file di genere (range tipico `[0.6, 0.95]`).
- È utile generare sia `H_M.txt` sia `M_H.txt`: l'esito dipende dal ruolo **Head/Modifier**.
- I percorsi sono centralizzati in `cocos_config.py` (es. `COCOS_DIR` per la destinazione dei file).

7.f Collegamento al Modulo 3b

I file `H_M.txt` prodotti qui sono consumati da `cocos.py`, che costruisce gli scenari di combinazione, seleziona i **best** e li appende al file (oltre a generare, se richiesto, un JSON con gli scenari raccomandati).

8 CoCoS: combinazione e scenari (Modulo 3b)

In questo capitolo descriviamo l'esecuzione di **CoCoS** sul set di coppie **HEAD/MODIFIER** generato dal preprocessing. L'obiettivo è selezionare gli scenari migliori e scrivere il prototipo del concetto ibrido (es. **rock–trap**).

8.a Input e avvio

Input. Per ogni coppia **H_M** è presente un file `prototipi_music/H_M.txt` con:

intestazione: **Head Concept Name**, **Modifier Concept Name**;

rigide di **head** e **modifier** (provenienti da `rigid/`);

tipiche annotate come righe del tipo `T(head|modifier), proprietà, p` con **p** in $(0.5, 1]$ (provenienti da `typical/`).

CLI. Esecuzione batch (esempio usato nel progetto):

```
python .\cocos.py "<BASE>\prototipi_music\H_M.txt" 3 -o
"<BASE>\prototipi_music\scenarios_json"
```

Il secondo argomento (qui 3) sovrascrive il default e impone il massimo numero di tipiche ereditabili.

8.b Generazione e scoring degli scenari

Dalle tipiche disponibili si costruiscono sottoinsiemi ammissibili (nel rispetto delle **rigid** e del limite sugli attributi). A ciascuno scenario si assegna uno **score** in stile **DISPONTE**: lo **score** è calcolato come prodotto dei gradi **p** delle tipicità attive (assunzione di indipendenza). Gli scenari sono ordinati per **score**; in caso di parità si mantengono più **best**.

8.c Selezione e output

Per ogni file `H_M.txt`:

a console vengono stampati gli scenari raccomandati;

nel file stesso si appendono due righe riepilogative:

Result: {"rock": 0.95, "trap": 0.95, "hook_repetition": 0.60, "@scenario_score": 3.4656} Scenario: [1, 1, 0, 1, 0, 3.4656]

Con l'opzione `-o` si salva anche un JSON in `prototipi_music/scenarios_json/` con tutti i **best**.

Esempio. Per **rock–trap** uno scenario raccomandato può essere: Result: {"rock": 0.95, "trap": 0.95, "hook_repetition": 0.60, "@scenario_score": 3.4656}

Per coppie poco informative o con profili scarsi si può ottenere: **NO recommended scenarios!**

8.d Parametri pratici

MAX_ATTRS. Numero massimo di tipiche ereditabili (default nel config, sovrascrivibile da CLI). **Coerenza HEAD/MODIFIER.** Le **rigid** di entrambi si rispettano sempre; tra le **typical** si privilegiano combinazioni coerenti con la fisionomia dell'**HEAD**, lasciando al **MODIFIER** aggiustamenti compatibili. **Fall-back.** Se nessuno scenario supera i vincoli, il file non viene modificato: questo aiuta a individuare coppie o profili da rivedere.

8.e Output per i moduli successivi

I **Result** (insieme di proprietà con relativo grado + **score** di scenario) sono l'input del classificatore e del recommender: forniscono sia i tratti ereditati sia la **forza** dello scenario selezionato, riutilizzata per **ranking** e spiegazioni.

9 Sistema di raccomandazione (Modulo 4)

Il modulo finale prende i prototipi ibridi generati da **CoCoS** e filtra/ordina i brani del dataset originale, producendo per ogni coppia **HEAD_MODIFIER** un file JSON con gli item consigliati e l'evidenza delle proprietà soddisfatte.

9.a Input e panoramica

Input.

I file `prototipi_music/H_M.txt` arricchiti da **CoCoS** (rigide, tipiche e righe finali `Result:` e `Scenario:`).

Il JSON sorgente dei brani configurato in `Recommender_config.py` (stessi campi usati nei moduli precedenti).

Output. Per ogni `H_M.txt` viene salvato un `H_M_recommendations.json` con: categoria, prototipo attivo, ancore, lista dei risultati id–titolo–artista–matches–anchors_hit, numero di brani classificati su totale. Gli output vengono poi spostati in `prototipi_music/recommender_out/`.

9.b Dal file CoCoS al prototipo "pulito"

Il classificatore legge `Result:` e l'array `Scenario:` del file `H_M.txt` e costruisce il prototipo attivo:

- seleziona le tipiche con bit 1 nello scenario;
- include le ancore (le rigide del file e, più in generale, le proprietà marcate obbligate nella sezione iniziale);
- rimuove duplicati e normalizza i nomi.

Il risultato è una lista di proprietà da soddisfare + l'insieme delle ancore richieste.

9.c Regole di matching

Per ogni brano del JSON sorgente il sistema:

- cerca ogni proprietà del prototipo nei campi configurati in `Recommender_config.py` (titolo + campi descrittivi);

- opzionalmente scarta il brano se contiene proprietà negate (prefisso `-` nel file prototipo);
- accetta il brano se sono vere entrambe le soglie:
  - `match-rate` minimo (default 0.15 delle proprietà del prototipo),
  - minimo numero di ancore colpite (default 1).

Le soglie sono modificabili da CLI: `--min-match-rate`, `--min-anchors` (e `--max-print` per limitare le righe stampate a console).

9.d Formato dell'output

Ogni JSON contiene:

- `category` (nome del file prototipo, es. `head_modifier.txt`),
- `prototype` (lista piatta delle proprietà del prototipo attivo),
- `anchors` (ancore richieste),
- `results` (array di oggetti { id, title, artist, matches, anchors_hit }),
- `classified` e `total`.

Esempio sintetico (rock_pop):

```
{
"category": "rock_pop.txt",
"prototype":
["rock","hook_repetition","catchy_chorus","pop","high_repetition"],
"anchors":
["catchy_chorus","rock","high_repetition","pop","hook_repetition"],
"results": [
{ "id": "...rap-god...", "title": "Rap God", "artist": "Eminem",
"matches": ["high_repetition"], "anchors_hit": ["high_repetition"] }
],
"classified": 48, "total": 48
}
```

9.e Uso a riga di comando e integrazione nella pipeline

Singolo prototipo.

```
python Sistema di raccomandazione/Classificatore/Recommender.py
prototipi_music\H_M.txt
```

Batch su tutte le coppie. Lo script PowerShell del progetto itera su tutti i `H_M.txt`, invoca il classificatore e sposta gli output nella cartella `recommender_out/`.

9.f Considerazioni e limiti

Spiegabilità. Ogni suggerimento riporta le proprietà che hanno fatto match e quali ancore sono state colpite.

Ranking. L'attuale versione filtra per soglie e non ordina con uno score continuo; in prospettiva si può introdurre un ranking per numero/`peso` di proprietà soddisfatte (es. riusando i pesi tipici di **CoCoS**).

Parametri. Le soglie di copertura e ancore permettono di rendere il sistema più `severo` o `inclusivo` senza modificare i prototipi di partenza.

Coerenza. Se **CoCoS** non aveva prodotto scenari per una coppia, non esiste un `Result:` valido e non viene generato alcun JSON di raccomandazioni per quella categoria.

9.g Collegamento ai capitoli successivi

I JSON di raccomandazione alimentano sia i **Risultati** sia le **Spiegazioni**, dove analizziamo copertura, varietà e casi tipici di match per le combinazioni testate.

10 Risultati

In questo capitolo sintetizziamo gli esiti della pipeline: prototipi compositi → classificatore/raccomandatore per ciascuna coppia HEAD/MODIFIER. Gli output sono file **recommendations.json che, per ogni coppia, riportano: categoria, prototipo (proprietà selezionate), ancore rigide/forti, lista dei brani consigliati con le proprietà effettivamente colpite, e contatori classified/total.**

10.a Metriche e formato

Ogni file espone:

prototype: proprietà del concetto ibrido (es. [«trap»,«high_repetition»,«metal»]).

anchors: sottoinsieme di proprietà chiave usate come vincoli/ancore.

results: brani con matches (proprietà colpite) e anchors_hit (ancore soddisfatte).

classified / total: copertura della classificazione per la coppia.

Esempio (estratto trap–metal): prototype = [«trap»,«high_repetition»,«metal»], classified = 48, total = 48.

10.b Copertura

Sulle coppie analizzate, la copertura è piena in molti casi (classified = total = 48, es. trap–metal, rap–rnb). In alcuni accoppiamenti con profili più scarsi, la copertura può ridursi (casi 79% osservati a log su rock–reggae), coerente con l'assenza di scenari o proprietà insufficienti nel prototipo.

10.c Coerenza dei suggerimenti con il prototipo

Trap–Metal. Il prototipo ibrido punta su trap, metal, high_repetition. Nei risultati:

tracce trap (Travis Scott, Future) colpiscono trap + high_repetition e soddisfano entrambe le ancore quando presenti (anchors_hit: trap, high_repetition);

classici metal (Metallica, Judas Priest) colpiscono metal + high_repetition;

brani di altri generi appaiono quando soddisfano comunque l'ancora di ripetizione (es. Weeknd, AC/DC, Marley), a conferma del ruolo trasversale di high_repetition.

Rap–R&B. Il prototipo enfatizza rnb + high_repetition. I suggerimenti includono Beyoncé e TLC (hit di rnb + high_repetition) e, per la componente di ripetizione, anche rap/pop/rock con forte hook (Eminem, Weeknd, AC/DC) quando soddisfano l'ancora.

10.d Interpretazione

Ancore forti funzionano bene. Dove il prototipo contiene proprietà altamente distintive (es. trap, metal, rnb) la lista privilegia brani emblematici dei generi coinvolti, mantenendo diversità sul terzo asse (high_repetition).

Segnali trasversali spiegano intersezioni ampie. high_repetition porta nella top anche brani di altri generi con forte chorus/hook: spiegazione coerente con i tag generati nell'enrichment.

10.e Esempi puntuali

Trap–Metal → "SICKO MODE" (Travis Scott). matches = [«high_repetition»,«trap»], anchors_hit = [«high_repetition»,«trap»]. Metal → "Enter Sandman" (Metallica). matches = [«high_repetition»,«metal»], anchors_hit = [«high_repetition»,«metal»].

Rap–R&B → "Drunk in Love" (Beyoncé). matches = [«high_repetition»,«rnb»], anchors_hit = [«high_repetition»,«rnb»]. Rap–R&B → "Rap God" (Eminem). matches = [«high_repetition»], richiamato dalla forte ripetizione anche se non rnb.

10.f Limiti osservati

Dipendenza da high_repetition. Essendo un segnale orizzontale, può allargare troppo la platea se il prototipo non contiene altre tipiche/rigide selettive; l'effetto è voluto per esplorare cross-over, ma va bilanciato in fase di presentazione.

Copertura non uniforme. Dove i profili di genere sono scarsi (poche tipiche/rigide) la selezione può risultare vuota o parziale (casi "NO scenario" in CoCoS e coperture $< 100\%$ a valle).

10.g Takeaway

Il raccomandatore preserva le scelte di CoCoS: le ancore del prototipo ibrido guidano effettivamente i suggerimenti.

I tag di ripetizione arricchiti su Genius si riflettono nei risultati, favorendo brani con hook/chorus marcati anche fuori dal macro-genere dell'HEAD/ MODIFIER.

Con profili più ricchi (più tipiche non trasversali) ci si attende una maggiore precisione semantica e minore dipendenza da high_repetition.

11 Discussione

Inquadriamo i risultati alla luce della pipeline (prototipi → CoCoS → raccomandatore), discutendo criticità, confronto con approcci affini e implicazioni d'uso.

11.a Dove il sistema fallisce (e perché)

Propagazione degli errori. Ogni modulo erede i vincoli del precedente: se i profili typical/rigid sono scarsi o sbilanciati, CoCoS genera pochi scenari o nessuno; il raccomandatore, a cascata, ha copertura ridotta o suggerimenti troppo generici.

Dominanza di segnali trasversali. Tag orizzontali (es. high_repetition) migliorano la copertura ma, se gli altri tratti sono deboli, allargano troppo la platea e "schiacciano" la specificità di genere. Rimedi pratici:

aumentare topk_typical solo se esistono tratti non-trasversali;

alzare la penalità dei "globaloni" (COMMON_PENALTY) o innalzare le soglie typical_thr;

usare MAX_ATTRS ≥ 2 ma con almeno una tipica fortemente distintiva dell'HEAD.

Assunzioni di indipendenza. Lo score degli scenari moltiplica pesi tipici come se fossero indipendenti. Quando due proprietà sono correlate (es. trap e 808), lo scenario può sovrastimarle. Possibili fix: regole di mutua esclusione/coesione o "gruppi" di feature con peso congiunto.

Heuristics Head/Modifier. La priorità semantica all'HEAD è utile ma rigida: alcuni mix (es. pop↔rnb) potrebbero richiedere simmetria o switch dinamico del ruolo. Si può introdurre un parametro di "plasticità" o una scelta H/M guidata dai punteggi.

Rumore testuale. Tokenizzazione/stopword inglesi e qualità variabile dei testi di Genius introducono:

sinonimia superficiale (manca normalizzazione a lemmi se TreeTagger non è disponibile);

lessico di dominio non catturato se fuori whitelist;

mappe SUB2MACRO incomplete su tag rari. Mitigazioni: ampliare DOMAIN_WHITELIST, raffinando SUB2MACRO e (se disponibile) usare lemmatizzatori robusti.

11.b Confronto con approcci affini

Baselines "bag-of-words" o tf–idf. Offrono buone similitudini ma non distinguono tra tipicità (difettibile) e vincoli rigidi; mancano ruoli HEAD/MODIFIER e non motivano perché certe proprietà "saltano" o vengono sospese.

Collaborative filtering / embedding neurali. Predicono bene preferenze e vicinanze, ma la spiegabilità è bassa e la combinazione concettuale richiede hack (intersezione di vicini). Qui, invece, il prototipo ibrido è esplicito e auditabile, con scenari e pesi tracciabili.

TCL/CoCoS. Rispetto a semplici regole logiche o fuzzy, aggiunge: (i) priorità gerarchica e sospendibilità delle tipiche, (ii) selezione per scenari con punteggi, (iii) ruolo H/M. Il trade-off è una maggiore sensibilità alla qualità dei profili e all'ipotesi d'indipendenza dei pesi.

11.c Implicazioni pratiche

Trasparenza e controllo. Ogni suggerimento eredita ancore e matches del prototipo: l'utente (o il curatore) può vedere quali tratti hanno governato il ranking e regolare soglie/whitelist per pilotare le proposte.

Discovery di crossover. La presenza di segnali trasversali (hook, ripetizione) fa emergere brani di generi "lontani" ma plausibili rispetto al concept ibrido. È utile per playlist tematiche, format radio e ideazione creativa (moodboard sonori).

Cura dei profili. Il sistema incentiva la manutenzione leggera dei dizionari: aggiungere 2–3 tipiche distintive per genere migliora molto gli scenari; la rigidità va usata con parsimonia (ancore poche ma forti).

11.d Minacce alla validità

Copertura dati. Il campione è limitato; pochi esempi per un genere riducono la stabilità dei typical/rigid.

Bias di sorgente. Testi/metadata di Genius riflettono cataloghi e pratiche editoriali specifiche.

Scelte di iperparametri. MIN_W/MAX_W, soglie e MAX_ATTRS influenzano direttamente gli scenari; risultati diversi con settaggi diversi.

11.e Cosa migliorare subito

Rinforzare la specificità. Aumentare DISTINCTIVE_BOOST o arricchire i profili con 2–3 tratti non-trasversali per genere (riduce l'effetto calamita di high_repetition).

Regole di coerenza. Piccolo set di vincoli: se trap allora preferisci 808; se reggae evita double_kick.

Selezione scenari soft. Oltre al best, tenere top-k scenari e far decidere al raccomandatore con un rimescolamento pesato (diversifica le playlist).

Diagnostica di copertura. Report automatico: brani non classificati per coppia, proprietà mai attivate, rigid che azzerano gli scenari.

Arricchimento linguistico. Estendere whitelist e mappature SUB2MACRO; dove possibile, lemmatizzazione stabile per ridurre la frammentazione del lessico.

11.f Takeaway

Il paradigma prototipi + combinazione fornisce spiegazioni locali e controllabilità globale con pochissimi iperparametri.

La qualità dei profili typical/rigid è la leva principale: quando sono ricchi, gli scenari sono sensati e le raccomandazioni coerenti; quando sono poveri, prevale il segnale trasversale.

Il sistema è adatto a discovery e curation di crossover, lasciando spazio a moduli neurali/CF come re-ranker, mantenendo però tracciabilità delle scelte.

12 Conclusioni e sviluppi futuri

Chiudiamo la tesi riassumendo i contributi principali, i risultati emersi e le direzioni che riteniamo più promettenti.

12.a Conclusioni

Contributo metodologico. Abbiamo mostrato che una pipeline leggera, interamente spiegabile, può supportare la combinazione concettuale di generi musicali:

raccolta ed enrichment dei testi da Genius con stima della ripetizione;

prototipi per brano con pesi normalizzati;

costruzione di profili typical/rigid per macro-genere;

preprocessing Head/Modifier e combinazione con CoCoS (scenari pesati in stile TCL);

raccomandatore che classifica e spiega i suggerimenti sulla base dei tratti ereditati dallo scenario selezionato.

Trasparenza. Ogni raccomandazione è accompagnata da: rigid rispettate, tipiche attivate e score di scenario. Questo consente auditabilità e controllo fine (es. agendo sulle soglie o sul ruolo Head/Modifier).

Risultati pratici. La pipeline ha prodotto prototipi ibridi plausibili per molte coppie di generi, con copertura piena nel classificatore e spiegazioni coerenti con i profili. Dove i profili sono ricchi, CoCoS propone più scenari sensati; dove sono poveri, prevalgono segnali trasversali (ad es. high_repetition), confermando l'importanza della cura delle typical distintive.

Limiti. (i) dipendenza dalla qualità di testi/metadata e dalle mappe SUB2MACRO; (ii) ipotesi di indipendenza dei pesi tipici nello scoring degli scenari; (iii) ruoli Head/Modifier talvolta troppo rigidi; (iv) scarsa multilingua: l'intera pipeline è tarata sull'inglese (tokenizzazione, stoplist, tagger).

12.b Sviluppi futuri

Multilingua (priorità).

Portare l'estrazione e i prototipi a più lingue (italiano in primis): tokenizzazione, stoplist e lemmatizzazione per lingua;

normalizzazione cross-lingua delle proprietà (sinonimi, varianti morfologiche) e mapping dei tag;

scelta dinamica del modello in base alla lingua del brano e possibilità di mix multilingua.

Feature audio e metadata strutturati.

Integrare deskriptor audio (tempo stimato, energy, spectral features) e campi strutturati (anno, provenienza, mood editoriali) come typical aggiuntive o vincoli rigid;

fusione tardo/early con pesi apprendibili dal feedback.

Apprendimento dei pesi e dei vincoli.

Stimare automaticamente gradi typical e regole di co-occorrenza/antagonismo tra proprietà (dipendenze) a partire dai dati;

active learning per far correggere al curatore gli scenari e aggiornare i profili.

Arricchimento lessicale e mapping di genere.

Ampliare DOMAIN_WHITELIST e SUB2MACRO, includendo slang e sottogeneri emergenti;

usare embedding per consolidare sinonimi e ridurre la frammentazione del vocabolario.

CoCoS più espressivo.

Scenari con gruppi coerenti di feature (es. se trap allora 808) e penalità per combinazioni incoerenti;

plasticità del ruolo Head/Modifier e scelta automatica del verso più naturale (H/M o M/H) per ogni coppia.

Valutazione su utenti.

Studio utente e A/B test sulle spiegazioni: misurare fiducia, utilità percepita e qualità del discovery;

metriche di diversità/novità nelle playlist ibride e confronto con baseline neurali o collaborative.

Tooling e riproducibilità.

Report automatici di copertura, proprietà mai attivate e bottleneck di scenario;

packaging della pipeline con config condivisibili e seed fissati per esperimenti ripetibili.

In sintesi, la tesi dimostra che prototipi + combinazione tipica è un paradigma efficace e trasparente per generare crossover musicali spiegabili. Con multilingua, feature audio e apprendimento dei pesi, il sistema può diventare uno strumento pratico di curation e discovery per playlist, editoria e creatività assistita.

Bibliografia / Sitografia

[1] *UniTO Typst Template*. (2024). [Online]. Disponibile su: https://github.com/eduardz1/UniTO-typst-template

[2] «Typst — A new markup-based typesetting system». [Online]. Disponibile su: https://typst.app/

[3] Alberto Marocco, *DEGARI-Music*. (2025). [Online]. Disponibile su: https://github.com/albymar01/DEGARI-Music

[4] Alberto Marocco, *Tesi-UniTO (manoscritto)*. (2025). [Online]. Disponibile su: https://github.com/albymar01/Tesi-UniTO

[5] «Genius». [Online]. Disponibile su: https://genius.com/

[6] «Scrapy». [Online]. Disponibile su: https://docs.scrapy.org/en/latest/

[7] «NLTK — Natural Language Toolkit». [Online]. Disponibile su: https://www.nltk.org/

[8] «TreeTaggerWrapper». [Online]. Disponibile su: https://treetaggerwrapper.readthedocs.io/

[9] Helmut Schmid, «TreeTagger». [Online]. Disponibile su: https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

[10] A. Valese, «CoCoS: uno strumento per la combinazione di concetti», 2020.

[11] «scikit-learn». [Online]. Disponibile su: https://scikit-learn.org/

[12] «pandas». [Online]. Disponibile su: https://pandas.pydata.org/

[13] «NumPy». [Online]. Disponibile su: https://numpy.org/

[14] «Matplotlib». [Online]. Disponibile su: https://matplotlib.org/

[15] Peter Gärdenfors, «Concept Combination and Prototypes», 2004.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, e Clifford Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

Ringraziamenti

TO DO