

**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

<b>Introduzione</b>	<b>1</b>
1.a Obiettivi .....	1
1.b Contributi .....	2
1.c Metodologia in breve .....	2
1.d Perimetro e limiti .....	3
1.e Sintesi .....	3
<b>Fondamenti teorici: tipicità e combinazione di concetti (TCL)</b>	<b>4</b>
2.a Tipicità e chiusura razionale .....	4
2.a.I Notazione di base .....	4
2.b TCL: Typicality-based Compositional Logic .....	4
2.b.I Ruolo HEAD / MODIFIER .....	5
2.b.II Perché serve TCL (il “pet-fish”) .....	5
2.c Semantica probabilistica delle tipicità .....	5
2.d Il sistema CoCoS (cenni) .....	5
<b>Probabilità e strumenti per la combinazione concettuale (TCL e CoCoS)</b>	<b>6</b>
3.a Inclusioni probabilistiche e scenari in TCL .....	6
3.b CoCoS: input, algoritmo, output .....	6
3.c Estensioni e contesto d’uso .....	7
3.d Collocazione nella pipeline di DEGARI-Music .....	7
<b>Estrazione e pre-processing dei dati (Genius)</b>	<b>8</b>
4.a Accesso all’API Genius e gestione del token .....	8
4.b Raccolta dei brani: criteri e formato di output .....	8
4.c Enrichment della ripetizione e tag derivati .....	9
4.d Nota sulla variante “extended” .....	9
<b>Creazione dei prototipi (Modulo 1)</b>	<b>10</b>
5.a Input e obiettivo .....	10
5.b Pre-processing linguistico .....	10
5.c Estrazione e pesatura delle feature .....	10
5.d Formato dell’output .....	11
5.e Esempio sintetico .....	11

5.f	Considerazioni implementative .....	11
<b>Costruzione delle proprietà tipiche e rigide (Modulo 2)</b>		<b>12</b>
6.a	Scopo e dati di ingresso .....	12
6.b	Mappatura dei generi e vocabolario .....	12
6.c	Pipeline di calcolo (per genere) .....	13
6.d	Differenze tra <b>TAG</b> e <b>WORD</b> .....	13
6.e	Iperparametri e impatto pratico .....	14
6.f	Esempio di esecuzione e lettura dei risultati .....	14
6.g	Buone pratiche .....	15
<b>Preprocessing CoCoS: costruzione dei file H_M (Modulo 3a)</b>		<b>16</b>
7.a	Scopo, input e percorsi .....	16
7.b	Algoritmo ( <code>cocos_preprocessing.py</code> ) .....	16
7.c	Esecuzione .....	17
7.d	Formato del file <code>H_M.txt</code> (esempi) .....	17
7.e	Note pratiche .....	17
7.f	Collegamento al Modulo 3b .....	18
<b>CoCoS: combinazione e scenari (Modulo 3b)</b>		<b>19</b>
8.a	Input e avvio .....	19
8.b	Generazione e scoring degli scenari .....	19
8.c	Selezione e output .....	19
8.d	Parametri pratici .....	20
8.e	Output per i moduli successivi .....	20
<b>Sistema di raccomandazione (Modulo 4)</b>		<b>21</b>
9.a	Input e panoramica .....	21
9.b	Dal file CoCoS al prototipo “pulito” .....	21
9.c	Regole di matching .....	21
9.d	Formato dell’output .....	22
9.e	Uso a riga di comando e integrazione nella pipeline .....	22
9.f	Considerazioni e limiti .....	23
9.g	Collegamento ai capitoli successivi .....	23
<b>Risultati</b>		<b>24</b>
10.a	Metriche e formato .....	24
10.b	Copertura .....	24
10.c	Riclassificazioni di brani (case-study) .....	25
10.c.I	Trap-Metal (HEAD = trap, MODIFIER = metal) .....	25
10.c.II	Metal-Trap (HEAD = metal, MODIFIER = trap) .....	26
10.c.III	Reggae-Rap vs Rap-Reggae .....	27
10.d	Coerenza dei suggerimenti con il prototipo .....	29
10.e	Esempi puntuali (ancore colpite) .....	29

10.f Limiti osservati .....	29
10.g Sintesi dei risultati .....	29
<b>Discussione</b>	<b>30</b>
11.a Dove il sistema fallisce (e perché) .....	30
11.b Confronto con approcci affini .....	30
11.c Implicazioni pratiche .....	31
11.d Minacce alla validità .....	31
11.e Cosa migliorare subito .....	31
11.f Takeaway .....	31
<b>Conclusioni e sviluppi futuri</b>	<b>33</b>
12.a Conclusioni .....	33
12.b Sviluppi futuri .....	33
12.c In sintesi .....	34
<b>Bibliografia / Sitografia</b>	<b>35</b>

# 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 multi-lingua 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 TIPICTÀ

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)

Approfondiamo adesso invece la componente probabilistica di **TCL** e 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  $p \in (0.5, 1]$ . 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)

Nel seguente capitolo vengono descritte 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 viene utilizzato `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`  $\geq$  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 stopwords 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**  $\rightarrow$  **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]`:

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

dove



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

. Se

$$\text{max\_freq} = \text{min\_freq} \quad (3)$$

, si assegna

$$\text{score} = 0.9 \quad (4)$$

(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 nella cartella `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 COSTRUZIONE DELLE PROPRIETÀ TIPICHE E RIGIDE (MODULO 2)

Passiamo adesso al secondo modulo dove `BuildTypicalRigid.py` costruisce, per ciascun macro-genere, due insiemi di proprietà: **rigide** (vincoli non derogabili) e **tipiche** (inclusioni con grado), a partire dal JSON dei brani generato nel preprocessing di Genius. Il risultato alimenta il preprocessing di **CoCoS** e la successiva combinazione **HEAD/MODIFIER**.

### 6.A SCOPO E DATI DI INGRESSO

- **Scopo.** Per ogni macro-genere si producono due file: `typical/<genere>.txt` (righe `proprietà: peso`) e `rigid/<genere>.txt` (una proprietà per riga).
- **Input.** Un JSON con i brani (titolo, artista, lyrics, `tags`, eventuali `subgenres`), da cui si calcolano frequenze documentali per tag e parole.
- **Output.** Insiemi compatti e bilanciati di tratti caratteristici: le **rigide** come ancora forti, le **tipiche** come proprietà con grado `p in [0.60, 0.95]` normalizzato per **CoCoS**.

### 6.B MAPPATURA DEI GENERI E VOCABOLARIO

- **Macro-generi.** L'analisi avviene su un set di macro-generi predefinito (es. `rap`, `metal`, `rock`, `pop`, `trap`, `reggae`, `rnb`, `country`).
- **Da subgenere a macro-genere.** Una mappa `SUB2MACRO` ricondurrà etichette come `hip_hop`, `boom_bap`, `drill` a `rap`; `dancehall` a `reggae`; ecc. Inoltre si controllano `title` / `album` / `artist` per eventuali occorrenze indicative.
- **Token testuali (lyrics).** Si estraggono parole “utili” con una regex alfabetica ( $\geq 3$  caratteri), normalizzando minuscole/apostrofi. Si filtra una stoplist inglese, ma un `DOMAIN_WHITELIST` preserva termini di dominio (es. `hiphop`, `808`, `trap`, `metal`, `hook_repetition`, `catchy_chorus`, `high_repetition`).

## 6.C PIPELINE DI CALCOLO (PER GENERE)

1. **Partizionamento.** Ogni brano è assegnato a uno o più macro-generi usando `subgenres`, `tags` e, in seconda battuta, il testo di `title / album / artist`.
2. **Document frequency.** Per ciascun genere si calcolano DF di `tags` e di parole (dalle lyrics). Per le parole si applica `min_df_words` per scartare termini troppo rari.
3. **Frazione di presenza.** Si ottiene, per ogni proprietà `p` nel genere `g`, la frazione  $\text{frac}(p,g) = \text{df}(p,g) / N_g$ , dove `N_g` è il numero di brani del genere.
4. **Selezione “raw”.** Entrano candidati **tipici**:
  - `tags` con `frac >= typical_thr_tags`;
  - parole con `frac >= typical_thr_words`, escludendo globalità come `high_repetition` se non nel `DOMAIN_WHITELIST`.
5. **Scoring delle tipiche.** Per ogni candidato si combina **prevalenza** nel genere e **specificità** cross-genere (tipo-IDF), con peso `ALPHA`:
  - $\text{score} \approx \text{ALPHA} * \text{frac} + (1-\text{ALPHA}) * (\text{frac} / \text{idf})$ ; alle parole si applica un fattore di prudenza (es. `0.8`).
  - Penalità ai tratti molto globali (`COMMON_PENALTY`) e **boost** alle proprietà distintive che compaiono in pochi generi (`DISTINCTIVE_BOOST` con soglia `DISTINCTIVE_MAX_GENRES`).
6. **Top-k e normalizzazione.** Si prendono le `topk_typical` proprietà per score; quindi si normalizzano i pesi nell'intervallo `[MIN_W, MAX_W]` (default `0.60..0.95`) con min-max scaling (caso degenerato  $\rightarrow$  `0.80`).
7. **Scelta delle rigide.**
  - `tags` con `frac >= rigid_thr_tags`;
  - parole con `frac >= rigid_thr_words` ma **solo** se presenti nel `DOMAIN_WHITELIST`;
  - si limita a `max_rigid`, preservando l'ordine di apparizione (ancore concise e robuste).
8. **Scrittura file.**
  - `typical/<genere>.txt` contiene righe `proprieta: peso` ordinate per peso decrescente (poi lessico);
  - `rigid/<genere>.txt` elenca le proprietà rigide (una per riga).

## 6.D DIFFERENZE TRA TAG E WORD

- I **TAG** (es. `high_repetition`, `catchy_chorus`) derivano dai metadati/arricchimenti e tendono a essere segnali puliti; possono diventare facilmente **rigide** se ubiqui nel genere.

- Le **WORD** provengono dalle lyrics, sono più rumorose: per questo c'è `min_df_words` e un fattore prudenziale nello scoring; inoltre diventano **rigide** solo se ricadono nel `DOMAIN_WHITELIST` e superano `rigid_thr_words`.

## 6.E IPERPARAMETRI E IMPATTO PRATICO

- `typical_thr_tags` / `typical_thr_words`: alzare le soglie rende i **typical** più selettivi (meno proprietà, più pulizia); abbassarle amplia la copertura (più proprietà, più rumore).
- `rigid_thr_tags` / `rigid_thr_words`: soglie alte creano **rigide** davvero onnipresenti; soglie più basse aumentano le ancore ma rischiano di vincolare eccessivamente **CoCoS**.
- `min_df_words`: alzare riduce il rumore lessicale; abbassare permette a più termini informativi di entrare.
- `topk_typical`: più alto → più materiale per **CoCoS** (scenari più ricchi), ma potenziale rumore; più basso → tipiche più “forti” ma minor varietà.
- `max_rigid`: più alto → più ancore (scenari più vincolati, rischio “NO scenario”); più basso → più flessibilità (ma ancore meno protettive).
- `ALPHA`: se cresce, il profilo privilegia la prevalenza interna al genere; se diminuisce, enfatizza la specificità (tratti distintivi cross-genere).
- `COMMON_PENALTY`, `DISTINCTIVE_MAX_GENRES`, `DISTINCTIVE_BOOST`: controllano rispettivamente la penalità ai tratti “orizzontali”, la soglia per considerare distintiva una proprietà e l’entità del boost.
- `MIN_W`, `MAX_W`: fissano il range finale dei pesi tipici (coerente con `p` in `(0.5,1]` per l’uso in **TCL**).

## 6.F ESEMPIO DI ESECUZIONE E LETTURA DEI RISULTATI

Esecuzione (parametri lievemente più permissivi rispetto ai default “strict”):

```
python BuildTypicalRigid.py --input "<base>/descr_music_GENIUS.json"
--out "<base>" --typical_thr_tags 0.80 --rigid_thr_tags 0.98 --
typical_thr_words 0.80 --rigid_thr_words 0.98 --min_df_words 8 --
topk_typical 6 --max_rigid 2
```

- Confronto con i default “strict”: `typical_thr_*` da `0.85` → `0.80` (più copertura), `min_df_words` da `10` → `8` (più termini candidati), `topk_typical` da `5` → `6` (più tipiche), `max_rigid` da `3` → `2` (meno ancore, più flessibilità).
- Effetto atteso su **CoCoS**: più tipiche disponibili per scenario, minore rischio di over-constrain grazie a `max_rigid` più basso; potenziale aumento di scenari ammissibili e minore frequenza di “NO scenario”.

## 6.G BUONE PRATICHE

- Preferire poche **rigide** molto robuste (due o tre) e un set di **tipiche** bilanciato (5–8) per genere.
- Se compaiono troppe proprietà “orizzontali” (es. `high_repetition`), valutare un `COMMON_PENALTY` più forte o aumentare `typical_thr_words`.
- Se i profili risultano poveri, abbassare moderatamente `typical_thr_*` e/o `min_df_words` e aumentare `topk_typical`, verificando a valle l’impatto su **CoCoS**.

## 7 PREPROCESSING CoCoS: COSTRUZIONE DEI FILE H\_M (MODULO 3A)

Il preprocessing genera i file di input per CoCoS partendo dai profili per genere prodotti nel Modulo 2 ( `typical/` e `rigid/` ), 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 **rigide** dell'**head** ( `head, <prop>` ), poi del **modifier** ( `modifier, <prop>` );
  - blocco **tipiche** del **modifier** ( `T(modifier), <prop>, <peso>` );
  - blocco **tipiche** 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 Con-  
cept 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 **rigide** sono riportate come vincoli duri e saranno sempre rispettate da **CoCoS**.
- I pesi delle **tipiche** sono copiati dai file di genere (range tipico `[0.60, 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)

CoCoS elabora le coppie **HEAD/MODIFIER** generato nel passo precedente. 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), <proprieta>, <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** (`MAX_ATTRS`).

### 8.B GENERAZIONE E SCORING DEGLI SCENARI

Dalle **tipiche** disponibili si costruiscono sottoinsiemi ammissibili (nel rispetto delle **rigide** 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**.

**Nota.** L'array `Scenario: [...]` codifica, nell'ordine in cui compaiono nel file, le **tipiche** attive (1) e inattive (0); l'ultimo valore è lo **score** dello scenario.

## 8.D PARAMETRI PRATICI

- `MAX_ATTRS`. Numero massimo di tipiche ereditabili (default nel config, sovrascrivibile da CLI).
- **Coerenza HEAD/MODIFIER**. Le **rigide** di entrambi si rispettano sempre; tra le **tipiche** 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 (messaggio: **NO recommended scenarios!**): 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, 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** (sezioni iniziali con **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 `H_M_recommendations.json` con: categoria, prototipo attivo, ancore, lista dei risultati `id-title-artist-matches-anchors_hit`, numero di brani classificati su totale. Gli output sono poi raccolti 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 come obbligate nella sezione iniziale);
- rimuove duplicati e normalizza i nomi.

L'esito è una lista di proprietà da soddisfare e 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, lyrics e campi descrittivi come **tags** / **subgenres**);
- opzionalmente scarta il brano se contiene proprietà **negate** (prefisso **-** nel prototipo);

- accetta il brano se sono vere entrambe le soglie:
  - `match-rate` minimo (default `0.15`), dove  $\text{match-rate} = \# \text{match} / \# \text{proprietà\_prototipo}$ ;
  - minimo numero di **ancore** colpite (default `1`).

Le soglie sono modificabili da CLI (`--min-match-rate`, `--min-anchors`). L'opzione `--max-print` limita solo la verbosità a console (non influisce sui JSON).

## 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": "Eminem - Rap God",
      "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 applica uno **score** continuo; in prospettiva è possibile ordinare per numero di **match** o per somma pesata dei gradi tipici (riusando i **p** di **CoCoS**).

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

**Coerenza.** Se **CoCoS** non ha 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 i capitoli **Risultati** e **Spiegazioni**, dove analizziamo copertura, varietà e casi tipici di **match** per le combinazioni testate.

# 10 RISULTATI

I risultati vengono analizzati alla luce degli esiti della pipeline: prototipi ibridi → classificatore/recommender per ciascuna coppia **HEAD/MODIFIER**. Gli output sono file `H_M_recommendations.json` che, per ogni coppia, riportano: categoria, prototipo (proprietà selezionate), **ancore** (rigide/forti), lista dei brani consigliati con le proprietà effettivamente colpite e i 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"]`,  
`anchors = ["trap","metal","high_repetition"]`, `classified = 48`,  
`total = 48`.

Riepilogo campi del JSON (esempio **trap-metal**):

- **Campo** → `prototype` : `["trap","high_repetition","metal"]`
- **Campo** → `anchors` : `["trap","metal","high_repetition"]`
- **Campo** → `classified` : `48`
- **Campo** → `total` : `48`

## 10.B COPERTURA

Nel run corrente, la copertura è **piena** su tutte le coppie (`classified = total = 48`; es. **trap-metal**, **metal-trap**, **reggae-rap**). In altri run o con profili più scarsi, la copertura può ridursi (es. 79% su coppie con poche tipiche/rigide), coerentemente con assenza di scenari in **CoCoS** o con proprietà insufficienti nel prototipo.

## 10.C RICLASSIFICAZIONI DI BRANI (CASE-STUDY)

### 10.C.I TRAP-METAL (HEAD = TRAP, MODIFIER = METAL)

Scenario selezionato (da "scenarios\_json/trap\_metal"): { «metal»: 0.95, «trap»: 0.95, «high\_repetition»: 0.618, «@scenario\_score»: 3.5695680000000003 } Il prototipo ibrido enfatizza trap, metal, high\_repetition. Le **ancore** attive sono ["trap", "metal", "high\_repetition"].

```
{
  "head": "trap",
  "modifier": "metal",
  "recommended_scenarios": [
    {
      "metal": 0.95,
      "trap": 0.95,
      "high_repetition": 0.618,
      "@scenario_probability": 3.5695680000000003,
      "@scenario_score": 3.5695680000000003
    }
  ]
}
```

Top-5 brani riclassificati (trap-metal, da trap\_metal\_recommendations.json):

- 1) Rap God — **Eminem** — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]
- 2) Lose Yourself — **Eminem** — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]
- 3) Mockingbird — **Eminem** — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]
- 4) Not Like Us — **Kendrick Lamar** — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]
- 5) HUMBLE. — **Kendrick Lamar** — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]

#### Lettura dei risultati.

- molte tracce **trap** colpiscono **high\_repetition** e soddisfano l'ancora omonima;
- l'identità **metal** resta vincolo del prototipo: i brani accettati mostrano tendenze aggressive/"dense" compatibili;
- compaiono brani con **anchors\_hit = ["high\_repetition"]** quando l'**hook** è particolarmente marcato.

### 10.C.II METAL-TRAP (HEAD = METAL, MODIFIER = TRAP)

Scenario selezionato (da "scenarios\_json/metal\_trap"): { «metal»: 0.95, «trap»: 0.95, «high\_repetition»: 0.618, «@scenario\_score»: 3.5695680000000003 }.

Lo scenario coincide con la direzione precedente. Anche qui `prototype = ["high_repetition", "trap", "metal"]` e `anchors = ["trap", "metal", "high_repetition"]`.

```
Trap Metal Recommendations
{
  "category": "metal_trap.txt",
  "prototype": [
    "high_repetition",
    "metal",
    "trap"
  ],
  "anchors": [
    "trap",
    "metal",
    "high_repetition"
  ],
  "results": [
    {
      "id": "rap_eminem_rap-god_2013_235729",
      "title": "Rap God",
      "artist": "Eminem",
      "matches": [
        "high_repetition"
      ],
      "anchors_hit": [
        "high_repetition"
      ]
    }
  ],
}
```

Top-5 brani riclassificati (metal-trap, da metal\_trap\_recommendations.json):

- 1) Rap God — Eminem — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]
- 2) Lose Yourself — Eminem — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]
- 3) Mockingbird — Eminem — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]
- 4) Not Like Us — Kendrick Lamar — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]
- 5) HUMBLE. — Kendrick Lamar — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]

**Osservazione direzionale ( $H \rightarrow M$  vs  $M \rightarrow H$ ).** Scenario e punteggio sono simmetrici. La riclassificazione concreta è però più ricca sul lato **trap**→**me-**



tal (maggior disponibilità di crossover trap con tratti aggressivi) rispetto a metal→trap, dove i match sono spesso trainati da high\_repetition.

### 10.C.III REGGAE-RAP VS RAP-REGGAE

Reggae→Rap (HEAD = reggae, MODIFIER = rap). CoCoS propone più scenari coerenti, ad es.: { «reggae»: 0.95, «high\_repetition»: 0.8, «hook\_repetition»: 0.6, «@scenario\_score»: 7.296000000000001 }.

{ «reggae»: 0.95, «high\_repetition»: 0.6, «@scenario\_score»: 7.296000000000001 }.

{ «reggae»: 0.95, «high\_repetition»: 0.8, «catchy\_chorus»: 0.6, «@scenario\_score»: 7.296000000000001 }.

Il profilo reggae dispone di una proprietà identitaria reggae: 0.95 (typical) e di rigid high\_repetition, hook\_repetition: questo consente di preservare l'identità reggae come ancora, integrando segnali trasversali (high\_repetition, hook\_repetition) e, in misura minore, contributi da rap.

```
{
  "head": "reggae",
  "modifier": "rap",
  "recommended_scenarios": [
    {
      "reggae": 0.95,
      "high_repetition": 0.8,
      "hook_repetition": 0.6,
      "@scenario_probability": 7.296000000000001,
      "@scenario_score": 7.296000000000001
    },
    {
      "reggae": 0.95,
      "high_repetition": 0.6,
      "@scenario_probability": 7.296000000000001,
      "@scenario_score": 7.296000000000001
    },
    {
      "reggae": 0.95,
      "high_repetition": 0.8,
      "catchy_chorus": 0.6,
      "@scenario_probability": 7.296000000000001,
      "@scenario_score": 7.296000000000001
    }
  ]
}
```

Top-5 brani riclassificati (reggae-rap, da reggae\_rap\_recommendations.json):

- 1) Rap God — Eminem — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]
- 2) Lose Yourself — Eminem — matches = ["high\_repetition"], anchors\_hit = ["high\_repetition"]
- 3) Mockingbird — Eminem —

```

matches = ["high_repetition"], anchors_hit = ["high_repetition"] 4)
Not Like Us — Kendrick Lamar — matches = ["high_repetition"],
anchors_hit = ["high_repetition"] 5) HUMBLE. — Ken-
drick Lamar — matches = ["high_repetition", "hook_repetition"],
anchors_hit = ["high_repetition", "hook_repetition"]

```

\*Nota\*: il **prototype** per **reggae-rap** è [«hook\_repetition», «high\_repetition»] con **ancore** [«high\_repetition», «hook\_repetition»]; i brani 1–4 colpiscono solo l’ancora **high\_repetition**, mentre **HUMBLE.** colpisce entrambe le ancore, risultando il caso più informativo.

**Rap→Reggae** (**HEAD** = **rap**, **MODIFIER** = **reggae**). Per questa direzione, CoCoS **non** genera scenari raccomandati. La ragione è strutturale:

- **rap** (**HEAD**) ha profilo povero: `rap rigid = ["high_repetition"], rap typical = {"high_repetition": 0.8};`
- non esiste una tipica identitaria `rap: 0.95` da “ancorare”;
- imporre **reggae** come **MODIFIER** violerebbe il vincolo di preservare l’identità dell’**HEAD** (che qui non è distinguibile dal solo segnale trasversale **high\_repetition**).

Di conseguenza, la combinazione decadrebbe a un profilo **quasi generico** di ripetizione/ritornello, che **CoCoS** scarta come scenario ibrido **rap-reggae**.

```

{
  "head": "reggae",
  "modifier": "rap",
  "recommended_scenarios": [
    {
      "reggae": 0.95,
      "high_repetition": 0.8,
      "hook_repetition": 0.6,
      "@scenario_probability": 7.296000000000001,
      "@scenario_score": 7.296000000000001
    },
    {
      "reggae": 0.95,
      "high_repetition": 0.6,
      "@scenario_probability": 7.296000000000001,
      "@scenario_score": 7.296000000000001
    },
    {
      "reggae": 0.95,
      "high_repetition": 0.8,
      "catchy_chorus": 0.6,
      "@scenario_probability": 7.296000000000001,
      "@scenario_score": 7.296000000000001
    }
  ]
}

```

## 10.D COERENZA DEI SUGGERIMENTI CON IL PROTOTIPO

**Trap–Metal.** Il prototipo ibrido enfatizza `trap`, `metal`, `high_repetition`. Nei risultati:

- tracce **trap** colpiscono `high_repetition` (e talvolta `trap`) soddisfacendo le **ancore**;
- brani **metal** colpiscono `metal` + `high_repetition`;
- ingressi di altri generi sono giustificati dalla forte `high_repetition`.

**Reggae–Rap.** Il mantenimento dell’identità `reggae` (tipica `reggae: 0.95` + rigide di ripetizione) spiega perché gli scenari esistono solo nella direzione `reggae→rap`.

## 10.E ESEMPI PUNTUALI (ANCORE COLPITE)

- **Trap–Metal** → brano con `anchors_hit = ["high_repetition", "trap"]`.
- **Metal–Trap** → brano con `anchors_hit = ["high_repetition", "metal"]`.
- **Reggae–Rap** → brano con `anchors_hit = ["high_repetition", "hook_repetition"]`.
- **Rap–Reggae** → **nessuno scenario**: impossibile preservare un’identità **rap** distinta dal solo `high_repetition`.

## 10.F LIMITI OSSERVATI

**Dipendenza da `high_repetition`.** Essendo un segnale “orizzontale”, allarga la platea se il prototipo non contiene altre tipiche/rigide selettive; l’effetto è utile per esplorare **cross-over**, ma va bilanciato.

**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**).

## 10.G SINTESI DEI RISULTATI

Il recommender preserva le scelte di **CoCoS**: le **ancore** del prototipo ibrido guidano i suggerimenti. I tag di ripetizione (da enrichment) 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) cresce la precisione semantica e diminuisce la dipendenza da `high_repetition`.

# 11 DISCUSSIONE

Questo capitolo interpreta i risultati alla luce della pipeline (prototipi → **CoCoS** → recommender), evidenziando limiti, confronto con alternative e implicazioni d’uso. Funziona da ponte logico tra **Risultati** e **Conclusioni**.

## 11.A DOVE IL SISTEMA FALLISCE (E PERCHÉ)

**Propagazione degli errori.** Ogni modulo eredita i vincoli del precedente: profili **tipiche/rigide** scarsi o sbilanciati generano pochi scenari e bassa copertura del recommender.

**Dominanza di segnali trasversali.** Trattati “orizzontali” (es. `high_repetition`) allargano la platea ma riducono la specificità se mancano **tipiche** distintive. **Rimedi:** (i) aumentare `MAX_ATTRS` solo quando esistono **tipiche** non trasversali, (ii) rinforzare **tipiche** caratterizzanti dell’**HEAD**, (iii) alzare moderatamente `--min-match-rate` e `--min-anchors` nel recommender.

**Assunzione di indipendenza.** Lo **scenario score** usa il prodotto dei `p` delle **tipiche** attive; proprietà fortemente correlate possono risultare sovrastimate. **Mitigazione:** semplici regole di co-occorrenza/alternanza tra gruppi di feature.

**Euristica HEAD/MODIFIER.** La priorità semantica all’**HEAD** è utile ma rigida: coppie come `pop-rnb` possono richiedere simmetria o **switch** dinamico dei ruoli. **Estensioni:** parametro di “plasticità” dell’asimmetria oppure scelta `H/M` guidata dallo score di scenario.

**Rumore testuale.** Tokenizzazione/stopword inglesi e qualità variabile dei testi introducono sinonimia superficiale (senza lemmatizzazione), lessico di dominio fuori lista e residui di markup. **Rimedi:** ampliare liste/lessici, pulizia markup, lemmatizzazione quando disponibile.

## 11.B CONFRONTO CON APPROCCI AFFINI

**Bag-of-words / tf-idf.** Buone similitudini ma nessuna distinzione tra **tipiche** (difettibili) e **rigide**; manca il ruolo `HEAD/MODIFIER` e la suspendibilità dei tratti.

**Collaborative filtering / embedding neurali.** Ottime previsioni di preferenza, spiegabilità ridotta; la combinazione concettuale richiede workaround. Qui il prototipo ibrido è esplicito e auditabile (scenari + pesi).

**TCL/CoCoS.** Rispetto a logiche crisp/fuzzy tradizionali offre (i) priorità e sospendibilità delle **tipiche**, (ii) selezione di scenari con punteggi, (iii) ruolo **HEAD/MODIFIER**. Controparte: maggiore sensibilità alla qualità dei profili e all'ipotesi d'indipendenza.

## 11.C IMPLICAZIONI PRATICHE

**Trasparenza e controllo.** Ogni suggerimento espone **ancore** e **matches** del prototipo; il curatore può regolare soglie e liste proprietà con impatto prevedibile.

**Discovery di crossover.** Segnali trasversali (hook/ritornelli) fanno emergere brani “lontani” ma plausibili: utile per playlist tematiche, format editoriali e ideazione creativa.

**Cura dei profili.** Poche **rigide** forti e 2–3 **tipiche** distintive per genere migliorano gli scenari senza complicare la pipeline.

## 11.D MINACCE ALLA VALIDITÀ

**Copertura dati.** Pochi esempi per genere riducono la stabilità di **tipiche/rigide** e lo score scenari. **Bias di sorgente.** Testi/metadata di Genius riflettono pratiche editoriali specifiche. **Iperparametri.** `MAX_ATTRS`, `--min-match-rate`, `--min-anchors` influenzano direttamente scenari e copertura.

## 11.E COSA MIGLIORARE SUBITO

**Specificità.** Arricchire i profili con **tipiche** non trasversali per genere (riduce la dipendenza da `high_repetition`). **Coerenza leggera.** Poche regole di preferenza/evitamento tra proprietà correlate/incompatibili. **Selezione scenari più soft.** Conservare i **top-k** scenari e demandare diversificazione al recommender con pesi. **Diagnostica.** Report automatico: brani non classificati per coppia, proprietà mai attivate, **rigide** che annullano scenari. **Arricchimento linguistico.** Liste termini e mappature verso macro-tratti; lemmatizzazione quando possibile.

## 11.F TAKEAWAY

Il paradigma **prototipi + combinazione** offre spiegazioni locali e controllo globale con pochi iperparametri. La qualità di **tipiche/rigide** è la leva principale: profili ricchi → scenari sensati e raccomandazioni coerenti; profili poveri → prevalgono segnali trasversali. Il sistema è adatto a **discovery** e **curation**

di crossover e può integrare modelli neurali/CF come **re-ranker**, mantenendo tracciabilità delle scelte.

## 12 CONCLUSIONI E SVILUPPI FUTURI

Si chiude il lavoro riassumendo i contributi, i risultati emersi e le linee evolutive più promettenti.

### 12.A CONCLUSIONI

**Contributo metodologico.** È stata mostrata una pipeline leggera e spiegabile per la combinazione concettuale di generi musicali, articolata in: (i) raccolta ed **enrichment** dei testi da Genius con stima della ripetizione; (ii) prototipi per brano con pesi normalizzati; (iii) profili di macro-genere con **tipiche/rigide**; (iv) preprocessing **HEAD/MODIFIER** e combinazione con **CoCoS** (scenari pesati in stile **TCL**); (v) **recommender** che classifica e spiega i suggerimenti in base ai tratti ereditati dallo scenario selezionato.

**Trasparenza.** Ogni raccomandazione espone **rigide** rispettate, **tipiche** attivate e **scenario score**, abilitando auditabilità e controllo fine (soglie, elenco proprietà, ruolo **HEAD/MODIFIER**).

**Risultati pratici.** La pipeline ha prodotto prototipi ibridi plausibili per molte coppie di generi, con copertura piena in vari casi e spiegazioni coerenti con i profili. Dove i profili sono ricchi, **CoCoS** propone più scenari sensati; dove sono poveri, prevalgono segnali trasversali (es. **high\_repetition**), evidenziando l'importanza di **tipiche** distintive.

**Limiti.** (i) dipendenza dalla qualità di testi/metadata; (ii) ipotesi di indipendenza nello **scenario score**; (iii) ruoli **HEAD/MODIFIER** talvolta rigidi; (iv) supporto multilingua non ancora end-to-end (tokenizzazione, stoplist, lemmatizzazione centrati sull'inglese).

### 12.B SVILUPPI FUTURI

**Multilingua (priorità).** Portare estrazione e prototipi a più lingue (italiano in primis): tokenizzazione, stoplist e lemmatizzazione per lingua; normalizzazione **cross-lingua** di proprietà/sinonimi e mapping dei tag; scelta dinamica della lingua del brano ed eventuale combinazione multilingua.

**Feature audio e metadata strutturati.** Integrare descrittori audio (tempo, energia, spettrali) e campi strutturati (anno, provenienza, **mood**) come **tipiche** o **rigide**; fusione **early/late** con pesi apprendibili dal feedback.

**Apprendimento di pesi e vincoli.** Stimare automaticamente gradi tipici e regole leggere di co-occorrenza/antagonismo; **active learning** per correggere scenari e aggiornare i profili con il curatore.

**Arricchimento lessicale e mapping di genere.** Ampliare liste di termini (slang, sottogeneri emergenti) e usare rappresentazioni distribuzionali per consolidare sinonimi e ridurre la frammentazione del vocabolario.

**CoCoS più espressivo.** Scenari con gruppi coerenti di feature e penalità per combinazioni incoerenti; plasticità del ruolo **HEAD/MODIFIER** e scelta automatica del verso ( **H/M** o **M/H** ) per ogni coppia.

**Valutazione su utenti.** Studio utente e A/B test sulle spiegazioni per misurare fiducia/utile percepita; metriche di diversità/novità per playlist ibride e confronto con **baseline** neurali o collaborative.

**Tooling e riproducibilità.** Report automatici di copertura (brani non classificati, proprietà mai attivate, **rigide** bloccanti); **packaging** con configurazioni condivisibili e **seed** fissati per esperimenti ripetibili.

## 12.C IN SINTESI

Il paradigma **prototipi + combinazione tipica** risulta efficace e trasparente per generare crossover musicali spiegabili. Con multilingua, feature audio e apprendimento dei pesi, il sistema può evolvere in uno strumento pratico di **curation** e **discovery** per playlist, editoria e creatività assistita, mantenendo tracciabilità delle scelte.



# BIBLIOGRAFIA / SITOGRAFIA

- [1] *UniTO Typst Template*. (2024). [Online]. Disponibile su: <https://github.com/eduardz1/UniTO-typst-template>
- [2] «Typst — A markup-based typesetting system». [Online]. Disponibile su: <https://typst.app/>
- [3] «Genius». [Online]. Disponibile su: <https://genius.com/>
- [4] «lyricsgenius — Genius API client for Python». [Online]. Disponibile su: <https://lyricsgenius.readthedocs.io/>
- [5] «NLTK — Natural Language Toolkit». [Online]. Disponibile su: <https://www.nltk.org/>
- [6] Helmut Schmid, «TreeTagger». [Online]. Disponibile su: <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/>
- [7] A. Valse, «CoCoS: uno strumento per la combinazione di concetti», 2020.

# RINGRAZIAMENTI

Desidero ringraziare innanzitutto il professor Gian Luca Pozzato per la disponibilità e per avermi dato la possibilità di lavorare su questo progetto interessante, nonostante i tempi stretti con le scadenze

Un ringraziamento profondo ai miei genitori, per il sostegno costante e la pazienza dimostrata in questi anni, soprattutto nei momenti in cui non ero

sicuro del percorso che stavo affrontando. La loro presenza mi ha permesso di superare le difficoltà e di arrivare a questo traguardo.

Ringrazio anche i miei amici e colleghi per i confronti, le risate e il supporto quotidiano, dentro e fuori l'università. Il loro incoraggiamento ha reso questo percorso più leggero e ricco di soddisfazioni.

Infine, grazie a tutte le persone che, anche solo con una parola o un gesto, mi hanno accompagnato lungo il cammino e mi hanno aiutato ad arrivare fin qui.