Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica

A.a. 2025/2026

Sessione di laurea Marzo 2026

Clustering e comprensione semantica di database relazionali:

un approccio graph-based con Graph Neural Networks e community detection

Relatori:

Silvia Anna Chiusano

Candidati:

Vito Silvestri

Sommario

La modernizzazione dei sistemi informativi aziendali, in particolare il passaggio da consolidate architetture monolitiche a modelli basati su microservizi, costituisce oggi una delle sfide più complesse nell'ambito dell'ingegneria del software. Molto spesso, i database relazionali sottostanti mancano di una documentazione aggiornata e sono caratterizzati da una fitta rete di dipendenze incrociate, il che rende estremamente difficile individuare i confini logici tra i vari domini applicativi.

Questo lavoro di tesi, sviluppato nell'ambito del progetto Dataslice AI, propone una pipeline innovativa per il *reverse engineering* e la modularizzazione automatica dei database relazionali. La soluzione modellizza lo schema dati come un grafo relazionale attribuito, fondendo la topologia strutturale (guidata dai vincoli di *Foreign Key*) con la ricchezza semantica contenuta nelle descrizioni delle tabelle e delle colonne.

Per estrarre e codificare il significato funzionale di ciascun nodo, sono stati impiegati Large Language Models (LLM) di ultima generazione, i cui output testuali sono stati proiettati in vettori continui (*text embedding*). Su questa base informativa ibrida sono state applicate architetture avanzate di *Deep Graph Clustering*, con particolare focus sui modelli neurali DMON e DGCluster. Al fine di valutare l'efficacia di tali reti rispetto ad approcci tradizionali (come l'euristica di Leiden o il K-Means puramente semantico), è stato definito un rigoroso framework di valutazione basato su metriche topologiche, metriche semantiche e metriche ibride di sintesi.

La campagna sperimentale è stata condotta su scenari di complessità crescente: da *toy dataset* a schemi ERP strutturati, fino ad arrivare a un caso di studio industriale reale e fortemente frammentato nel dominio assicurativo. I risultati sperimentali hanno messo in luce l'elevata efficacia delle Graph Neural Networks nel task di partizionamento. Queste architetture si sono dimostrate in grado di individuare in modo autonomo e accurato i macro-moduli funzionali del business, ottimizzando il compromesso tra coesione strutturale e affinità concettuale tra le tabelle e superando i limiti intrinseci degli approcci basati unicamente sulla topologia o esclusivamente sui contenuti semantici.

In conclusione, l'integrazione tra intelligenza artificiale generativa (LLM) e reti neurali su grafo (GNN) si conferma una tecnologia all'avanguardia e di inestimabile valore per l'analisi dei sistemi informativi opachi, ponendo solide basi per l'automazione del *refactoring* architetturale e la scomposizione dei database complessi.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

DatasliceAI

Nell'attuale scenario economico digitale, i dati si sono affermati come l'asset strategico fondamentale per il mantenimento del vantaggio competitivo. Tuttavia, la crescente digitalizzazione dei processi aziendali ha portato a un aumento esponenziale della quantità e complessità dei dati gestiti dalle organizzazioni. I grandi ecosistemi informativi di oggi sono costituiti da centinaia o persino migliaia di tabelle relazionali, spesso distribuite su diversi database, e affiancate da una documentazione incompleta, ridondante o ormai datata. In questo contesto, attività chiave come la business intelligence, l'analisi dei dati e la data governance richiedono una conoscenza approfondita delle strutture informative sottostanti. Questa conoscenza, però, è stata finora affidata a processi manuali lenti, costosi e altamente soggetti al rischio di errore.

Il progetto **Dataslice AI** nasce proprio per rispondere a questa esigenza: creare strumenti basati sull'intelligenza artificiale capaci di automatizzare la comprensione e l'analisi dei database relazionali, con l'obiettivo di affiancare data analyst, ingegneri e altre figure tecniche nelle attività di esplorazione, documentazione e interpretazione dei dati aziendali. L'iniziativa si inserisce nel sempre più ampio filone di ricerca dedicato all'utilizzo di modelli linguistici generativi (LLM) per il parsing, la semantizzazione e l'organizzazione di grandi basi di dati strutturati.

Il lavoro svolto in questa tesi si inserisce pienamente in questo contesto di analisi, focalizzandosi in modo specifico sulla fase di clustering del grafo di tabelle. Questo passaggio rappresenta uno snodo cruciale per l'intera pipeline di elaborazione: l'obiettivo non è semplicemente raggruppare oggetti, ma individuare automaticamente insiemi coerenti di tabelle che condividano una forte vicinanza semantica, un ruolo funzionale comune o strette relazioni logiche all'interno del database. Applicando tecniche di community detection alla topologia del grafo, dove le tabelle sono nodi e le relazioni sono archi, è possibile far emergere la struttura latente del sistema. Il clustering agisce come un filtro di astrazione, raggruppando entità che 'lavorano

insieme' per supportare specifici processi di business. Ciò permette, infine, di offrire a chi si approccia per la prima volta a database complessi o legacy una rappresentazione visiva immediata e intuitiva, ricostruendo di fatto il modello mentale originale con cui il sistema è stato progettato e implementato.

1.1 Criticità Strutturali dei Sistemi Informativi Aziendali

Come anticipato, nell'attuale scenario economico digitale, i dati costituiscono la risorsa strategica essenziale per preservare il vantaggio competitivo. Le aziende moderne si trovano a dover gestire, governare e valorizzare volumi di informazioni senza precedenti (Big Data), generati da un mosaico eterogeneo di sorgenti: dai sistemi transazionali core (OLTP), alle piattaforme di gestione integrata (ERP), fino ai sistemi di relazione con il cliente (CRM) e a innumerevoli applicazioni verticali specializzate. Tuttavia, la complessità di questi ecosistemi non deriva unicamente dalla vastità dei dati, ma soprattutto dalla loro varietà e dalla storia evolutiva dei sistemi stessi. La struttura complessiva dei database aziendali è raramente il frutto di un unico disegno architetturale coerente; al contrario, essa è spesso il risultato di una stratificazione sedimentata nel tempo. Anni di manutenzione, migrazioni tecnologiche parziali, fusioni societarie e integrazioni rapide di nuovi servizi hanno trasformato molti sistemi informativi in architetture ibride e complesse, difficili da mappare.

La letteratura scientifica e l'analisi industriale convergono nell'identificare quattro criticità strutturali che ostacolano l'efficace utilizzo di questi patrimoni informativi:

- **Disallineamento tra documentazione e implementazione (Schema Decay)**: Una delle problematiche più insidiose è l'obsolescenza della documentazione tecnica. Come evidenziato negli studi sulla software evolution, esiste spesso un divario crescente tra il modello concettuale originale e lo schema fisico attualmente in uso [1]. Le modifiche apportate "al volo" per soddisfare urgenze di business raramente vengono riportate nei dizionari dati, rendendo gli schemi non più rappresentativi dei processi aziendali reali e costringendo gli analisti a costose operazioni di reverse engineering.

- **Eterogeneità semantica e strutturale (Data Silos)**: La presenza di applicazioni eterogenee genera il problema dei "silos informativi". Ciascun sottosistema utilizza proprie convenzioni di denominazione, tipologie di dati e strutture relazionali, spesso tra loro incompatibili. Questa varietà, ampiamente discussa nell'ambito della Data Integration, complica la costruzione di

una visione complessiva e coerente [2]. In assenza di un livello di astrazione condiviso, le stesse entità di business possono essere modellate in modi profondamente diversi all'interno di un unico ecosistema aziendale.

- **Ridondanza e Debito Tecnico**: La stratificazione dei sistemi porta inevitabilmente all'accumulo di quello che Cunningham definì "Debito Tecnico". Nei database, questo si manifesta sotto forma di tabelle duplicate, colonne deprecate ma mai rimosse e vincoli di integrità logica gestiti via codice applicativo anziché a livello di DBMS. Tale ridondanza non solo spreca risorse di storage, ma complica la Data Governance, aumentando il rischio di disallineamento dei dati e incoerenza nelle reportistiche [3].
- **Insostenibilità dei costi di analisi manuale**: Infine, entra in gioco un problema di scala. Quando si ha a che fare con data warehouse o data lake che ospitano decine di migliaia di tabelle, un approccio manuale alla comprensione dello schema (schema understanding) diventa praticamente impossibile per una persona. Il carico cognitivo necessario per tenere a mente le dipendenze tra migliaia di entità eccede le capacità dei singoli esperti, facendo sì che i tempi di onboarding per nuovi sviluppatori o data scientist risultino estremamente lunghi e poco sostenibili.

Alla luce di queste criticità, l'automazione delle fasi di data discovery e di schema matching non rappresenta più una semplice funzionalità secondaria, ma diventa un requisito operativo imprescindibile. Progettare algoritmi in grado di dedurre automaticamente le relazioni, raggruppare entità semanticamente correlate e rendere visibile la struttura nascosta dei dati è l'unica strategia realmente praticabile per assicurare qualità del dato, efficienza nelle analisi e rapidità nei processi decisionali.

1.2 Il progetto Dataslice AI: obiettivi, architettura e pipeline

È proprio per rispondere a queste esigenze operative che nasce il progetto di Ricerca e Sviluppo **Dataslice AI**. L'iniziativa è stata concepita per affrontare la crescente complessità dei moderni ecosistemi informativi aziendali, nei quali i database relazionali assumono dimensioni sempre più estese e articolate. Tali sistemi risultano spesso scarsamente documentati e difficilmente interpretabili dal punto di vista funzionale, una problematica che emerge con forza soprattutto quando figure tecniche si interfacciano con lo schema per la prima volta.

In tali scenari, la sola analisi della struttura tecnica degli schemi SQL e l'utilizzo degli strumenti tradizionali messi a disposizione dai DBMS non risultano sufficienti a comprendere il reale significato delle entità, il ruolo delle relazioni tra le tabelle e

l'assetto logico del dominio di business. Questa limitazione comporta un notevole dispendio di tempo e risorse per software engineer e data engineer, costretti a ricostruire manualmente la semantica del sistema mediante analisi iterative e facendo leva, spesso in modo eccessivo, sulla propria conoscenza implicita.

Dataslice AI si propone di colmare questo divario introducendo un approccio automatico e *data-driven* alla comprensione dei database relazionali. Il progetto combina tecniche di Intelligenza Artificiale Generativa, rappresentazioni a grafo e algoritmi di analisi e clustering, con l'obiettivo di superare una visione puramente strutturale dello schema. L'idea centrale consiste nel trasformare un insieme di file SQL grezzi in una rappresentazione semantica strutturata, navigabile e interpretabile, in grado di supportare efficacemente attività strategiche quali l'esplorazione, la documentazione, il refactoring e la governance dei dati.

1.2.1 Obiettivi del progetto

L'obiettivo principale alla base del progetto **Dataslice AI** è ridurre in modo drastico il carico cognitivo e le latenze operative che, tradizionalmente, sono richieste per analizzare, comprendere e prendere in carico database relazionali complessi. Nel contesto classico, infatti, il processo di onboarding su un nuovo ecosistema dati costituisce un rilevante "collo di bottiglia": esso richiede settimane, se non mesi, di lavoro manuale ad alto consumo di risorse, fondato su tecniche artigianali di reverse engineering, sulla consultazione di documentazione frammentaria (spesso soggetta a schema decay) e sulla necessità di interazioni sincrone e ripetute con chi possiede la conoscenza pregressa del sistema (i cosiddetti "esperti di dominio").

In netta contrapposizione a questo paradigma manuale, Dataslice AI si prefigge di automatizzare la catena del valore della knowledge extraction, perseguendo quattro obiettivi specifici:

- **Automazione del Reverse Engineering Logico**: Il primo traguardo consiste nel superare la lettura sintattica degli script SQL per estrarne la struttura logica profonda. Il sistema mira a processare massivamente codice eterogeneo (DDL) per ricostruire deterministicamente le dipendenze, rendendo l'analisi indipendente dalla disponibilità di diagrammi ER aggiornati.

- **Colmare il "Semantic Gap"**: Gli schemi tecnici spesso utilizzano nomenclature criptiche o legacy che non riflettono il linguaggio del business. L'obiettivo è arricchire la struttura tecnica con un livello semantico generato dall'AI, fornendo descrizioni in linguaggio naturale che traducano i vincoli tecnici in regole di business comprensibili, colmando così il divario tra implementazione e dominio applicativo.

- **Rilevamento Automatico delle comunità**: Il progetto punta a individuare insiemi coerenti di tabelle (comunità) che mostrano alta coesione interna e basso accoppiamento esterno. L'identificazione di questi domini funzionali è cruciale per comprendere l'architettura modulare latente del sistema.

- **Diffusione Democratica del Sapere Architettonico**: Infine, si intende fornire una rappresentazione visiva, interattiva e condivisa dell'ecosistema dati. L'obiettivo è abbattere le barriere tecniche, rendendo la topologia del database accessibile non solo agli sviluppatori, ma anche agli stakeholder di business, facilitando un dialogo comune basato su una mappa univoca e aggiornata.

In una prospettiva di evoluzione dei sistemi informativi, Dataslice AI si configura come uno strumento di supporto alle moderne architetture dati decentralizzate, quali il Data Mesh. La possibilità di segmentare automaticamente un database monolitico in insiemi coerenti di tabelle riconducibili a domini funzionali distinti rappresenta un prerequisito essenziale per attività di micro-migrazione o processi di refactoring verso nuove architetture. Il contributo del progetto non si limita alla produzione di documentazione tecnica, ma consiste nella costruzione di una rappresentazione strutturata e interpretabile del database, in grado di trasformare lo schema SQL in una mappa concettuale dinamica. Tale rappresentazione consente una comprensione più rapida e sistematica del patrimonio informativo, favorendo un maggiore controllo e una più efficace gestione dei sistemi dati complessi.

1.2.2 Architettura e Pipeline di Elaborazione

Sotto il profilo tecnico-architetturale, Dataslice AI è progettato come un ecosistema integrato, in cui più componenti collaborano in modo coordinato per interpretare e rendere comprensibile la complessità dello schema. Per perseguire gli obiettivi delineati, il sistema impiega una pipeline di elaborazione *end-to-end* che converte progressivamente script SQL grezzi in artefatti visuali e in reportistica interattiva, secondo quanto rappresentato nello schema logico in Figura 1.1.
L'intero flusso si compone dei seguenti stadi principali, ognuno dei quali supportato da precise scelte tecnologiche:

1. **DDL Input**. La fase iniziale del workflow riguarda l'acquisizione e l'organizzazione degli script SQL da sottoporre ad analisi. In questo passaggio vengono raccolti tutti i file DDL (*Data Definition Language*) necessari a descrivere l'architettura del sistema informativo in esame. Tale attività è caratterizzata da un'elevata versatilità nella gestione dell'input: la pipeline è infatti progettata per accettare sia singoli file monolitici (dump completi), sia insiemi articolati di script suddivisi in più parti. Tutte queste sorgenti confluiscono in un'unica

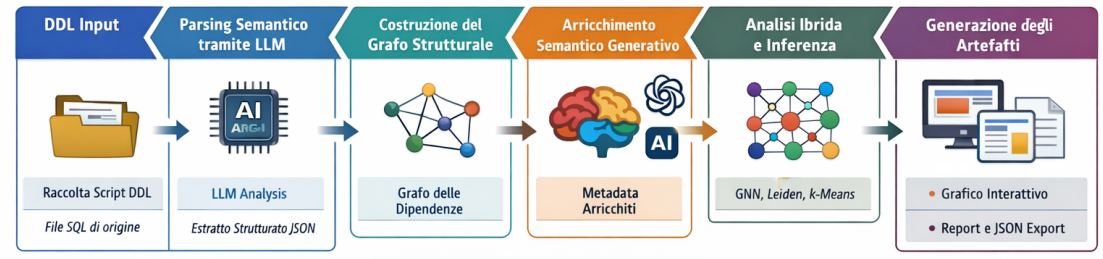

Figura 1.1: Schema logico della pipeline di elaborazione di Dataslice AI

directory di origine, che funge da punto di ingresso per il modulo di parsing semantico attivato nello stadio successivo.

2. **Parsing Semantico tramite LLM**. In questa fase, l'analisi dei file di origine viene delegata alle capacità interpretative del Large Language Model accessibile tramite le API di OpenAI. L'interazione con il modello è governata da una specifica strategia di *prompt engineering*: attraverso istruzioni mirate, si guida il sistema nell'esame dei file DDL raccolti, con l'obiettivo di separare le istruzioni di definizione strutturale dal contenuto non significativo. In particolare, il modello viene configurato per riconoscere ed estrarre in modo sistematico tutte le entità tabellari e le relazioni esplicite presenti nel codice. L'esito di questa elaborazione è un oggetto strutturato in formato JSON, che svolge il ruolo di livello intermedio di normalizzazione. Al suo interno sono organizzati gerarchicamente tutti i nodi individuati — arricchiti da metadati fondamentali, come il nome della tabella e l'elenco degli attributi (colonne) — insieme all'insieme delle relazioni topologiche derivate dai vincoli di chiave esterna (Foreign Key) rilevati. Questo approccio assicura una maggiore flessibilità e una più elevata tolleranza rispetto ad eventuali errori sintattici presenti nel codice legacy.

3. **Costruzione del Grafo Strutturale**. Una volta completata la fase di parsing, l'output strutturato in formato JSON viene ulteriormente elaborato per costruire il grafo delle dipendenze, utilizzando la libreria Python `networkx`. L'operazione si configura come una traduzione sistematica degli oggetti logici individuati: ogni entità tabellare contenuta nel file di interscambio viene rappresentata come un nodo della rete, mentre le relazioni di chiave esterna (*Foreign Keys*) vengono trasformate in archi non orientati. Questa procedura di mapping permette di ottenere una rappresentazione matematica rigorosa dello schema, ovvero un grafo connesso che riproduce in modo accurato la topologia strutturale del database ed è pronto per essere elaborato dai successivi algoritmi di analisi.

4. **Arricchimento Semantico Generativo**. Una volta consolidata la struttura topologica, la pipeline evolve verso l'arricchimento semantico del grafo, fase in cui si sfruttano nuovamente le capacità inferenziali dei Large Language Models tramite le API di OpenAI. Il processo avviene mediante una strategia di interrogazione iterativa: per ciascun nodo (tabella) del grafo, viene inviato un prompt specifico finalizzato a estrarre una comprensione profonda del dato. Il modello è istruito per restituire un output strutturato in formato JSON, popolando attributi informativi cruciali, come ad esempio:

   - `description`: una sintesi olistica che definisce cosa rappresenta l'entità nel mondo reale;
   - `columns description`: una disamina granulare del significato di ciascun attributo o colonna;
   - `business function`: l'identificazione del macro-processo aziendale o del dominio funzionale di appartenenza.

   Parallelamente all'analisi dei nodi, il modello generativo viene impiegato per esplicitare la semantica degli archi, fornendo descrizioni in linguaggio naturale delle relazioni logiche che intercorrono tra le diverse tabelle, trasformando così i vincoli tecnici in connessioni dotate di significato.

5. **Analisi Ibrida e Inferenza (GNN e Topologia)**. Prima di avviare la fase cruciale di inferenza, il sistema esegue uno step intermedio di trasformazione dei dati, con l'obiettivo di mappare le descrizioni in linguaggio naturale in uno spazio vettoriale latente. A questo scopo, l'architettura sfrutta l'ecosistema cloud di **Amazon Web Services (AWS)**, e in particolare il servizio gestito Bedrock. Mediante il modello `amazon.titan-embed-text-v2`, le descrizioni testuali prodotte nello stadio precedente vengono tradotte in embedding ad alta dimensionalità. Tali vettori vengono inseriti come attributi all'interno di ciascun nodo del grafo, ancorando l'informazione semantica alla struttura

topologica della tabella. Completata questa fase di arricchimento, l'oggetto grafo — ora dotato sia di struttura sia di rappresentazione semantica vettoriale — viene serializzato e archiviato su Amazon S3 (Simple Storage Service), assicurando la persistenza dei dati e rendendoli disponibili per il successivo stadio di addestramento e analisi.

L'ultima fase della pipeline sfrutta la duplice natura dei dati così ottenuti, integrando le informazioni strutturali con quelle semantiche. Per ottimizzare l'accuratezza nell'identificazione dei domini funzionali, il progetto fa uso di un approccio multimodale che applica in parallelo tre differenti strategie di clustering:

- **Graph Neural Networks (GNN)**: Addestrate mediante il servizio AWS SageMaker, queste reti neurali apprendono rappresentazioni (embedding) dei nodi che sintetizzano sia le feature semantiche interne sia le informazioni propagate dal vicinato topologico, catturando pattern complessi e non lineari.
- **Algoritmi Topologici**: Viene applicato l'algoritmo di Leiden (evoluzione dell'algoritmo di Louvain), focalizzato esclusivamente sulla massimizzazione della modularità e sulla densità dei collegamenti strutturali.
- **Clustering Semantico**: Viene utilizzato l'algoritmo *k-means* applicato puramente sugli embedding testuali generati da Titan, per raggruppare entità semanticamente simili a prescindere dalle loro connessioni dirette.

Tale metodologia comparativa permette non solo di ottenere le inferenze finali per l'assegnazione delle comunità, ma anche di confrontare e validare i cluster semantico-strutturali (GNN) rispetto a quelli derivanti da analisi mono-dimensionali, offrendo una visione olistica dell'architettura del database.

6. **Generazione degli Artefatti**. Al termine del flusso di elaborazione, gli output prodotti dai modelli di inferenza vengono aggregati e trasformati in artefatti informativi direttamente utilizzabili dall'utente finale. Il sistema coordina la generazione di diversi tipi di output, progettati per soddisfare molteplici modalità di fruizione:

   - **Visualizzazione Esplorativa**: Viene costruito un grafo interattivo, renderizzato in formato HTML, pensato come interfaccia di navigazione visuale che consente agli stakeholder di esplorare la topologia dei domini funzionali, osservando le relazioni sia a livello macro (cluster) sia a livello micro (singole tabelle).
   - **Documentazione e Interoperabilità**: In parallelo, il sistema produce una reportistica testuale approfondita e file di interscambio in formato JSON, esportabili e integrabili con altri strumenti.

Questa fase di output rappresenta il compimento del ciclo di vita del dato all'interno della pipeline, concretizzando il percorso di trasformazione che eleva lo script SQL grezzo iniziale allo stato di conoscenza strutturata e immediatamente utilizzabile a supporto dei processi decisionali.

Capitolo 2

Clustering sui grafi

2.1 Introduzione al clustering sui grafi

Nel paragrafo precedente è stata illustrata la pipeline complessiva del progetto Dataslice AI, evidenziando le diverse fasi che compongono il processo di comprensione automatica dei database relazionali. In questo capitolo si intende esaminare in modo approfondito uno degli elementi centrali che rappresentano il fulcro del contributo scientifico di questa tesi: il clustering delle tabelle mediante l'utilizzo dei grafi.

Sebbene le origini formali della teoria dei grafi vengano fatte risalire alla celebre risoluzione del problema dei ponti di Königsberg da parte di Eulero nel 1736, è solamente nel corso del XX secolo, e in particolare con l'avvento dell'era digitale, che tali strutture matematiche si sono affermate come il linguaggio universale per la rappresentazione e l'analisi dei sistemi complessi. La rivoluzione informatica ha infatti messo a disposizione degli studiosi una quantità di dati e risorse computazionali senza precedenti, permettendo di passare dallo studio di reti di piccole dimensioni all'analisi di sistemi composti da milioni o miliardi di nodi. In questo scenario, la necessità di gestire una tale mole di entità interconnesse ha prodotto un cambiamento radicale nell'approccio metodologico: le reti biologiche, sociali, tecnologiche e informative non vengono più studiate come oggetti statici, ma come sistemi dinamici le cui proprietà funzionali emergono dalla loro architettura topologica [4], [5].

Per comprendere in che misura tali teorie siano applicabili ai database relazionali è essenziale considerare la distinzione di fondo tra i modelli teorici di grafi casuali e le reti che osserviamo nel mondo reale. Il paradigma tradizionale di "grafo disordinato" è incarnato dal modello Random Graph proposto da Erdős e Rényi nel 1959, in cui la probabilità che esista un arco tra due nodi qualsiasi è costante e la distribuzione dei collegamenti risulta sostanzialmente uniforme. Le evidenze empiriche, tuttavia,

hanno mostrato che le reti utilizzate per rappresentare sistemi reali — come il World Wide Web, le reti di interazione tra proteine o, nello specifico, gli schemi di database aziendali — si discostano profondamente da questo schema di pura casualità. Esse costituiscono strutture in cui ordine e disordine coesistono, e sono caratterizzate da marcate eterogeneità strutturali. Nella maggior parte dei casi, la distribuzione del grado dei nodi segue una legge di potenza (power law): in altre parole, vi è un numero molto elevato di nodi con poche connessioni e una piccola quantità di "hub" con un numero di collegamenti estremamente elevato. Questa eterogeneità si manifesta non solo su scala globale ma anche locale, rivelandosi in una forte densità di archi all'interno di specifici insiemi di vertici e in una relativa scarsità di collegamenti fra insiemi differenti.[6].

Questa caratteristica topologica è definita in letteratura come Struttura a Comunità (Community Structure). In termini pratici, le comunità (o cluster) sono gruppi di nodi che risultano molto collegati tra loro e poco collegati con l'esterno: nel nostro caso, rappresentano insiemi di tabelle che lavorano insieme per svolgere una specifica funzione di business [7]].
L'esistenza di queste comunità non è casuale, ma rispecchia il modo naturale in cui sono organizzati i sistemi complessi, ovvero in gerarchie. Come osservato da Herbert Simon, i sistemi stabili tendono a dividersi in sottosistemi più piccoli e gestibili [8]:

- Un organismo biologico è fatto di organi, che sono fatti di tessuti, che sono fatti di cellule.
- Un'azienda è divisa in dipartimenti, che sono divisi in team.
- Allo stesso modo, un database complesso è composto da moduli logici (es. Vendite, Magazzino).

Identificare automaticamente questi moduli significa creare una mappa semplificata del sistema ("a grana grossa o coarse-grained). Questo processo di astrazione è fondamentale per chi deve analizzare il database: invece di perdersi in un caos di migliaia di tabelle singole, l'analista può ragionare per blocchi funzionali, trasformando un dato grezzo in una mappa concettuale chiara e navigabile.

Dal punto di vista applicativo, la Community Detection ha radici profonde che spaziano dalla sociologia, dove pionieri come Weiss e Jacobson già negli anni '50 cercavano di identificare gruppi di lavoro basandosi sulle matrici di relazione, fino all'informatica teorica, dove il problema del partizionamento dei grafi è stato storicamente affrontato per ottimizzare il calcolo parallelo e minimizzare le comunicazioni tra processori. Tuttavia, è con il contributo della fisica statistica e dei sistemi complessi, in particolare dopo il lavoro di Girvan e Newman del 2002, che il campo

ha visto una fioritura di nuovi algoritmi basati su misure di qualità globale della partizione, come la modularità [9]. Questi approcci moderni non richiedono di specificare a priori la dimensione o il numero dei gruppi, ma lasciano che sia la struttura stessa della rete a rivelare la sua organizzazione latente. Applicare queste metodologie a un database relazionale significa quindi superare la visione piatta dello schema fisico per far emergere, in modo data-driven, quei confini logici e quei contesti funzionali che erano presenti nella mente del progettista ma che sono andati perduti nella traduzione in codice SQL.

Parallelamente al consolidamento degli approcci basati sull'ottimizzazione della modularità, la ricerca scientifica degli ultimi anni ha esplorato nuove direzioni, convergendo verso il paradigma del *Graph Representation Learning*. In questo ambito, un ruolo di primo piano è stato assunto dalle Graph Neural Networks (GNN), modelli di deep learning progettati per operare su dati non euclidei. L'interesse verso queste tecniche deriva dalla loro capacità di gestire grafi ricchi di informazioni (*attributed graphs*): mentre gli algoritmi topologici tradizionali si focalizzano esclusivamente sulla matrice delle adiacenze (la struttura delle connessioni), le GNN permettono di elaborare congiuntamente anche le caratteristiche intrinseche dei nodi (feature o attributi). Attraverso meccanismi di aggregazione dell'informazione dal vicinato, questi modelli generano per ogni nodo un vettore numerico (*embedding*) che ne sintetizza sia la posizione strutturale sia il contenuto informativo. L'adozione di tali tecniche in questa tesi non mira a sostituire l'analisi topologica classica, bensì ad affiancarla, sfruttando la possibilità di fondere in un'unica rappresentazione vettoriale il rigore dei vincoli relazionali (Foreign Key) e la ricchezza semantica estratta dai Large Language Models.

2.2 Costruzione del grafo relazionale e definizione di Comunità

Per comprendere il processo di clustering implementato in questo lavoro di tesi, è necessario introdurre preliminarmente una definizione formale del concetto di *comunità* a partire dalla sua rappresentazione topologica. La Figura 2.1 fornisce un'astrazione visiva di un grafo caratterizzato da una marcata struttura modulare, nella quale è possibile individuare regioni topologiche ben definite. In particolare, tali regioni sono costituite da insiemi di nodi fortemente connessi tra loro, che formano aree ad alta densità di archi, separate da connessioni esterne più rare e sporadiche. Questi collegamenti, che agiscono come ponti o colli di bottiglia tra differenti porzioni del grafo, delineano in modo naturale i confini tra le comunità, rendendo evidente la distinzione tra struttura interna ed interazione esterna.

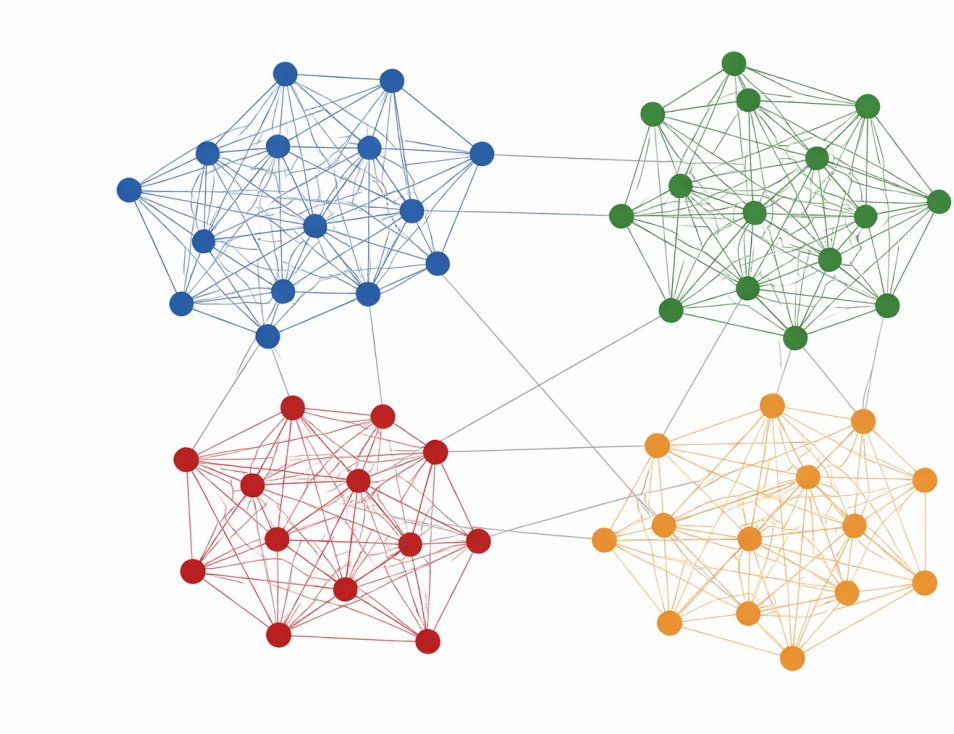

Figura 2.1: Rappresentazione schematica di un grafo con struttura a comunità. I nodi dello stesso colore appartengono al medesimo cluster, evidenziando un'alta densità di connessioni interne.

2.2.1 Definizione del grafo

Dal punto di vista matematico, un grafo (o rete) è definito come una coppia ordinata

$$G = (V, E)$$

dove V rappresenta l'insieme dei vertici (o nodi) ed E l'insieme degli archi (o collegamenti) che connettono coppie di vertici. Il concetto intuitivo di *cluster* o *comunità*, visibile in figura, può essere tradotto in una definizione formale basata sulla densità dei collegamenti. Come analizzato approfonditamente da Santo Fortunato nella sua rassegna sulla community detection, non esiste una definizione universale, ma la condizione più condivisa afferma che un sottoinsieme di nodi $C \subset V$ costituisce una comunità se la densità di archi interni al sottoinsieme (*intra-cluster density*) è significativamente superiore alla densità di archi che collegano i nodi di C con il resto del grafo (*inter-cluster sparsity*) [6]. In termini analitici, se consideriamo il grado k_i di un nodo come il numero di connessioni che esso possiede, la struttura a comunità emerge quando la probabilità che due nodi siano connessi

è alta se appartengono allo stesso gruppo e bassa se appartengono a gruppi diversi. Questa proprietà è fondamentale poiché, come evidenziato da Newman e Girvan, riflette l'organizzazione funzionale del sistema: i nodi all'interno di una comunità tendono a condividere proprietà comuni o a svolgere funzioni simili all'interno della rete globale [9, 5].

Nel contesto specifico del progetto Dataslice AI, questo formalismo matematico viene applicato direttamente all'architettura dei dati aziendali. La fase di costruzione del grafo non è, dunque, una mera rappresentazione astratta, ma il risultato deterministico del processo di parsing descritto nel capitolo precedente. Come anticipato nell'introduzione, l'analisi dei file DDL (Data Definition Language) viene demandata alle capacità interpretative dell'Intelligenza Artificiale Generativa (OpenAI API). L'LLM, istruito tramite tecniche di prompt engineering, agisce come un estrattore di entità e relazioni, restituendo un output strutturato in formato JSON che funge, di fatto, da lista di adiacenza per la costruzione del grafo. La traduzione dal modello relazionale al modello a grafo avviene secondo il seguente schema di mappatura:

- **I Nodi (V)**: Ogni entità tabellare individuata nel codice SQL viene istanziata come un vertice $v_i \in V$. Il nodo non è un'entità puntiforme priva di contenuto, ma porta con sé attributi estratti dal parser (nome della tabella, lista delle colonne).
- **Gli Archi (E)**: Le relazioni che intercorrono tra le tabelle vengono modellate come archi diretti $e_{ij} = (v_i, v_j)$. Nello specifico, la presenza di un arco riflette l'esistenza di un vincolo di integrità referenziale (*Foreign Key*) dove la tabella v_i fa riferimento alla tabella v_j.

L'oggetto matematico che si ottiene è un grafo diretto $G_{DB} = (Tables, FKs)$, che rende esplicita la struttura schematica del database. È proprio su questa topologia che gli algoritmi di clustering andranno a individuare quelle concentrazioni locali di collegamenti (le comunità) che, nel dominio applicativo, si traducono nei moduli funzionali del sistema informativo.

2.2.2 Definizione del Grafo con Attributi

Tuttavia, la rappresentazione limitata alla coppia (V, E) risulta sufficiente solo per gli approcci puramente topologici (come l'algoritmo di Leiden descritto in seguito), ma appare riduttiva per l'implementazione di tecniche avanzate di *Deep Learning* su grafo. Uno schema di database, infatti, non è costituito solo da connessioni strutturali: ogni tabella porta con sé un ricco corredo semantico — il nome dell'entità, i nomi degli attributi, i tipi di dato e i commenti descrittivi —

che è cruciale per inferirne il ruolo funzionale. Per abilitare l'utilizzo delle Graph Neural Networks (GNN), è stato necessario estendere il formalismo matematico introducendo il concetto di Grafo con Attributi (*Attributed Graph*).

In questa estensione, il grafo viene ridefinito come una tripla:

$$G = (V, E, X)$$

dove X rappresenta la Matrice delle Feature (o degli attributi). Se indichiamo con $N = |V|$ il numero di nodi (tabelle) e con d la dimensione dello spazio delle caratteristiche, allora $X \in \mathbb{R}^{N \times d}$ è una matrice in cui la riga i-esima, indicata con $\mathbf{x}_i$, corrisponde al vettore delle feature associato al nodo v_i. Nel contesto specifico di questo lavoro, la costruzione della matrice X è un passaggio fondamentale che trasforma i metadati testuali in input numerici processabili dalla rete neurale. Ogni vettore $\mathbf{x}_i$ non contiene valori arbitrari, ma è il risultato di un processo di *embedding* semantico: le informazioni estratte dall'LLM per la tabella i (nome, colonne, descrizione) vengono convertite in un vettore denso a d dimensioni che ne cattura il significato. Di conseguenza, l'input fornito ai modelli GNN implementati in questa tesi non sarà limitato alla sola struttura delle Foreign Key (A, matrice di adiacenza derivata da E), ma comprenderà anche il contenuto semantico (X), permettendo all'algoritmo di apprendere sia dalla topologia dello schema sia dalla semantica del dominio.

2.3 Cenno alle tecniche di clusterizzazione del grafo

Una volta definito formalmente il concetto di comunità come sottografo caratterizzato da un'elevata densità interna, il problema computazionale si sposta sulla ricerca algoritmica della partizione ottimale dei nodi V. In letteratura, il Graph Clustering costituisce una sfida algoritmica di notevole spessore: determinare la partizione esatta che massimizza una metrica di qualità globale (come la modularità) è formalmente classificato come un problema NP-hard [6]. Ciò implica che il numero di possibili partizioni di un insieme di nodi cresce più che esponenzialmente rispetto alla dimensione del grafo, rendendo impraticabile qualsiasi approccio di forza bruta (brute-force) anche per reti di dimensioni modeste. Di conseguenza, la ricerca scientifica si è orientata quasi esclusivamente verso lo sviluppo di euristiche e algoritmi di approssimazione.

2.3.1 Graph Partitioning e Community Detection

Prima di procedere con la classificazione delle diverse tecniche utilizzate, è tuttavia necessario introdurre una distinzione concettuale essenziale tra il tradizionale *Graph*

Partitioning e la più recente *Community Detection*.

Graph Partitioning

Il *Graph Partitioning* affonda le sue radici nell'informatica teorica e nell'ingegneria dei sistemi paralleli. Il problema classico consiste nel suddividere i nodi di un grafo in k gruppi di dimensioni predeterminate, spesso uguali, minimizzando il numero di archi che collegano gruppi diversi. Questo approccio è tipicamente imposto dall'esterno e nasce con l'obiettivo di ottimizzare l'utilizzo di una risorsa computazionale, ad esempio distribuendo in modo bilanciato il carico di lavoro su k processori in architetture parallele o riducendo la comunicazione inter-processo. Dal punto di vista formale, il problema può essere modellato come un'ottimizzazione vincolata, in cui si ricerca una partizione del grafo che minimizzi una funzione di costo basata sui tagli tra i cluster, soggetta a vincoli sulle dimensioni dei gruppi. In questo contesto, il numero di cluster k è fissato e noto a priori, mentre le cardinalità dei cluster sono spesso vincolate per garantire una distribuzione omogenea dei nodi.

Una caratteristica fondamentale di questo approccio è che l'obiettivo di minimizzazione dei tagli può prevalere sulla preservazione della struttura locale del grafo. Di conseguenza, nodi fortemente connessi o strutturalmente affini possono essere separati qualora ciò sia necessario per soddisfare i vincoli dimensionali imposti. Tale comportamento rende il *Graph Partitioning* particolarmente adatto a scenari in cui l'equilibrio delle partizioni rappresenta un requisito primario, ma meno indicato per applicazioni in cui si desidera identificare strutture emergenti o comunità naturali all'interno del grafo. È inoltre noto che il problema del *Graph Partitioning* è computazionalmente complesso: nella sua formulazione generale risulta essere un problema NP-difficile. Per questo motivo, nella pratica vengono impiegate euristiche e algoritmi approssimati, come ad esempio metodi basati su rilassamenti spettrali, approcci multilevel o tecniche greedy, che consentono di ottenere soluzioni di buona qualità in tempi computazionalmente sostenibili.

Community Detection

Al contrario, la Community Detection nasce con un approccio orientato alla "scoperta" (discovery-driven). In questo scenario, si assume che la rete possieda una struttura organizzativa intrinseca e latente che l'algoritmo deve rivelare, senza imposizioni esterne. Nel contesto dell'analisi di database e dei sistemi informativi reali, l'approccio del partizionamento risulta inadeguato: non è possibile sapere a priori quanti domini funzionali compongano un database (il k è ignoto) e, soprattutto, i domini funzionali reali sono intrinsecamente eterogenei (un modulo "Core" potrebbe contenere centinaia di tabelle, mentre un modulo "Configurazioni" solo poche unità). Pertanto, l'attenzione di questa ricerca si focalizza esclusivamente

sulla Community Detection, intesa come l'insieme di metodologie che lasciano emergere spontaneamente sia la cardinalità delle comunità sia la loro distribuzione dimensionale, spesso sbilanciata (power-law distribution) [7].

Data la complessità intrinseca del problema, nel corso degli ultimi decenni la letteratura scientifica ha prodotto un ventaglio eterogeneo di strategie algoritmiche che rientrano nell'ambito della *community detection*. Di seguito si propone un approfondimento delle principali famiglie metodologiche:

- **Metodi Divisivi e Gerarchici**: Questa classe di algoritmi, il cui esponente più celebre è il metodo di Girvan-Newman, approccia il problema con una modalità top-down. L'idea centrale si basa sul concetto di *Edge Betweenness*, ovvero gli archi che collegano comunità diverse agiscono come "ponti" e sono attraversati da un numero elevato di cammini minimi tra coppie di nodi. L'algoritmo procede rimuovendo iterativamente gli archi con la betweenness più alta, frammentando progressivamente la rete in componenti connesse isolate che corrispondono alle comunità. Sebbene teoricamente eleganti, questi metodi soffrono di un'elevata complessità computazionale ($O(m^2n)$), che li rende poco scalabili su grandi database.

- **Metodi Spettrali (Spectral Clustering)**: Questi approcci trasformano il problema combinatorio discreto del clustering in un problema di algebra lineare continuo. Sfruttando le proprietà spettrali delle matrici associate al grafo (in particolare la *Matrice Laplaciana*), l'algoritmo calcola i primi k autovettori per proiettare i nodi in uno spazio vettoriale a bassa dimensionalità. In questo nuovo spazio, i cluster topologici risultano geometricamente separati e possono essere identificati mediante tecniche di clustering standard come il K-means. Il limite principale risiede nell'alto costo computazionale della diagonalizzazione di grandi matrici.

- **Modelli basati sull'Inferenza Statistica**: Rappresentati in particolare dagli *Stochastic Block Models* (SBM), questi approcci seguono una logica generativa. Si ipotizza che il grafo osservato derivi da un modello probabilistico latente, in cui la probabilità che esista un arco tra due nodi dipende unicamente dal gruppo di appartenenza dei nodi stessi. Il compito dell'algoritmo è dunque quello di "invertire" il processo generativo: stimare i parametri del modello (cioè le assegnazioni dei nodi ai gruppi) che massimizzano la verosimiglianza (*likelihood*) di osservare il grafo dato. Si tratta di un'impostazione matematicamente rigorosa, capace di individuare strutture anche molto articolate, ma spesso complessa da impostare e calibrare.

- **Approcci Dinamici (Random Walks)**: Questa classe di metodi, che comprende algoritmi come *Walktrap* e *InfoMap*, definisce le comunità a partire da

un processo dinamico di diffusione sul grafo. L'idea di base è che un "camminatore casuale" (**random walker**) che si sposta sulla rete tenda a rimanere per tempi relativamente lunghi all'interno di regioni con elevata densità di archi (le comunità), uscendo solo occasionalmente tramite i pochi collegamenti verso l'esterno. Analizzando le probabilità di transizione oppure comprimendo la descrizione della traiettoria del camminatore (come fa InfoMap), si può ricostruire la struttura modulare della rete.

2.3.2 Approcci adottati: Analisi Topologica vs Graph Representation Learning

Di fronte all'ampio spettro di algoritmi appena illustrati, questa tesi compie una scelta metodologica ben definita. Nonostante la grande varietà delle tecniche disponibili, per lo sviluppo di questo lavoro si è optato di restringere il campo di analisi, concentrandosi su due macro-categorie metodologiche distinte. Tale selezione non è casuale, ma mira a contrapporre e mettere in dialogo due paradigmi filosoficamente differenti nell'analisi dei grafi: da un lato l'approccio puramente strutturale (o topologico), dall'altro l'apprendimento automatico su grafi (Deep Learning).

Di seguito si esplicita la suddivisione concettuale adottata come riferimento per il prosieguo della discussione:

1. **Metodi Topologici (o Strutturali)**: In questa categoria rientrano gli algoritmi "classici" di Community Detection (inclusi quelli basati su modularità e i metodi spettrali menzionati in precedenza). Tali algoritmi agiscono esclusivamente sulla matrice delle adiacenze A del grafo, basando quindi le proprie decisioni unicamente sulla presenza o assenza di archi tra coppie di nodi. L'ipotesi fondamentale è che l'intera informazione necessaria alla definizione dei raggruppamenti sia codificata nella topologia della rete. Questi metodi risultano particolarmente efficaci nell'individuare insiemi coesi fondati su vincoli rigidi (nel nostro caso, le chiavi esterne), ma rimangono sostanzialmente "ignari" del contenuto informativo e degli attributi associati ai nodi.

2. **Metodi Neurali basati su GNN (Graph Neural Networks)**: Questa seconda famiglia rappresenta l'evoluzione contemporanea del clustering, riconducibile all'area del *Graph Representation Learning*. A differenza dei metodi meramente topologici, tali approcci integrano in modo congiunto due sorgenti informative: la struttura del grafo (matrice A) e gli attributi dei nodi (matrice delle feature X). Mediante l'impiego di reti neurali profonde, essi mappano i nodi in uno spazio vettoriale latente (*embedding space*), in cui il clustering viene effettuato raggruppando vettori prossimi non solo dal punto di vista topologico, ma anche da quello semantico [10].

La scelta di trattare in modo distinto e di mettere a confronto queste due famiglie metodologiche nasce da un'esigenza strettamente legata alla natura del dato relazionale. Lo schema di un database è infatti un'entità intrinsecamente duale: da una parte presenta una struttura rigida e deterministica (i vincoli di chiave esterna), che gli algoritmi topologici sono in grado di catturare in maniera particolarmente efficace; dall'altra racchiude una forte componente semantica (nomi delle tabelle, commenti, tipi di dato) che sfugge completamente a una lettura puramente strutturale. Se un metodo topologico "vede" soltanto nodi anonimi connessi da archi, un approccio neurale ha invece la capacità di comprendere il contenuto informativo di ciascun nodo. La sfida scientifica, che sarà approfondita nei capitoli successivi, consiste proprio nel combinare l'affidabilità derivante dalla struttura topologica con la ricchezza espressiva offerta dalla dimensione semantica.

Capitolo 3

Metodi topologici per il clustering

La definizione dello stato dell'arte nella comunità scientifica nel campo del clustering sui grafi, presentata nel capitolo precedente, permette ora di introdurre la prima famiglia di approcci su cui si basa questo lavoro: i metodi di natura topologica. La decisione di dedicare un capitolo specifico all'analisi esclusivamente strutturale non è arbitraria, ma deriva da una considerazione peculiare riguardante il contesto dei database relazionali. A differenza delle reti sociali o biologiche, in cui le connessioni possono essere incerte o rumorose, in un database i legami tra entità sono regolati da vincoli di integrità referenziale (*Foreign Keys*) rigidamente definiti. Tali vincoli costituiscono l'impalcatura logica concepita dagli sviluppatori per assicurare la coerenza dei dati. Ne consegue che l'assunzione di fondo che orienta questo capitolo è che la topologia della rete — cioè la struttura delle connessioni tra tabelle — contenga già un segnale forte e informativo per l'identificazione dei domini funzionali.

Il capitolo persegue un duplice scopo. In primo luogo, si intende fornire le basi teoriche necessarie per comprendere come l'idea informale di "insieme di tabelle collegate" venga tradotta in termini matematici tramite il concetto di Modularità. In secondo luogo, verrà ripercorso il percorso evolutivo degli algoritmi che ha condotto all'adozione del metodo di Leiden come nucleo di clustering per il progetto Dataslice AI, descrivendone infine gli aspetti implementativi e le librerie software impiegate.

3.1 Funzione obiettivo: Ottimizzazione della Modularità

La prima famiglia metodologica esaminata raccoglie gli approcci che identificano le comunità facendo riferimento unicamente alla topologia della rete, cioè alla disposizione degli archi che collegano i nodi. In questo quadro, il problema della Community Detection viene formulato come un problema di ottimizzazione: lo scopo è trovare la partizione dei nodi che massimizza una determinata funzione di costo o di qualità. Stabilire che cosa sia una partizione "buona" è però tutt'altro che immediato. Una semplice valutazione della densità delle connessioni non basta: in un grafo molto denso, individuare sottogruppi con numerosi archi potrebbe riflettere un artefatto statistico piuttosto che una reale organizzazione strutturale. Per superare questa ambiguità, è necessario confrontare la struttura osservata con un modello di riferimento. Tra le molteplici metriche di qualità proposte in letteratura, la più riconosciuta e largamente utilizzata è la Modularità (Q), introdotta da Newman e Girvan nel loro lavoro pionieristico [9].
L'intuizione brillante alla base della modularità risiede nell'uso di un Modello Nullo come termine di paragone. L'idea centrale è che una struttura a comunità sia significativa solo se la densità di collegamenti all'interno dei gruppi è superiore a quella che si osserverebbe in una rete casuale equivalente. Nello specifico, la modularità quantifica la differenza tra:

- La frazione di archi che cadono all'interno delle comunità date;
- La frazione di archi che ci si aspetterebbe di trovare all'interno delle stesse comunità se i collegamenti fossero distribuiti in modo puramente casuale, pur mantenendo inalterato il grado di ciascun nodo.

In altre parole, Q misura quanto la struttura a cluster del grafo si discosta dalla casualità. Se i collegamenti interni superano l'aspettativa statistica del modello nullo, allora la partizione ha intercettato una reale organizzazione modulare del sistema. Formalmente, data una rete partizionata, la modularità Q è definita come:

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) \tag{3.1}$$

dove:

- A_{ij} rappresenta l'elemento della matrice di adiacenza (1 se esiste un arco tra i e j, 0 altrimenti);
- k_i e k_j sono i gradi dei nodi i e j;

- m è il numero totale di archi nel grafo ($m = \frac{1}{2}\sum_i k_i$);
- $\frac{k_i k_j}{2m}$ rappresenta la probabilità attesa di un collegamento tra i e j nel modello nullo (grafo casuale con stessa distribuzione di grado);
- $\delta(c_i, c_j)$ è la funzione delta di Kronecker, che vale 1 se i nodi appartengono alla stessa comunità ($c_i = c_j$) e 0 altrimenti.

Il valore di Q varia tra -1 e 1. Valori positivi indicano una struttura a comunità marcata, mentre valori prossimi allo zero suggeriscono che la partizione non differisce da una distribuzione casuale.

3.2 L'Algoritmo di Louvain

Una volta definita matematicamente la funzione obiettivo Q, il problema si sposta sul piano computazionale: come individuare la configurazione specifica di nodi e comunità che ne massimizza il valore. Come già ricordato, l'analisi esaustiva di tutte le possibili partizioni costituisce un problema NP-hard, e risulta quindi impraticabile per reti di dimensioni realistiche. Per questo motivo l'attenzione si è spostata su algoritmi euristici di tipo approssimativo. Tra questi, l'Algoritmo di Louvain, introdotto da Blondel et al. nel 2008 [11], si è affermato per oltre un decennio come standard di riferimento, grazie alla sua notevole efficienza computazionale ($O(n \log n)$). Il metodo adotta una strategia *greedy* e gerarchica, articolata in due fasi che vengono ripetute iterativamente, come mostrato schematicamente nella Figura 3.1.

1. **Ottimizzazione locale (Modularity Optimization)**: Ogni nodo viene inizialmente considerato come una comunità a sé stante. L'algoritmo esamina sequenzialmente ogni nodo i e valuta il guadagno di modularità ΔQ che si otterrebbe rimuovendo i dalla sua comunità attuale e inserendolo nella comunità di uno dei suoi vicini j. Il guadagno ΔQ derivante dallo spostamento di un nodo isolato i in una comunità C è calcolato analiticamente come:

$$\Delta Q = \left[\frac{\Sigma_{in} + 2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \quad (3.2)$$

   dove:

   - Σ_{in} è la somma dei pesi degli archi interni a C
   - Σ_{in} è la somma dei pesi degli archi interni a C

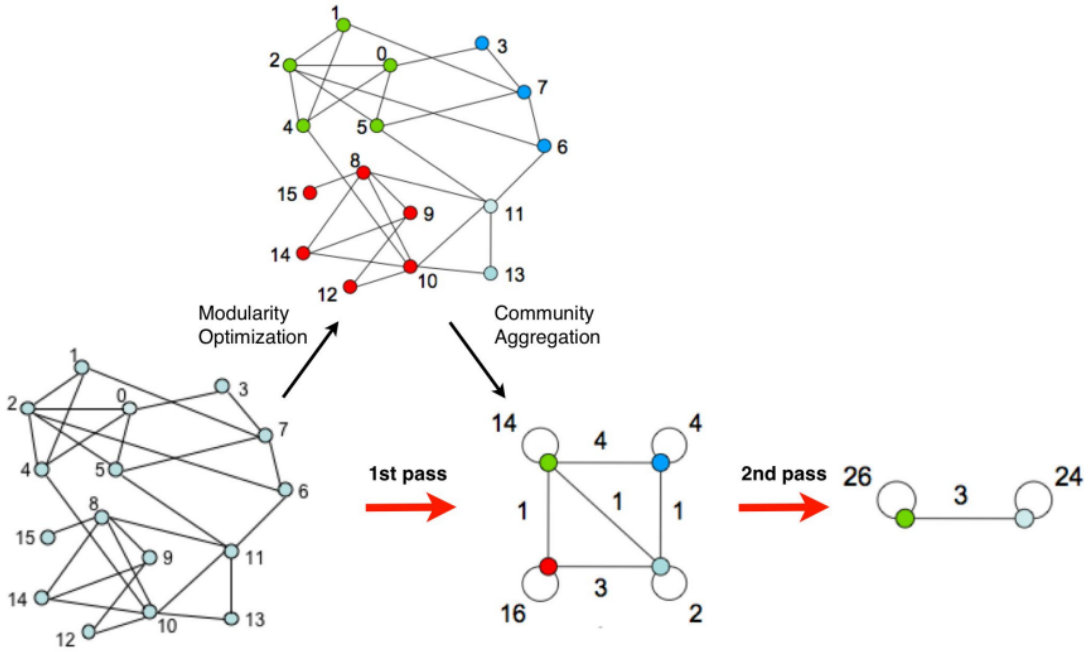

Figura 3.1: Visualizzazione schematica dell'algoritmo di Louvain. Il processo procede per iterazioni successive (Passi). Ogni passo è composto da due fasi distinte: (1) *Ottimizzazione della Modularità*, in cui i nodi vengono raggruppati in comunità locali; (2) *Aggregazione*, in cui le comunità trovate vengono compresse in singoli super-nodi per costruire un nuovo grafo pesato di livello superiore. L'algoritmo termina quando non sono possibili ulteriori incrementi della modularità. (Adattato da [11]).

- Σ_{tot} è la somma dei pesi degli archi incidenti ai nodi di C
- k_i è il grado del nodo i
- $k_{i,in}$ è la somma dei pesi degli archi tra i e gli altri nodi di C
- m è la somma totale dei pesi nel grafo

Il nodo viene assegnato alla comunità che massimizza questo incremento positivo; se nessun spostamento produce un guadagno, il nodo rimane nella sua comunità originaria.

2. **Aggregazione della rete (Community Aggregation)**: Una volta raggiunto l'ottimo locale (nessun nodo può essere spostato per migliorare Q), si costruisce un nuovo grafo ("meta-grafo"). I nodi di questo nuovo livello rappresentano le comunità trovate nella fase precedente. Gli archi interni alle comunità vengono trasformati in *self-loops* (archi che collegano il nodo a se

stesso), mentre gli archi tra nodi di comunità diverse vengono aggregati in un unico arco pesato che collega i nuovi super-nodi.

Questi due passaggi costituiscono un "passo" dell'algoritmo. Il processo viene reiterato sul meta-grafo finché la modularità non può più essere incrementata, generando una struttura gerarchica di comunità.

3.2.1 Limitazioni strutturali e criticità dell'algoritmo

Nonostante l'algoritmo di Louvain abbia rappresentato per anni lo stato dell'arte grazie alla sua velocità, l'applicazione estensiva su reti complesse ha fatto emergere diverse criticità strutturali che ne limitano l'affidabilità, specialmente in contesti sensibili come l'analisi di schemi di database.

Il problema delle comunità disconnesse

Il difetto più grave, identificato formalmente da Traag et al., riguarda la tendenza di Louvain a generare comunità internamente disconnesse o scarsamente connesse [12]. Ciò accade perché l'algoritmo sposta un nodo basandosi solo sul guadagno globale di modularità, senza verificare se il nodo funge da "ponte" essenziale all'interno della sua vecchia comunità. Rimuovendo un nodo-ponte, la vecchia comunità può spezzarsi in due componenti isolate che però, per l'algoritmo, continuano a far parte della stessa etichetta di gruppo. Nel contesto di Dataslice AI, questo è un problema bloccante: un cluster di tabelle deve rappresentare un dominio funzionale navigabile tramite join. Avere tabelle raggruppate insieme ma non collegate da alcun percorso di foreign key renderebbe il cluster semanticamente invalido.

Instabilità e dipendenza dall'ordine

Essendo un algoritmo *greedy*, il risultato finale dipende dall'ordine in cui i nodi vengono esaminati durante la fase di ottimizzazione locale. Sebbene l'impatto sulla modularità finale sia spesso trascurabile, la composizione esatta delle comunità può variare tra diverse esecuzioni. In un contesto aziendale, dove la riproducibilità dell'analisi è fondamentale, questa instabilità rappresenta un fattore di rischio.

Limite di Risoluzione (Resolution Limit)

Uno dei limiti teorici più noti della modularità è il cosiddetto *resolution limit*. Questo fenomeno indica l'incapacità della funzione di modularità di identificare comunità molto piccole all'interno di grafi di grandi dimensioni. Poiché l'ottimizzazione della modularità valuta la qualità della partizione su scala globale, piccoli gruppi di nodi fortemente connessi tra loro possono risultare invisibili all'algoritmo e venire incorporati in comunità più grandi.

Di conseguenza, Louvain eredita questo limite intrinseco della modularità. In grafi molto estesi, l'algoritmo tende a inglobare piccole comunità ben definite all'interno di cluster più grandi, oscurando la struttura fine del database e potenzialmente fondendo domini funzionali distinti.

3.3 L'algoritmo di Leiden

Al fine di superare le criticità strutturali emerse con l'utilizzo di Louvain, e in particolare la tendenza alla generazione di partizioni internamente disconnesse, la comunità scientifica ha recentemente proposto una nuova strategia risolutiva. Nel 2019, Traag, Waltman e van Eck hanno introdotto l'Algoritmo di Leiden [12], dimostrando formalmente come l'approccio classico di aggregazione greedy fosse insufficiente per garantire la coerenza topologica dei cluster.
Leiden non si limita a essere una semplice ottimizzazione del suo predecessore, ma ne rivede l'architettura logica introducendo un meccanismo di controllo più sofisticato durante il processo di formazione delle comunità. Mentre Louvain si basa su un ciclo a due fasi (ottimizzazione locale e aggregazione), Leiden inserisce una fase intermedia cruciale denominata raffinamento locale (*refinement phase*), modificando il modo in cui i nodi vengono predisposti per l'aggregazione.

L'adozione di questo algoritmo come motore topologico nella pipeline di Dataslice AI non è casuale, ma risponde a tre esigenze specifiche imposte dalla natura dei dati relazionali:

1. **Garanzia di Connessione**: Il vantaggio primario di Leiden risiede nella sua capacità di garantire matematicamente che tutte le comunità prodotte siano connesse. A differenza di Louvain, che può raggruppare nodi isolati tra loro solo perché ciò massimizza la modularità globale, Leiden assicura che esista sempre un percorso di archi che collega ogni coppia di nodi all'interno di un cluster. Nel contesto di un database, questo è fondamentale: assicura che ogni cluster di tabelle individuato rappresenti un sottografo coerente e navigabile tramite join.

2. **Migliore Risoluzione**: Grazie alla fase di raffinamento, l'algoritmo è in grado di scoprire partizioni sub-ottimali che Louvain avrebbe ignorato. Questo mitiga gli effetti del limite di risoluzione (resolution limit) tipico della modularità, permettendo di identificare domini funzionali più granulari e coesi, evitando l'inglobamento di piccole strutture funzionali in macro-comunità prive di significato semantico specifico.

3. **Efficienza Computazionale**: Nonostante la maggiore complessità logica dovuta all'introduzione del raffinamento, Leiden converge significativamente

più velocemente rispetto a Louvain. L'algoritmo ottimizza le visite ai nodi, riducendo il numero di iterazioni necessarie per raggiungere la stabilità. Questo rende l'analisi scalabile anche su schemi di database estremamente complessi composti da migliaia di tabelle.

In sintesi, l'adozione di Leiden costituisce la scelta metodologica di riferimento per la componente puramente topologica di questo lavoro, offrendo il miglior compromesso attuale tra rigore formale (connessione garantita) ed efficienza operativa.

3.3.1 Algoritmo e fase di raffinamento

Come già messo in luce, il principale limite dell'approccio di Louvain è legato alla natura irreversibile della fase di aggregazione, che tende a fissare partizioni non ottimali trasformandole in super-nodi monolitici. L'algoritmo di Leiden affronta questa criticità introducendo, tra l'ottimizzazione locale e l'aggregazione, una procedura intermedia essenziale: la fase di raffinamento (*Refinement Phase*). L'idea di fondo di questo passaggio è che non tutte le comunità individuate in prima istanza siano effettivamente pronte per essere aggregate. Alcune potrebbero risultare troppo grossolane o includere sottostrutture che sarebbe preferibile mantenere separate. Per cogliere appieno il contributo innovativo di Leiden, è tuttavia utile esaminare il flusso di esecuzione dell'algoritmo attraverso la sua rappresentazione schematica. La Figura 3.2 mette a confronto le diverse fasi logiche che conducono dalla partizione iniziale alla costruzione del grafo aggregato.

Come si può notare dall'immagine l'algoritmo procede per iterazioni successive seguendo tre step fondamentali.

Spostamento locale dei nodi (Local Move): Come illustrato nel passaggio da (a) a (b), l'algoritmo esegue inizialmente una scansione rapida simile a quella di Louvain. I nodi vengono spostati nelle comunità vicine per massimizzare la modularità. Questa fase produce una partizione preliminare (indicata dall'ellisse grande nella figura) che tuttavia, come discusso, potrebbe nascondere disconnessioni interne.

Fase di Raffinamento (Refinement Phase): È qui, nel passaggio verso la configurazione (c), che risiede il cuore dell'innovazione. L'algoritmo non accetta la partizione (b) come definitiva per l'aggregazione, ma avvia una procedura di verifica interna.

Operativamente, come mostrato nello schema, il raffinamento riparte da una configurazione a nodi singoli, in cui all'interno di ogni macro-comunità identificata, ogni nodo viene considerato momentaneamente isolato. A partire da questa granularità fine, l'algoritmo procede iterativamente a unire i nodi in nuove sotto-comunità (le aree rosse e verdi distinte all'interno dell'ellisse), applicando però un vincolo

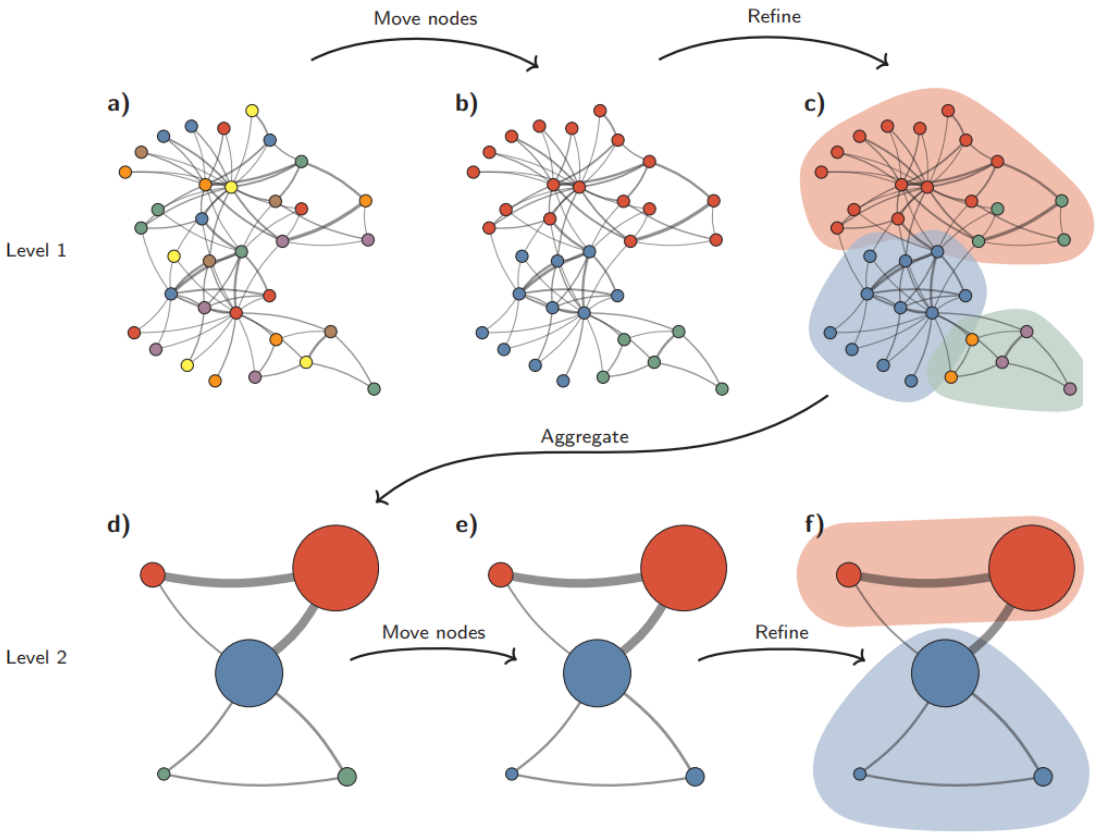

Figura 3.2: Fasi principali dell'algoritmo di Leiden per il rilevamento delle comunità. L'algoritmo di Leiden parte da una partizione singleton (a). L'algoritmo sposta i singoli nodi da una comunità all'altra per trovare una partizione (b), che viene poi perfezionata (c). Viene creata una rete aggregata (d) basata sulla partizione perfezionata, utilizzando la partizione non perfezionata per creare una partizione iniziale per la rete aggregata. Ad esempio, la comunità rossa in (b) viene raffinata in due sottocomunità in (c), che dopo l'aggregazione diventano due nodi separati in (d), entrambi appartenenti alla stessa comunità. L'algoritmo sposta quindi i singoli nodi nella rete aggregata (e). In questo caso, il raffinamento non modifica la partizione (f). Questi passaggi vengono ripetuti fino a quando non è possibile apportare ulteriori miglioramenti. (Adattato da [12]).

topologico rigoroso: un nodo può essere fuso con un gruppo solo se entrambi appartengono alla stessa comunità identificata nella fase precedente (b). Questa restrizione impedisce che il raffinamento alteri i macro-confini già stabiliti, ma consente di frammentare internamente una comunità qualora essa non risulti

sufficientemente coesa. In tal modo, un cluster che in Louvain sarebbe stato forzatamente trattato come un blocco unico, in Leiden viene scisso in più sottogruppi distinti e garantiti connessi.

Un ulteriore elemento distintivo di questa fase, non visibile staticamente ma cruciale per la dinamica, riguarda il criterio decisionale di fusione. A differenza dell'approccio deterministico greedy della fase 1, il raffinamento adotta una logica stocastica basata su una distribuzione di probabilità. La scelta di unire un nodo a una comunità raffinata avviene con una frequenza proporzionale all'incremento di qualità previsto (spesso definito tramite una funzione esponenziale del guadagno di modularità). L'introduzione della casualità permette all'algoritmo di esplorare lo spazio delle soluzioni in modo più ampio, evitando di rimanere intrappolato in minimi locali precoci.

Aggregazione della rete: Infine, nel passaggio (d), il grafo viene aggregato. La differenza sostanziale rispetto a Louvain è che i super-nodi del livello successivo non corrispondono alla partizione grezza (b), ma alle comunità raffinate (c). Questo garantisce che il grafo di livello superiore preservi fedelmente la struttura topologica e che non vengano trascinati errori di disconnessione nelle iterazioni successive.

L'esito finale di questa procedura è una gerarchia di partizioni in cui ogni comunità, a qualsiasi livello di scala, è matematicamente garantita essere ben connessa, fornendo una base solida e coerente per la mappatura dei domini funzionali del database.

3.4 Implementazione dell'algoritmo Leiden

Per l'implementazione concreta dell'algoritmo di Leiden all'interno della pipeline di Dataslice AI è stata realizzata in ambiente Python, con l'obiettivo di integrare in modo fluido la fase di community detection nel flusso di elaborazione del grafo strutturale del database. Il codice è stato progettato per operare su grafi costruiti a partire dagli script DDL, mantenendo una chiara separazione tra informazione topologica e arricchimento semantico.

Qui di seguito vengono descritti i dettagli implementativi dell'algoritmo, le librerie adottate e il flusso logico che consente di passare dal grafo strutturale grezzo alla definizione delle comunità.

3.4.1 Librerie utilizzate

L'implementazione fa uso di un insieme ristretto ma mirato di librerie, ciascuna con un ruolo specifico:

- **NetworkX**. Utilizzata come struttura dati primaria per la rappresentazione del grafo del database all'interno della pipeline. NetworkX consente una modellazione flessibile dei nodi (tabelle) e degli archi (Foreign Key), risultando particolarmente adatta alla fase di costruzione e serializzazione del grafo.

- **python-igraph**. Libreria ad alte prestazioni impiegata per l'esecuzione computazionale dell'algoritmo. Essa fornisce un'interfaccia Python verso la libreria `igraph`, implementata in C, permettendo di gestire grafi di dimensioni rilevanti con tempi di esecuzione contenuti.

- **leidenalg**. Pacchetto ufficiale che implementa l'algoritmo di Leiden sopra il motore di `igraph`. La libreria espone la funzione `find_partition`, che incapsula le fasi di ottimizzazione locale, raffinamento e aggregazione del grafo.

- **pickle**. Utilizzato per la serializzazione e il caricamento persistente del grafo, consentendo di separare la fase di costruzione del grafo da quella di analisi e clustering.

3.4.2 Flusso di esecuzione dell'algoritmo

L'implementazione dell'algoritmo è stata incapsulata nella funzione `findLeidenCommunities`. Quest'ultima segue un flusso di esecuzione sequenziale strutturato in tre fasi logiche principali: *Caricamento*, *Conversione* ed *Esecuzione*.

Caricamento del grafo strutturale

Il punto di partenza dell'elaborazione è il caricamento del grafo serializzato:

```
1 with open('..._graph_emb.pkl', 'rb') as f:
2     Graph = pickle.load(f)
```

È importante sottolineare che il grafo caricato in questa fase contiene esclusivamente l'informazione topologica derivante dalla fase di parsing degli script SQL tramite LLM. In particolare, i nodi rappresentano unicamente le tabelle individuate nei file DDL, mentre gli archi rappresentano esclusivamente i vincoli di integrità referenziale (Foreign Key) estratti durante la fase di analisi.
Infatti in questa fase dell'analisi, eventuali attributi semantici o embedding associati ai nodi non vengono utilizzati. Questo poichè l'algoritmo di Leiden opera su una rappresentazione puramente strutturale del database.

Conversione del grafo e preparazione dei dati

Poiché l'implementazione di Leiden richiede un grafo in formato `igraph`, il grafo NetworkX viene preliminarmente convertito. La conversione avviene trasformando il grafo in una versione non orientata, coerentemente con l'ipotesi che una Foreign Key rappresenti una dipendenza strutturale indipendente dalla direzione del vincolo. Durante questa fase, i nomi delle tabelle vengono preservati come attributo `name` dei vertici `igraph`, consentendo di mantenere una corrispondenza diretta tra i nodi del grafo e le entità del database originale.

Esecuzione dell'algoritmo di Leiden

Infine, l'esecuzione dell'algoritmo avviene tramite la funzione:

```
1 leidenalg.find_partition(g_ig, ModularityVertexPartition)
```

La configurazione adottata utilizza la classe `ModularityVertexPartition`, che imposta la modularità standard come funzione obiettivo da massimizzare. Tale scelta risulta coerente con l'obiettivo di individuare comunità caratterizzate da un'elevata densità di connessioni interne e da un numero ridotto di collegamenti esterni.

L'algoritmo restituisce una partizione del grafo, rappresentata come un insieme di comunità, ciascuna definita da un sottoinsieme di nodi. Questo output viene successivamente trasformato in un dizionario di mapping nella forma:

$$\{\texttt{Nome\_Tabella} \rightarrow \texttt{Community\_ID}\}$$

che associa a ogni tabella l'identificativo numerico della comunità di appartenenza. Tale mapping nodo–comunità rappresenta il primo livello di classificazione del database ottenuto mediante un approccio puramente topologico. Esso costituisce una baseline strutturale, che verrà successivamente confrontata con approcci basati su informazioni semantiche ed embedding testuali, al fine di valutare il contributo dell'arricchimento semantico rispetto alla sola architettura schematica del database.

Capitolo 4

Graph Neural Networks per il clustering

L'analisi condotta nel capitolo precedente ha mostrato come i metodi puramente topologici, e in particolare l'algoritmo di Leiden, costituiscano uno strumento efficace per la scomposizione di uno schema di database basata sui vincoli di integrità referenziale. Tuttavia, un approccio esclusivamente strutturale presenta una limitazione intrinseca: i nodi del grafo vengono trattati come entità anonime, distinguibili unicamente in funzione della loro posizione nella rete, senza tenere conto del contenuto informativo che essi rappresentano.

Nel contesto applicativo reale, le tabelle di un database non sono semplici vertici astratti, ma entità portatrici di una ricca semantica, espressa attraverso i nomi, le descrizioni e la tipologia delle colonne. È quindi possibile che due tabelle non siano direttamente collegate da un vincolo di chiave esterna — e risultino di conseguenza distanti secondo un criterio puramente topologico — pur appartenendo allo stesso dominio funzionale, in quanto modellano concetti affini o complementari. Per catturare questa duplice natura del dato relazionale — strutturale e semantica — si rende necessario estendere l'analisi oltre i metodi di ottimizzazione discreta. In questo scenario si inserisce il paradigma del *Graph Representation Learning*, che consente di integrare la topologia del grafo con attributi descrittivi dei nodi, aprendo la strada all'impiego delle *Graph Neural Networks* come strumento per l'apprendimento di rappresentazioni latenti semanticamente e strutturalmente informative.

4.1 Principi generali delle GNN

Negli ultimi anni, il Deep Learning ha rivoluzionato l'analisi dei dati in molteplici domini, dall'elaborazione delle immagini (Computer Vision) al trattamento del linguaggio naturale (NLP). Tuttavia, le architetture neurali tradizionali, come le Convolutional Neural Networks (CNN) o le Recurrent Neural Networks (RNN), sono progettate per operare su dati strutturati in forme euclidee regolari, come griglie di pixel o sequenze temporali lineari. I dati relazionali, come quelli che compongono lo schema di un database, non si adattano a queste strutture rigide: essi vivono in uno spazio non euclideo, caratterizzato da relazioni complesse e irregolari, dove ogni nodo può avere un numero arbitrario di vicini e non esiste un ordinamento spaziale predefinito.

Per estendere le capacità dell'apprendimento automatico a strutture relazionali complesse, la ricerca ha progressivamente sviluppato modelli neurali in grado di operare direttamente su grafi. In questo contesto si collocano le Graph Neural Networks (GNN), introdotte formalmente da Scarselli et al. [13] e successivamente evolute in numerose varianti architetturali. A differenza degli approcci di embedding statico, che producono rappresentazioni pre-calcolate e indipendenti dal task, le GNN sono modelli parametrici addestrabili end-to-end. Esse apprendono rappresentazioni dei nodi attraverso un processo iterativo di aggregazione locale dell'informazione, nel quale la struttura del grafo non è un semplice supporto, ma parte integrante del meccanismo computazionale. Questa caratteristica risulta particolarmente rilevante nel contesto dei database relazionali, in cui ogni tabella è definita simultaneamente dalla propria semantica interna e dalla rete di dipendenze strutturali in cui è inserita.

Tale capacità di modellazione congiunta costituisce la peculiarità fondamentale che rende le GNN insostituibili per gli obiettivi di questa tesi: la fusione informativa. Mentre un algoritmo puramente topologico (come Leiden, analizzato nel capitolo precedente) osserva esclusivamente la matrice di adiacenza A, e un modello di NLP classico elaborerebbe le sole descrizioni testuali disgiunte dalla rete, una GNN accetta in input l'intera tripla che definisce il Grafo con Attributi:

$$G = (V, E, X)$$

dove ricapitolando, X rappresenta la Matrice delle Feature (o degli attributi), la cui riga i-esima, indicata con $\mathbf{x}_i$, corrisponde al vettore delle feature associato al nodo v_i. Nel contesto specifico di questo lavoro, la costruzione della matrice X è un passaggio fondamentale che trasforma i metadati testuali in input numerici processabili dalla rete neurale. Ogni vettore $\mathbf{x}_i$ non contiene valori arbitrari, ma è il risultato di un processo di *embedding* semantico.

Di conseguenza, l'input ai modelli GNN implementati in questa tesi non sarà costituito unicamente dalle informazioni contenute nella matrice di adiacenza A (ottenuta dai collegamenti tra le foreign key fornite da E) e dall'insieme dei nodi V, ma includerà anche il contenuto semantico (X). Lo scopo della rete è quindi quello di apprendere una funzione di encoding parametrica che associ a ciascun nodo v_i un vettore latente $h_i \in \mathbb{R}^{d'}$, tale per cui h_i riassuma sinteticamente tre livelli informativi:

- Le caratteristiche semantiche native del nodo stesso (il vettore $\mathbf{x}_i$ generato dall'LLM);
- Le caratteristiche strutturali del suo vicinato (es. a quali altre tabelle è collegato tramite Foreign Key);
- Le caratteristiche semantiche dei nodi vicini (informazione diffusa attraverso gli archi).

L'evoluzione moderna di queste architetture, esplosa con l'introduzione delle Graph Convolutional Networks (GCN) da parte di Kipf e Welling [14], si basa su un paradigma operativo comune che ha rimpiazzato le complesse definizioni basate sulla teoria spettrale dei grafi: il framework del Neural Message Passing. Questo approccio unifica la maggior parte delle moderne architetture GNN sotto un unico schema algoritmico: la rappresentazione di un nodo viene aggiornata iterativamente aggregando i messaggi provenienti dai suoi vicini diretti. Nelle sezioni successive verranno dettagliati i meccanismi matematici di questo paradigma e come esso costituisca il blocco costruttivo fondamentale per i modelli di clustering avanzati utilizzati in questo progetto, ovvero DMON e DGCluster.

4.1.1 Il paradigma del Neural Message Passing

Il concetto centrale che governa l'apprendimento nelle moderne Graph Neural Networks è il Neural Message Passing, formalizzato da Gilmer et al. [15]. In questo framework, il calcolo dell'embedding di un nodo non avviene in isolamento, ma è il risultato di un processo iterativo di scambio di informazioni tra nodi adiacenti. L'intuizione alla base è che per comprendere il ruolo funzionale di una tabella, è necessario analizzare non solo le sue proprietà intrinseche, ma anche quelle delle tabelle a essa collegate. Ad esempio, una tabella anonima collegata a "Ordini" e "Fatture" assumerà probabilmente un ruolo legato al dominio "Vendite". Il message passing simula questo processo cognitivo: ogni nodo "invia un messaggio" contenente le proprie informazioni ai vicini, e successivamente aggiorna il proprio stato interno aggregando i messaggi ricevuti. Formalmente, indichiamo con $h_v^{(k)}$ la rappresentazione latente (feature vector) del nodo v al layer k-esimo della rete

neurale. Al livello iniziale ($k = 0$), questa rappresentazione coincide con le feature originali del nodo:

$$h_v^{(0)} = \mathbf{x}_v$$

La propagazione dell'informazione dal layer k al layer $k + 1$ avviene attraverso due fasi distinte e sequenziali: la fase di Aggregazione e la fase di Aggiornamento (Update). Per ogni nodo $v \in V$, l'operazione può essere descritta dalla seguente equazione generale:

$$h_v^{(k+1)} = \text{UPDATE}^{(k)}\left(h_v^{(k)}, \text{AGGREGATE}^{(k)}\left(h_u^{(k)} : u \in \mathcal{N}(v)\right)\right) \tag{4.1}$$

dove:

- $\mathcal{N}(v)$ rappresenta l'insieme dei nodi vicini a v (il vicinato o neighborhood);
- $h_u^{(k)}$ sono le rappresentazioni dei nodi vicini al passo precedente;
- $\text{AGGREGATE}^{(k)}(\cdot)$ è una funzione differenziabile e invarinte alle permutazioni (come la somma, la media o il massimo) che comprime l'insieme dei messaggi in arrivo dai vicini in un unico vettore contesto;
- $\text{UPDATE}^{(k)}(\cdot)$ è una funzione non lineare (spesso una rete neurale, come una MLP o una GRU) che combina la vecchia rappresentazione del nodo $h_v^{(k)}$ con il vettore aggregato dai vicini per produrre il nuovo embedding.

Il campo recettivo e la profondità della rete

Un aspetto fondamentale di questo paradigma è il concetto di campo recettivo (*receptive field*). Dopo K iterazioni (o layer) di message passing, la rappresentazione finale di un nodo v conterrà informazioni aggregate provenienti da tutti i nodi che distano al massimo K salti (hop) da esso nel grafo. Nel contesto di questo progetto, questo meccanismo permette a ogni tabella di acquisire consapevolezza del contesto strutturale in cui è immersa. Difatti, prendendo come esempio la figura 4.1 notiamo che:

- Al layer 1, la tabella apprende dai suoi vicini diretti (es. tabelle collegate da una FK).
- Al layer 2, l'informazione si propaga dai "vicini dei vicini", permettendo di catturare dipendenze funzionali indirette o di ordine superiore.

L'output finale dell'ultimo layer K, indicato con $Z = H^{(K)}$, costituisce la matrice degli embedding nodali che verrà utilizzata come input per i task successivi, come appunto il clustering non supervisionato.

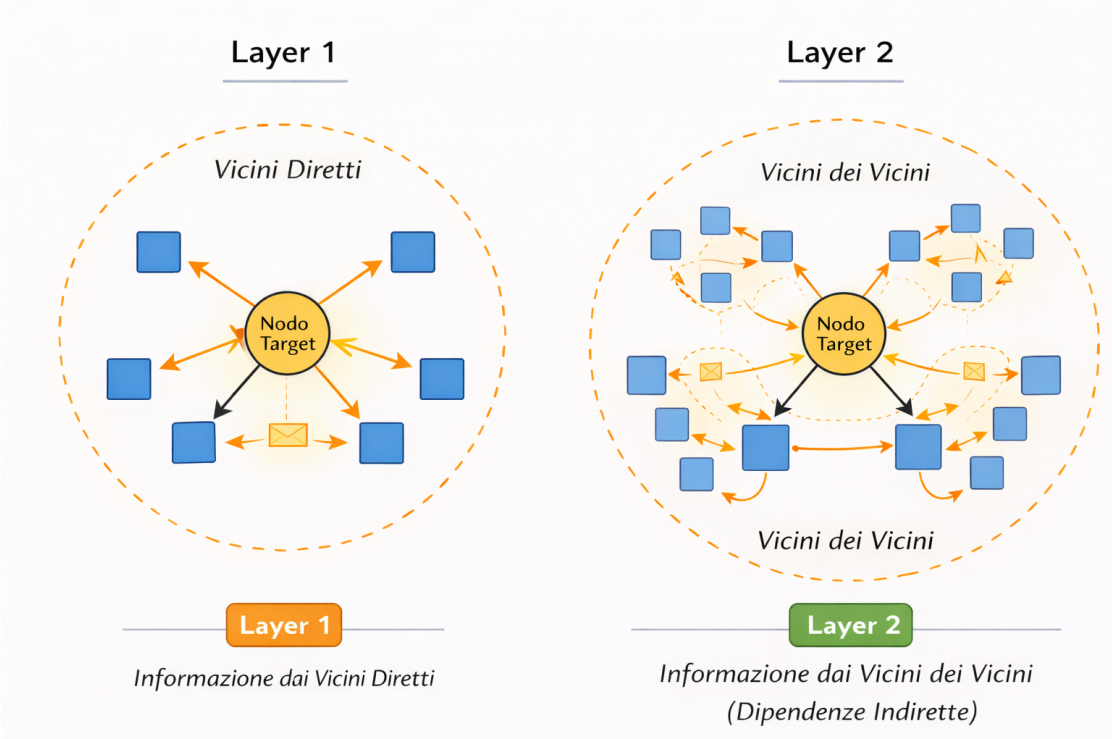

Figura 4.1: Rappresentazione del campo recettivo in una Graph Neural Network. Nel Layer 1 (sinistra), il nodo target aggrega informazione esclusivamente dai vicini diretti (nodi evidenziati in blu), mentre i nodi più distanti rimangono non osservabili. Nel Layer 2 (destra), il campo recettivo si espande ai vicini dei vicini, includendo dipendenze di secondo ordine, mentre i nodi oltre due hop restano esclusi dal processo di aggiornamento.

4.1.2 Graph Convolutional Networks (GCN)

Tra le numerose architetture basate sul paradigma del Message Passing, le Graph Convolutional Networks (GCN), introdotte da Kipf e Welling nel 2017 [14], rappresentano senza dubbio il modello più influente e ampiamente adottato. La loro importanza nel campo del Geometric Deep Learning è paragonabile a quella delle CNN per la Computer Vision: hanno fornito per la prima volta un metodo efficiente, scalabile e matematicamente elegante per estendere l'operazione di convoluzione ai grafi. Nel contesto di questa tesi, la GCN non costituisce l'architettura di clustering in sé, ma funge da Encoder (codificatore). Il suo ruolo è quello di trasformare i dati grezzi dello schema database (struttura delle Foreign Key + feature testuali delle tabelle) in una rappresentazione vettoriale latente ricca di significato, sulla quale verranno poi applicati gli algoritmi di partizionamento (come vedremo con DMON e DGCluster).

Dalle convoluzioni spettrali all'approssimazione spaziale

Per comprendere la genesi della formula della GCN, è necessario fare un breve passo indietro verso la teoria spettrale dei grafi. Inizialmente, il concetto di convoluzione su grafo era stato definito nel dominio di Fourier. Data una trasformata di Fourier su grafo definita tramite gli autovettori del Laplaciano del grafo ($L = D - A$), una convoluzione veniva calcolata proiettando il segnale di input nello spazio spettrale, moltiplicandolo per un filtro parametrico e ritrasformandolo nel dominio spaziale. Sebbene teoricamente rigoroso, questo approccio presentava due limitazioni fatali per l'applicazione su database reali:

1. Inefficienza computazionale: La decomposizione in autovettori del Laplaciano è un'operazione con complessità $O(N^3)$. Per uno schema database con migliaia di tabelle, questo calcolo è proibitivo.

2. Non localizzazione: I filtri spettrali non sono naturalmente localizzati nello spazio, il che significa che una modifica in un nodo potrebbe influenzare istantaneamente nodi molto distanti, violando il principio di località delle dipendenze funzionali.

Il contributo rivoluzionario di Kipf e Welling è stato dimostrare che è possibile bypassare la costosa decomposizione spettrale utilizzando un'approssimazione al primo ordine dei polinomi di Chebyshev. Questa semplificazione ha permesso di tradurre la convoluzione spettrale in un'operazione spaziale estremamente efficiente, basata solo sul vicinato immediato dei nodi.

La regola di propagazione layer-wise

Il risultato di questa derivazione è una regola di propagazione semplice e potente, che costituisce il cuore della GCN. Dato un grafo con matrice di adiacenza A e una matrice delle feature $H^{(l)}$ al livello l, l'aggiornamento per ottenere il livello successivo $H^{(l+1)}$ è definito come:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{4.2}$$

Analizziamo nel dettaglio ogni componente di questa equazione fondamentale per comprenderne il significato fisico nel contesto delle tabelle di un database:

1. La Matrice di Adiacenza con Self-Loops ($\tilde{A}$)
La formula utilizza $\tilde{A} = A + I_N$, dove I_N è la matrice identità. L'aggiunta della matrice identità (ovvero l'aggiunta di un arco che collega ogni nodo a se stesso, detto self-loop) è cruciale. Senza di essa, durante la moltiplicazione $AH^{(l)}$, il vettore del nodo v verrebbe calcolato sommando solo le feature dei suoi vicini, perdendo completamente la propria informazione originale. In un database, questo

significherebbe che la rappresentazione di una tabella "Ordini" dipenderebbe solo dalle tabelle a cui è collegata (es. "Clienti", "Prodotti"), dimenticando il proprio contenuto (le proprie colonne). L'uso di $\tilde{A}$ garantisce che l'informazione strutturale dei vicini venga integrata con l'informazione semantica del nodo stesso.

2. La Normalizzazione Simmetrica ($\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$)
Se ci limitassimo a sommare le feature dei vicini (moltiplicando solo per $\tilde{A}$), i nodi con un grado elevato (tabelle hub molto connesse) accumulerebbero valori numerici enormemente più grandi rispetto ai nodi periferici, causando instabilità numerica e l'esplosione dei gradienti. Per evitare ciò, la GCN applica una normalizzazione simmetrica basata sulla matrice dei gradi $\tilde{D}$ (dove $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$).Matematicamente, l'elemento (i, j) della matrice normalizzata diventa:

$$\frac{1}{\sqrt{\deg(i)\deg(j)}}$$

Questo fattore di scala ha un importante effetto di bilanciamento: il messaggio proveniente da un vicino j verso i viene pesato meno se j ha molti altri vicini (il suo segnale è "diluito") e se i riceve già messaggi da molti nodi.

3. La Trasformazione Lineare ($W^{(l)}$)
$W^{(l)}$ è la matrice dei pesi addestrabili del layer l, ed è qui che risiede la conoscenza della rete. Durante il training, la GCN impara come combinare le feature estratte per massimizzare la funzione obiettivo. Questa matrice fa si che proietti i vettori in uno spazio latente diverso, permettendo di estrarre feature di livello superiore.

4. La Funzione di Attivazione (σ)
Infine, viene applicata una funzione non lineare, tipicamente la ReLU
($RectifiedLinearUnit$, $\sigma(x) = \max(0, x)$). Senza questa non-linearità, impilare più strati di GCN equivarrebbe a una singola moltiplicazione matriciale lineare, annullando la capacità del modello di apprendere pattern complessi.

Interpretazione intuitiva: Smoothing e Omofilia

Al di là della formulazione algebrica, è utile visualizzare cosa accade ai dati delle tabelle dopo il passaggio attraverso una GCN. L'operazione descritta dall'equazione (4.2) agisce essenzialmente come una media ponderata delle feature nel vicinato, seguita da una trasformazione. Dal punto di vista dell'elaborazione dei segnali, la GCN si comporta come un filtro passa-basso (*low-pass filter*). In un grafo, un segnale ad "alta frequenza" è un segnale che cambia bruscamente tra nodi vicini; un segnale a "bassa frequenza" varia gradualmente. Applicando la convoluzione, la GCN tende a rendere più "liscie" (*smooths*) le rappresentazioni dei nodi: le feature di un nodo tendono a diventare più simili a quelle dei suoi vicini. Questo meccanismo è il fondamento teorico che giustifica l'uso delle GCN per il clustering

basato sull'ipotesi di omofilia (*homophily*): nodi collegati tendono a essere simili. Nel contesto del nostro database:

- Immaginiamo due tabelle, T_A e T_B, collegate da una Foreign Key.
- Inizialmente, i loro vettori semantici X_A e X_B (generati dall'LLM) potrebbero essere distanti nello spazio vettoriale se usano terminologie leggermente diverse.
- Dopo un layer di GCN, la nuova rappresentazione di T_A conterrà una parte di T_B e viceversa. I loro vettori si avvicineranno.
- Dopo più layer, le tabelle che formano una comunità densamente connessa condivideranno una rappresentazione latente molto simile (un baricentro comune), rendendo facile per un algoritmo di clustering separarli dalle altre comunità.

Limiti e profondità: Il problema dell'Oversmoothing

Tuttavia, questa potente capacità di smoothing porta con sé un rischio noto come Oversmoothing. Se la GCN è troppo profonda (troppi layer), l'aggregazione ripetuta porta le rappresentazioni di tutti i nodi del grafo a convergere verso un unico vettore medio indistinguibile. In termini pratici, se si esagera con la propagazione, ogni tabella finisce per conoscere l'intero database e perde la sua specificità locale. Per questo motivo, nell'implementazione dei modelli DMON e DGCluster descritti nelle sezioni successive, l'encoder GCN è stato configurato con una profondità ridotta (tipicamente 1 o 2/3 hidden layers). Questo permette di catturare il contesto locale (le relazioni dirette e di secondo o terzo livello) necessarie per identificare i domini funzionali, senza diluire l'informazione semantica specifica che distingue, ad esempio, il dominio "Risorse Umane" dal dominio "Contabilità", anche se questi sono indirettamente collegati nel grafo aziendale globale.

4.2 Il modello DMON

La prima architettura di Deep Graph Clustering implementata e analizzata in questo lavoro è il modello DMON (Deep Modularity Networks), proposto da Tsitsulin et al. nel 2020 [16]. La scelta di questo modello non è casuale, ma rappresenta la naturale evoluzione in chiave neurale del percorso teorico iniziato nel capitolo precedente. Se l'algoritmo di Leiden affrontava il problema della Community Detection cercando di massimizzare la modularità Q attraverso euristiche combinatorie (spostamento di nodi e aggregazione), DMON si pone il medesimo obiettivo — ottenere partizioni ad alta modularità — ma lo persegue attraverso l'ottimizzazione di una funzione di costo differenziabile.

Il modello opera secondo un paradigma non supervisionato: non richiede etichette di verità (ground truth) per l'addestramento, ma apprende a raggruppare le tabelle sfruttando due segnali guida:

- Il segnale strutturale: La rete cerca di identificare gruppi di nodi densamente connessi, penalizzando i tagli che rompono cluster forti.
- Il segnale semantico: Grazie all'utilizzo di un encoder GNN, il clustering non è cieco al contenuto, ma tende a raggruppare nodi che hanno feature simili (descrizioni e colonne affini), anche in presenza di rumore topologico.

A differenza dei metodi come Leiden, che assegnano ogni nodo a una e una sola comunità in modo binario, DMON produce un Soft Clustering: per ogni nodo, il modello restituisce un vettore di probabilità che indica il grado di appartenenza a ciascuno dei cluster identificati. Questa natura probabilistica è ciò che rende l'intero processo differenziabile, permettendo al gradiente di retro-propagarsi per aggiornare i pesi dell'encoder.

La Figura 4.2 fornisce una rappresentazione visiva dell'intero processo di elaborazione. A partire da questa visione d'insieme, nelle sezioni successive verrà dissezionata l'architettura del modello, analizzando nel dettaglio come l'informazione fluisce dall'Encoder fino al modulo di Assignment, e come le diverse varianti implementative influenzino la capacità del sistema di rilevare domini funzionali coerenti.

4.2.1 Architettura del modello DMON

Il modello Deep Modularity Networks (DMON) rappresenta un approccio innovativo al clustering su grafi, progettato per superare i limiti di scalabilità dei metodi spettrali tradizionali e per sfruttare appieno la potenza espressiva delle Graph Neural Networks. A differenza dell'algoritmo di Leiden, che opera in uno spazio discreto esplorando euristicamente le combinazioni di nodi, DMON formula la Community Detection come un problema di ottimizzazione continua. Il principio architetturale di base segue il paradigma codificatore-decodificatore (Encoder-Decoder), sebbene con una declinazione specifica per il clustering:

1. Un Encoder neurale estrae le feature strutturali e semantiche dal grafo $G = (X, A)$ e le proietta in uno spazio vettoriale latente Z.
2. Un Modulo di Assegnamento mappa queste rappresentazioni latenti in una matrice di assegnamento probabilistico (Soft Clustering).
3. Una Funzione di Loss non supervisionata guida l'addestramento della rete calcolando la modularità differenziabile e applicando termini di regolarizzazione direttamente sulla matrice di assegnamento, permettendo l'aggiornamento dei pesi tramite retro-propagazione (backpropagation).

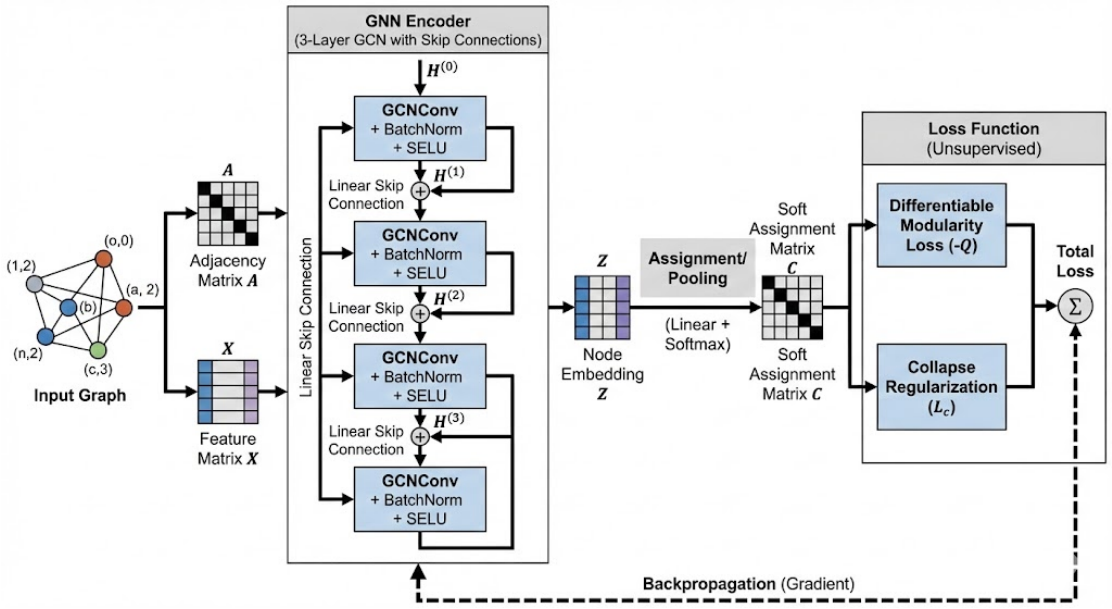

Figura 4.2: Architettura del modello DMON. Il flusso di elaborazione parte dal grafo di input definito dalla matrice di adiacenza (A) e dalla matrice delle feature semantiche (X). L'informazione viene processata da un *GNN Encoder* composto da tre layer convoluzionali (GCNConv) dotati di *skip connections* lineari, stabilizzati tramite BatchNorm e attivazione SELU. L'embedding latente risultante (Z) viene mappato in una matrice di assegnamento probabilistico (C) tramite un layer di pooling lineare seguito da Softmax. Infine, la fase di addestramento non supervisionato è guidata da una funzione di loss composita che minimizza la modularità differenziabile ($-Q$) e applica una penalità di collasso (L_c), permettendo l'ottimizzazione *end-to-end* dei pesi tramite retro-propagazione.

Nell'implementazione realizzata per questo lavoro, l'architettura originale di DMON è stata adattata e arricchita per rispondere alle specificità dei grafi estratti dai database relazionali. Di seguito verranno dissezionati i singoli componenti, confrontando le scelte teoriche del paper originale con l'effettiva traduzione in codice (sviluppata utilizzando il framework `PyTorch Geometric`).

4.2.2 L'Encoder (GNN con Skip Connections)

Nel paper originale di Tsitsulin et al. [16], l'encoder viene trattato in modo agnostico, suggerendo l'utilizzo di una semplice Multi-Layer Perceptron (MLP) o di una GCN standard. Tuttavia, l'analisi degli schemi database ha reso necessaria la progettazione di un codificatore nettamente più sofisticato per evitare il rischio di oversmoothing e per bilanciare l'importanza tra la topologia e i metadati semantici.

L'encoder implementato (classe `GNNEncoder`) si basa su una struttura a 3 livelli di convoluzione su grafo (GCNConv), profondità scelta per permettere a ogni tabella di apprendere il contesto funzionale fino a tre "salti" (hop) di distanza lungo i vincoli di Foreign Key. La grande innovazione rispetto a una GCN classica risiede nell'integrazione di connessioni residue lineari (Skip Connections) e tecniche di stabilizzazione avanzate. Formalmente, l'operazione compiuta in ciascun layer l (per $l \in \{1, 2, 3\}$) non è una semplice aggregazione di vicinato, ma è definita dalla seguente equazione:

$$H^{(l)} = \text{Dropout}\left(\text{SELU}\left(\text{BatchNorm}\Big(\text{GCN}(H^{(l-1)}, A)\Big) + W_{skip}^{(l)} H^{(l-1)}\right)\right) \quad (4.3)$$

Dove:

- GCN$(\cdot)$ calcola il message passing standard aggregando le informazioni dei nodi adiacenti.
- BatchNorm normalizza le feature lungo la dimensione del batch, stabilizzando la varianza interna e accelerando la convergenza, un aspetto critico quando si ottimizzano loss non supervisionate come la modularità.
- $W_{skip}^{(l)} H^{(l-1)}$ rappresenta la Skip Connection Lineare. Invece di sommare direttamente l'input (come nelle ResNet standard), l'input viene prima trasformato da un layer lineare (`nn.Linear`). Questa scelta è fondamentale: garantisce che il nodo non "dimentichi" le sue feature semantiche originali a causa dell'aggregazione massiva dei layer convoluzionali, preservando l'identità della tabella.
- SELU (Scaled Exponential Linear Unit) è utilizzata come funzione di attivazione al posto della tradizionale ReLU, poiché induce proprietà di autonormalizzazione nei gradienti.

L'output finale dell'encoder è la matrice di embedding $Z \in \mathbb{R}^{N \times d}$, dove d è la dimensione latente (`embed_dim`), che cattura in modo denso e robusto l'intersezione tra struttura relazionale e semantica LLM.

4.2.3 Assegnamento ai Cluster (Multi-head Pooling vs Implementazione)

Una volta ottenuta la rappresentazione latente Z, il modello deve decidere a quale comunità (dominio funzionale) appartiene ciascuna tabella. Nel paper teorico di DMON, gli autori affrontano il problema dell'instabilità del clustering introducendo un meccanismo di Multi-head Pooling. Poiché l'ottimizzazione della modularità

presenta numerosi minimi locali, l'architettura originale suggerisce di generare assegnamenti multipli in parallelo (le "teste" o heads), calcolando la modularità per ciascuna di esse e selezionando a posteriori quella che produce il valore migliore. Ogni testa nel paper è un modulo MLP che proietta Z nello spazio dei cluster. Nell'implementazione analizzata in questa sezione (classe `DMoNCluster`), si è optato per una variante semplificata e diretta. Grazie alla notevole robustezza fornita dall'encoder customizzato con skip-connections (che produce embedding già intrinsecamente ben separati), la complessità del multi-head pooling è stata ridotta a un singolo layer di proiezione lineare ad alta efficienza. Matematicamente, la matrice di assegnamento $C \in \mathbb{R}^{N \times K}$ (dove N è il numero di tabelle e K il numero di cluster desiderati) viene calcolata come:

$$C = \text{Softmax}\Big(ZW_{assign} + b_{assign}\Big) \tag{4.4}$$

L'applicazione della funzione *Softmax* lungo l'ultima dimensione garantisce che per ogni nodo i, il vettore riga C_i rappresenti una distribuzione di probabilità valida ($\sum_{j=1}^{K} C_{ij} = 1$). Il valore $C_{ij} \in [0, 1]$ indica quindi la probabilità (o il grado di certezza) che la tabella i appartenga al modulo funzionale j. Questa natura probabilistica (Soft Clustering) è il fattore abilitante dell'intero algoritmo: essendo un'operazione differenziabile, permette al gradiente della loss di scorrere all'indietro fino all'encoder. Al termine dell'addestramento, l'assegnamento finale viene ottenuto semplicemente estraendo l'indice del valore massimo (`torch.argmax(C, dim=1)`).

4.2.4 La Funzione di Loss (Modularità e Collapse Penalty)

Il cuore concettuale di DMON risiede nella sua funzione di loss (`dmon_loss`), la quale non avendo a disposizione etichette vere, si basa unicamente su metriche di qualità topologica. La loss totale è composta da due termini concorrenti: la modularità differenziabile e una regolarizzazione per prevenire il collasso dei cluster.

1. Modularità Spettrale Differenziabile

Nel capitolo 3, la modularità Q è stata presentata nella sua forma combinatoria. Per inserirla in una rete neurale, DMON la riformula sfruttando la traccia delle matrici. Definita la matrice di modularità $B = A - \frac{\mathbf{d}\mathbf{d}^T}{2m}$ (dove A è la matrice di adiacenza, $\mathbf{d}$ è il vettore dei gradi e m il numero totale di archi), la modularità per un assegnamento *soft* C diventa:

$$Q = \frac{1}{2m}\text{Tr}(C^T B C) \tag{4.5}$$

Per ragioni di efficienza in memoria (evitando di istanziare l'intera matrice densa B, che richiederebbe spazio $O(N^2)$), l'implementazione calcola questo valore scomponendolo in due addendi vettoriali:

1. **Termine di connessione effettiva** (`dot_edges`): Rappresenta $C^T AC$. Nel codice viene calcolato sfruttando la lista di adiacenza `edge_index`, moltiplicando e sommando le probabilità di cluster condivise tra i nodi connessi (`(cu * cv).sum()`).

2. **Termine di connessione attesa**: Rappresenta $C^T \frac{\mathbf{d}\mathbf{d}^T}{2m} C$. Nel codice viene efficientemente computato calcolando prima il grado atteso per ogni cluster $dC = \sum_i d_i C_{ij}$ e successivamente la somma dei loro quadrati normalizzata.

Poiché PyTorch minimizza le funzioni obiettivo tramite discesa del gradiente, il valore calcolato di Q viene negato (`loss_mod = -modularity`), trasformando il problema della massimizzazione della modularità nella minimizzazione della loss.

2. Collapse Regularization (Penalità di collasso)

Ottimizzare esclusivamente la modularità tramite reti neurali porta spesso al problema dei minimi locali banali: la rete potrebbe assegnare tutte le tabelle a una singola macro-comunità o svuotare interi cluster, non apprendendo nulla di utile sulla struttura funzionale del database. Per forzare la rete a distribuire i nodi in modo più equo e per utilizzare l'intero spazio di clustering K, DMON introduce un termine di penalità chiamato *Collapse Loss* (L_c). Indicando con $S_j = \sum_{i=1}^{N} C_{ij}$ la "dimensione *soft*" del cluster j, la regolarizzazione è definita come:

$$L_c = \frac{\sqrt{K}}{N} \sqrt{\sum_{j=1}^{K} S_j^2} - 1 = \frac{\sqrt{K}}{N} |C^T \mathbf{1}|_2 - 1 \tag{4.6}$$

Questo termine (perfettamente rispecchiato nel codice dall'espressione `(norm(cluster_sizes) / N) * sqrt(K) - 1`) raggiunge il suo minimo globale pari a 0 solo quando i nodi sono perfettamente equidistribuiti tra tutti i K cluster ($S_j = \frac{N}{K}\ \forall j$). Qualsiasi deviazione verso cluster troppo grandi viene penalizzata.

La funzione di loss finale risulta essere una combinazione convessa:

$$\mathcal{L} = -Q + \lambda L_c \tag{4.7}$$

dove λ (`collapse_lambda`, impostato di default a 1.0 nel framework) agisce come iperparametro per calibrare il compromesso tra la coesione strutturale (modularità) e l'equità della distribuzione dei domini applicativi (collapse). Il monitoraggio in fase di training di queste due metriche ha permesso di estrarre e congelare le

configurazioni ottimali, i cui risultati quantitativi saranno oggetto di discussione nel capitolo conclusivo.

4.2.5 Estensione dell'architettura DMON: Gestione degli embedding multipli

La formulazione standard delle Graph Neural Networks, e di conseguenza la prima architettura DMON descritta in precedenza, assume che ogni nodo del grafo sia descritto da un singolo vettore di feature. Matematicamente, l'input spaziale è rappresentato da una matrice $X \in \mathbb{R}^{N \times D}$, dove N è il numero di nodi e D la dimensione dell'embedding semantico. Tuttavia, l'applicazione di questa architettura al contesto dei database relazionali ha fatto emergere una sfida rappresentativa. Le tabelle aziendali sono spesso corredate da descrizioni estese (generate o estratte in fase di parsing), metadati complessi e un numero elevato di colonne. Comprimere tutto questo testo in un unico embedding vettoriale rischia di causare una grave perdita di granularità, fenomeno noto in letteratura come information bottleneck. Per ovviare a questo problema, in questo lavoro è stata implementata una strategia avanzata di chunking (che verrà dettagliata compiutamente nel capitolo relativo agli esperimenti). L'idea alla base consiste nello spezzare il testo descrittivo di ogni tabella in K segmenti separati (i chunk), calcolando un embedding distinto per ciascuno di essi. Di conseguenza, il dato di input semantico cambia dimensionalità: non è più una matrice 2D, ma un tensore tridimensionale $X_{chunks} \in \mathbb{R}^{N \times K \times D}$, dove ogni nodo possiede K vettori di dimensione D. Se un nodo possiede meno di K chunk, viene applicata una maschera di padding (identificata nel codice dalla variabile `mask`). Poiché il GNN Encoder non è strutturato per elaborare tensori 3D, si è reso necessario progettare una seconda variante dell'architettura DMON (classe `DMoNClusterWithPooling`), introducendo un modulo di aggregazione neurale a monte della rete convoluzionale.

Il meccanismo di Learned Pooling (Attention)

L'approccio più banale per ricondurre K embedding a un singolo vettore consisterebbe nell'applicare una media aritmetica (Average Pooling) o un massimo (Max Pooling). Tuttavia, queste operazioni non parametriche assumono che tutti i frammenti di testo abbiano la stessa rilevanza funzionale, un'ipotesi irrealistica: alcuni chunk potrebbero contenere parole chiave cruciali per identificare il dominio di appartenenza della tabella (es. "Gestione fatture"), mentre altri potrebbero contenere informazioni di contorno o rumore. Per questo motivo, è stato implementato un meccanismo di Learned Pooling basato sull'attenzione (*Attention Mechanism*). Questo modulo (classe `LearnedPooling`) impara dinamicamente ad assegnare un "peso di importanza" a ciascun chunk. Formalmente, dato un nodo i e i suoi K

embedding $\{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \ldots, \mathbf{x}_{i,K}\}$, la rete calcola un punteggio di attenzione $s_{i,k}$ per ogni chunk facendolo passare attraverso una Multi-Layer Perceptron (MLP) a un singolo valore di output:

$$s_{i,k} = W_2\Big(\mathrm{ReLU}(W_1\mathbf{x}_{i,k} + b_1)\Big) + b_2 \tag{4.8}$$

I punteggi vengono poi normalizzati in una distribuzione di probabilità tramite la funzione Softmax, applicando simultaneamente la maschera per ignorare i chunk di padding (impostando il loro punteggio a $-\infty$, che il Softmax converte in 0):

$$\alpha_{i,k} = \frac{\exp(s_{i,k} \cdot \mathrm{mask}i, k)}{\sum j = 1^K \exp(s_{i,j} \cdot \mathrm{mask}_{i,j})} \tag{4.9}$$

Infine, l'embedding unificato del nodo i, denotato come $\mathbf{x}_i^{pool}$, è ottenuto come somma pesata dei chunk originali:

$$\mathbf{x}i^{pool} = \sum k = 1^K \alpha_{i,k}\mathbf{x}_{i,k} \tag{4.10}$$

Multi-Head Pooling: Diversificare la rappresentazione

Una singola funzione di attenzione potrebbe focalizzarsi solo su un aspetto specifico della semantica del nodo. Ispirandosi all'architettura dei Transformer [17], il pooling è stato espanso in una configurazione Multi-Head (classe `MultiHeadPooling`). Invece di utilizzare una sola MLP per calcolare i pesi, il modello istanzia H "teste" di attenzione indipendenti (nel codice, $H = 4$). Ciascuna testa apprende una diversa distribuzione di pesi $\alpha^{(h)}$ e produce un proprio embedding unificato $\mathbf{x}^{(h)}$. Questi H vettori vengono concatenati lungo l'ultima dimensione, generando un vettore di dimensione $H \times D$, che viene successivamente riportato alla dimensione originale D tramite una proiezione lineare (matrice dei pesi W_{proj}):

$$\mathbf{x}i^{final} = Wproj\left[\mathbf{x}_i^{(1)}, |, \mathbf{x}_i^{(2)}, |, \ldots, |, \mathbf{x}i^{(H)}\right] + bproj \tag{4.11}$$

Il Multi-Head Pooling permette al modello di estrarre e combinare simultaneamente diversi "punti di vista" dal testo frammentato, garantendo che nessuna sfumatura semantica rilevante vada persa prima del message passing.

Integrazione End-to-End nell'architettura DMON

Rispetto al modello base descritto nella sezione precedente, il flusso di elaborazione della variante *WithPooling*, illustrato schematicamente nella **Figura 4.3**, si articola in quattro fasi sequenziali profondamente interconnesse:

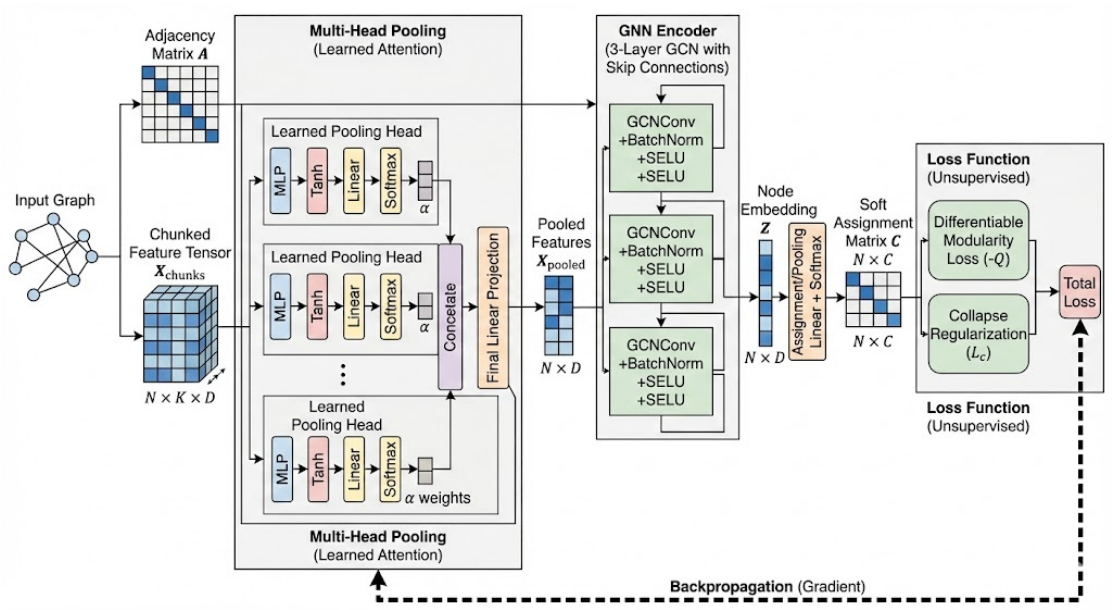

Figura 4.3: Architettura della variante DMON con Multi-Head Learned Pooling. A sinistra, il grafo viene processato separando la topologia (A) dai metadati testuali frammentati (X_{chunks}). Il blocco di pooling apprende i pesi di attenzione α per combinare i chunk in un unico embedding per nodo (X_{pooled}). Questo viene poi elaborato dal *GNN Encoder* per produrre lo spazio latente Z, da cui si ottiene la matrice di assegnamento *soft* C. L'intero processo è ottimizzato *end-to-end* minimizzando la loss composita non supervisionata.

1. **Aggregazione Semantica Locale**: Come visibile nel primo blocco dell'architettura, il tensore tridimensionale X_{chunks} viene elaborato dal modulo *Multi-Head Pooling*. Attraverso l'apprendimento di specifici pesi di attenzione (α) e una successiva proiezione lineare, il tensore viene compresso nella matrice bidimensionale unificata $X_{pooled} \in \mathbb{R}^{N \times D}$.

2. **Propagazione Strutturale**: Il risultato dell'aggregazione X_{pooled}, congiuntamente alla matrice di adiacenza A, viene passato al blocco centrale del *GNN Encoder*. Qui, l'informazione locale viene arricchita e propagata lungo gli archi (Foreign Keys) attraverso i tre layer convoluzionali con *skip connections*, ottenendo infine la matrice di embedding latente Z.

3. **Assegnamento**: Uscendo dall'encoder, la matrice Z viene convogliata nel modulo di *Assignment/Pooling* (costituito da una trasformazione lineare seguita da attivazione Softmax), dove viene mappata per produrre la matrice $C \in \mathbb{R}^{N \times C}$, contenente le probabilità di appartenenza di ciascun nodo ai vari cluster.

4. **Ottimizzazione End-to-End**: Il flusso in avanti (forward pass) termina con il calcolo della funzione di loss non supervisionata (Modularità Differenziabile + Collapse Penalty). Come indicato dalla freccia tratteggiata inferiore, il gradiente di questo errore viene retro-propagato non solo attraverso l'encoder, ma fino ai layer del modulo di attenzione iniziale, permettendo al sistema di calibrare dinamicamente l'importanza dei chunk testuali in funzione dell'ottimo topologico.

L'aspetto più dirompente di questa architettura risiede nel suo addestramento *end-to-end*. Poiché non viene fornita alcuna etichetta sui testi, il modulo di attenzione non sa "a priori" quali parole o chunk siano importanti. Tuttavia, essendo l'intera pipeline differenziabile, il gradiente della loss di modularità retro-propaga fino ai pesi dell'attenzione. In altre parole, il modello impara autonomamente a "prestare attenzione" ai chunk di testo che favoriscono la creazione di comunità dense e coese a livello strutturale. Questa sinergia tra la topologia del grafo (che guida l'errore) e la granularità semantica (che fornisce le feature) rappresenta un potenziamento significativo rispetto alla versione base di DMON, permettendo di districare schemi database in cui il ruolo funzionale delle tabelle è nascosto all'interno di lunghe descrizioni composite.

4.3 Il modello DGCluster

Sebbene il modello DMON consenta di ottimizzare la modularità in maniera differenziabile integrando direttamente il meccanismo di assegnamento all'interno

della rete neurale, tale approccio presenta alcune limitazioni operative, tra cui la necessità di specificare a priori il numero di cluster e la dipendenza da un layer di proiezione lineare per la determinazione delle comunità.

Per esplorare un paradigma complementare, in questo lavoro è stato implementato il modello DGCluster (Deep Graph Clustering) [18], che mantiene la modularità come principio guida dell'ottimizzazione ma separa esplicitamente la fase di apprendimento della rappresentazione dalla fase di clustering. In questa architettura, una Graph Neural Network è incaricata di apprendere embedding nodali strutturalmente e semanticamente informati, mentre l'assegnamento ai cluster viene delegato a un algoritmo di clustering applicato nello spazio latente. La specificità del modello risiede nell'integrazione dell'algoritmo gerarchico BIRCH, il quale consente di determinare i raggruppamenti emergenti sulla base della distribuzione geometrica degli embedding, evitando la definizione esplicita del numero di cluster. L'assegnamento prodotto nello spazio latente viene quindi utilizzato per calcolare una funzione obiettivo basata sulla modularità, che guida l'aggiornamento dei parametri della GNN.

Nelle sottosezioni seguenti verranno analizzati in dettaglio l'architettura del modello, il meccanismo di integrazione dell'algoritmo BIRCH e la funzione di loss che regola il processo di ottimizzazione.

4.3.1 Architettura del modello

Diversamente dagli approcci tradizionali di Graph Representation Learning, che si limitano a generare generici vettori densi, l'architettura neurale di DGCluster (realizzata nella classe `DGClusterGNN`) è progettata in modo mirato per modellare uno spazio latente che preservi le proprietà geometriche necessarie alla massimizzazione della modularità continua. L'input del modello è costituito dalla matrice delle feature nodali $X \in \mathbb{R}^{N \times D_{in}}$ (dove D_{in} rappresenta la dimensione degli embedding testuali delle tabelle estratti in fase di preprocessing) e dalla topologia del database rappresentata dalla matrice di adiacenza sparsa A. Il processo di codifica si sviluppa attraverso una sequenza di tre strati convoluzionali su grafo (`GCNConv`), che riducono progressivamente la dimensionalità dello spazio per distillare l'informazione:

1. Un primo strato proietta l'input a 256 dimensioni.
2. Un secondo strato riduce ulteriormente a 128 dimensioni.
3. Un terzo e ultimo strato genera l'embedding finale di dimensione D_{out} (tipicamente 64).

Per garantire la capacità della rete di apprendere relazioni non lineari, l'uscita dei primi due strati convoluzionali è processata dalla funzione di attivazione **SELU**

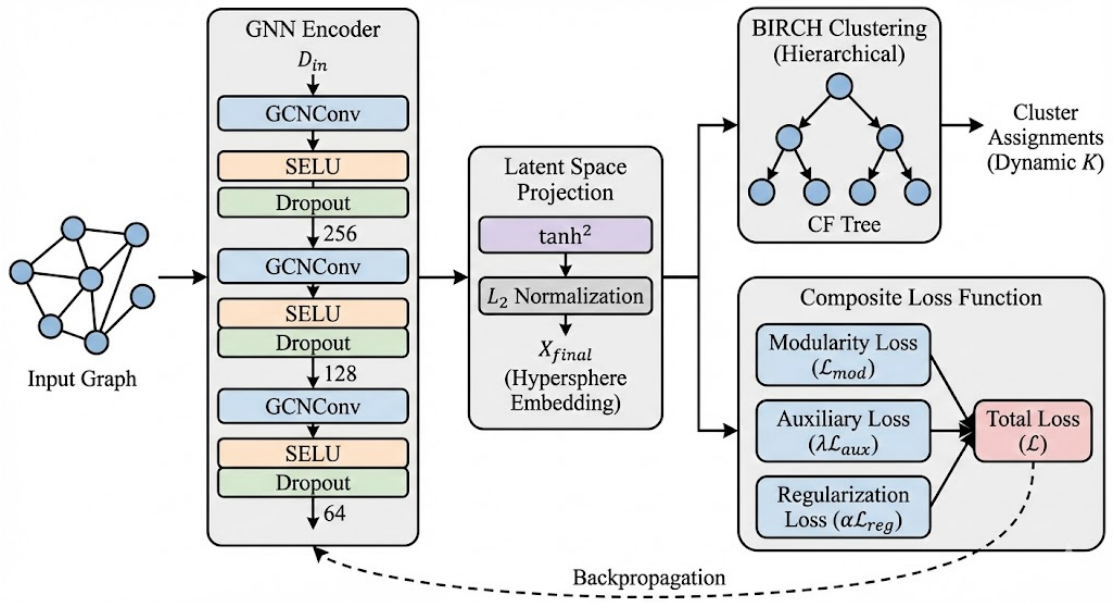

Figura 4.4: Architettura del modello DGCluster con integrazione BIRCH. Il grafo di input (A, X) viene elaborato da un *GNN Encoder* composto da tre strati convoluzionali (GCNConv) intervallati da attivazioni SELU e Dropout. I vettori latenti estratti subiscono una catena di trasformazioni non lineari ($\tanh^2$ e normalizzazione L_2) che li proietta sulla superficie di un'ipersfera di raggio unitario (X_{final}). Da questo spazio latente si diramano due processi: da un lato, l'algoritmo gerarchico BIRCH che costruisce un *CF Tree* per dedurre autonomamente il numero e l'assegnamento dei cluster; dall'altro, le coordinate vettoriali vengono utilizzate per calcolare la *Loss composita* (Modularità, Ausiliaria e Regolarizzazione), il cui gradiente retro-propaga per ottimizzare i pesi della rete in modo non supervisionato.

(*Scaled Exponential Linear Unit*). L'utilizzo della SELU, abbinata a strati di Dropout, favorisce la regolarizzazione del modello e previene il fenomeno dell'esplosione o della scomparsa dei gradienti, mantenendo stabili le attivazioni interne della rete. L'aspetto più peculiare e distintivo di questa architettura risiede tuttavia nelle trasformazioni finali applicate all'uscita del terzo strato convoluzionale. Affinché il prodotto scalare tra gli embedding di due nodi possa essere interpretato come una misura di "affinità di cluster" (necessaria per il calcolo della modularità), i vettori grezzi $H \in \mathbb{R}^{N \times D_{out}}$ vengono sottoposti a tre operazioni sequenziali:

$$H^{(1)} = \frac{H}{\sum H} \quad \longrightarrow \quad H^{(2)} = \tanh^2(H^{(1)}) \quad \longrightarrow \quad X_{final} = \frac{H^{(2)}}{|H^{(2)}|_2} \tag{4.12}$$

Questa catena di normalizzazioni svolge un ruolo cruciale:

- La divisione per la somma globale introduce un fattore di scala condiviso.
- L'applicazione del quadrato della tangente iperbolica ($\tanh^2$) forza tutti i valori dell'embedding a essere strettamente positivi e limitati nel range $[0, 1)$, simulando un comportamento simile a una probabilità senza imporre il vincolo a somma unitaria (come farebbe un Softmax) che limiterebbe l'indipendenza delle singole dimensioni.
- Infine, la normalizzazione L_2 proietta tutti i vettori sulla superficie di un'ipersfera di raggio unitario. In questo spazio, la similarità del coseno tra due nodi coincide con il loro prodotto scalare, rendendo le computazioni successive per la loss estremamente efficienti e geometricamente interpretabili.

4.3.2 Integrazione dell'algoritmo BIRCH

Una volta ottenuti gli embedding nodali normalizzati X_{final}, lo spazio latente risultante presenta una struttura geometrica in cui la vicinanza tra vettori riflette affinità strutturali e semantiche tra le tabelle del database. A questo punto, il problema si riduce alla determinazione di partizioni discrete coerenti con tale organizzazione spaziale.
Nel framework DGCluster, questa fase non è implementata tramite un layer neurale parametrico, ma è affidata all'algoritmo gerarchico BIRCH (*Balanced Iterative Reducing and Clustering using Hierarchies*) [19]. La scelta di BIRCH non è motivata esclusivamente da ragioni di scalabilità, ma dalla sua capacità di costruire una rappresentazione gerarchica compatta della distribuzione dei dati, adattandosi dinamicamente alla densità locale dello spazio latente.
BIRCH organizza progressivamente gli embedding all'interno di una struttura ad albero denominata *Clustering Feature Tree* (CF Tree), nella quale ogni nodo

contiene un riassunto statistico del sottoinsieme di punti assegnati. Tale riassunto, detto *Clustering Feature* (CF), è formalmente definito dalla tripla:

$$\mathrm{CF} = (N, \mathbf{LS}, \mathbf{SS})$$

dove N rappresenta il numero di punti aggregati, $\mathbf{LS} = \sum_{i=1}^{N} \mathbf{x}_i$ è la somma lineare degli embedding e **SS** è la somma dei quadrati. Questa rappresentazione compatta consente di calcolare in modo efficiente centroide, raggio e varianza del cluster senza dover memorizzare esplicitamente tutti i punti.

Nel contesto di questa implementazione, l'algoritmo viene configurato con il parametro `n_clusters=None`, eliminando la fase finale di ricondensazione globale in un numero prefissato di cluster. La granularità della partizione è quindi determinata unicamente dal parametro `threshold`, che definisce il raggio massimo consentito per ciascun nodo foglia del CF Tree. Operativamente, un nuovo embedding viene assorbito dal sotto-cluster più vicino qualora l'incremento del raggio resti inferiore alla soglia; in caso contrario, viene generato un nuovo nodo foglia.
Questo meccanismo produce una partizione adattiva guidata esclusivamente dalla geometria dello spazio latente appreso dalla GNN. Di conseguenza, il numero di cluster emerge implicitamente dalla distribuzione degli embedding, anziché essere imposto a priori. Inoltre, la natura gerarchica dell'albero consente una gestione più robusta di eventuali strutture multi-scala o punti isolati, che possono essere mantenuti in sotto-cluster distinti qualora risultino significativamente distanti dai centroidi esistenti.
Nel framework DGCluster, l'assegnamento discreto prodotto da BIRCH non rappresenta un semplice output finale, ma viene utilizzato per calcolare la funzione di modularità differenziabile che guida l'aggiornamento dei parametri della rete. In questo modo, la qualità del clustering gerarchico retroagisce sull'apprendimento della rappresentazione, chiudendo il ciclo tra struttura latente e ottimizzazione topologica.

4.3.3 Funzione di loss: Ottimizzazione e Regolarizzazione

L'addestramento della GNN nel framework DGCluster segue un regime principalmente non supervisionato, governato da una funzione obiettivo composita (`dgcluster_loss`). A differenza dei metodi di classificazione standard, qui l'ottimizzazione è calcolata interamente attraverso operazioni matriciali basate sui prodotti scalari degli embedding nello spazio latente. L'obiettivo è modellare una geometria vettoriale che codifichi implicitamente la topologia dei domini del database. La loss totale $\mathcal{L}$ si compone di tre termini fondamentali, bilanciati da iperparametri specifici:

$$\mathcal{L} = \mathcal{L}_{mod} + \lambda \mathcal{L}_{aux} + \alpha \mathcal{L}_{reg}$$

1. Loss di Modularità Differenziabile ($\mathcal{L}_{mod}$)

Il termine principale e trainante della funzione di costo mira a massimizzare la modularità continua derivata dalla matrice degli embedding normalizzati $X \in \mathbb{R}^{N \times D_{out}}$. Come descritto nel paper [18], la tradizionale formula spettrale discreta viene rilassata in uno spazio continuo. In questo spazio, il prodotto scalare tra due vettori rappresenta l'affinità (o la probabilità di co-appartenenza) tra due tabelle. Per questioni di efficienza computazionale e memoria, la modularità non viene calcolata istanziando l'ingombrante matrice densa di modularità B (di dimensione $N \times N$), ma viene scomposta algebricamente sfruttando la proprietà ciclica della traccia. Definita A la matrice di adiacenza sparsa ed m il numero totale di archi, il calcolo viene diviso in due componenti. Il primo termine valuta la forza delle connessioni topologiche reali proiettate nello spazio latente:

$$\text{Termine Effettivo} = \text{Tr}(X^T A X)$$

Il secondo termine agisce da penalità, approssimando la connettività attesa in un grafo casuale equivalente, sfruttando il vettore colonna dei gradi nodali $\mathbf{d}$:

$$\text{Termine Atteso} = \frac{1}{2m} \sum_{i=1}^{D_{out}} (X^T \mathbf{d})_i^2 = \frac{1}{2m} \|X^T \mathbf{d}\|_2^2$$

Poiché i framework di Deep Learning minimizzano le funzioni di costo tramite discesa del gradiente, la loss strutturale da minimizzare è definita come l'opposto della modularità:

$$\mathcal{L}_{mod} = -\frac{1}{2m} \left(\text{Tr}(X^T A X) - \frac{1}{2m} \|X^T \mathbf{d}\|_2^2 \right)$$

Operativamente, l'algoritmo implementato risulta altamente scalabile: l'utilizzo di tensori sparsi, per le moltiplicazioni che coinvolgono A riduce la complessità computazionale da quadratica a lineare rispetto al numero di vincoli di Foreign Key nel database.

2. Obiettivo Ausiliario ($\mathcal{L}_{aux}$)

Oltre alla pura ottimizzazione topologica, il framework si distingue per la sua flessibilità, prevedendo l'integrazione di un segnale guida addizionale controllato dall'iperparametro λ. In scenari reali, un *Data Engineer* potrebbe possedere conoscenze a priori sull'architettura, sapendo ad esempio che un gruppo specifico di tabelle appartiene indubbiamente al dominio "Risorse Umane". Fornendo in input

una matrice di assegnamenti noti (strutturata come matrice *one-hot* o probabilità *soft* C), la rete viene forzata ad allineare il proprio spazio latente X a tali vincoli. Matematicamente, la loss ausiliaria minimizza la distanza al quadrato basata sulla norma di Frobenius tra le due matrici ($\|X - C\|_F^2$), la quale, espansa algebricamente e normalizzata per il numero di coppie possibili (N^2), assume la forma:

$$\mathcal{L}_{aux} = \frac{1}{N^2}\Big(\mathrm{Tr}(C^T C) + \mathrm{Tr}(X^T X) - 2\mathrm{Tr}(X^T C)\Big)$$

L'attivazione di questo termine ($\lambda > 0$) trasforma elegantemente l'architettura da un modello puramente non supervisionato a uno semi-supervisionato, permettendo di ancorare l'apprendimento a *ground truth* parziali e guidare la GNN verso soluzioni semanticamente più aderenti alla logica di business.

3. Regolarizzazione dello Spazio Latente ($\mathcal{L}_{reg}$)

Il terzo e ultimo termine, governato dal parametro α, svolge il ruolo di stabilizzatore geometrico fondamentale. In assenza di una regolarizzazione esplicita, la massimizzazione della modularità tramite reti neurali risulta infatti altamente vulnerabile a soluzioni degeneri o a *minimi locali banali* (il cosiddetto *representation collapse*). La rete potrebbe, per esempio, conseguire un valore di modularità apparentemente elevato assegnando a tutte le tabelle uno stesso embedding identico, facendo collassare l'intero database in un'unica macro-comunità priva di significato. Per evitare questo comportamento, la funzione di regolarizzazione calcola il vettore somma di tutti gli embedding e penalizza il quadrato della sua norma:

$$\mathcal{L}_{reg} = \left(\frac{1}{N^2} \sum_{j=1}^{D_{out}} \Big(\sum_{i=1}^{N} X_{i,j} \Big)^2 \right)^2$$

Il significato geometrico di questa equazione è profondo. Sapendo che gli embedding X risiedono sulla superficie di un'ipersfera di raggio unitario (grazie alla normalizzazione L_2 finale dell'encoder), la norma della loro somma vettoriale raggiunge il massimo se tutti i vettori sono collineari (collasso). Al contrario, minimizzare questa loss (`reg_loss`) costringe la GNN a "spalmare" e respingere gli embedding verso direzioni ortogonali o diametralmente opposte. In questo modo, si garantisce l'utilizzo dell'intera dimensionalità latente disponibile, massimizzando la distanza spaziale tra domini funzionali logicamente non correlati. L'ottimizzazione simultanea ed equilibrata di questi tre gradienti ($\mathcal{L}_{mod}$, $\mathcal{L}_{aux}$, $\mathcal{L}_{reg}$) costituisce il vero motore computazionale del modello DGCluster. È proprio la qualità di questo spazio vettoriale finemente calibrato a permettere, nel passaggio successivo, all'algoritmo BIRCH di estrarre una gerarchia di cluster coerente e ad alta risoluzione.

Capitolo 5

Metriche di valutazione del clustering

La progettazione e l'addestramento di modelli di Community Detection e Deep Graph Clustering, per quanto complessi e matematicamente fondati, rappresentano solo una parte della pipeline analitica. In un contesto di apprendimento puramente non supervisionato, in cui mancano etichette di verità (*ground truth*) che indichino a priori quali tabelle appartengano a determinati domini funzionali, il problema della valutazione quantitativa e oggettiva dei risultati assume un'importanza cruciale. Come è possibile stabilire se la partizione generata dall'algoritmo di Leiden sia qualitativamente superiore o inferiore a quella prodotta da DMON o DGCluster?

Per rispondere a questa domanda e per comprendere a fondo le prestazioni delle diverse tecniche adottate durante lo sviluppo del progetto Dataslice AI, si è reso necessario implementare un solido framework di valutazione. Poiché i grafi estratti dai database relazionali sono intrinsecamente caratterizzati da una duplice natura — una rigorosa struttura topologica dettata dai vincoli di integrità e un ricco contenuto informativo descritto dai metadati testuali — valutare la qualità di un cluster osservando un'unica dimensione risulterebbe riduttivo e fuorviante. Per questo motivo, la valutazione dei modelli è stata condotta attraverso un approccio multi-prospettico, articolato in tre categorie fondamentali di metriche, ciascuna progettata per isolare e quantificare un aspetto specifico del clustering:

- **Metriche topologiche**: Si concentrano esclusivamente sulla struttura a grafo del database, misurando la densità e la coesione dei collegamenti (Foreign Keys) all'interno delle comunità e la separazione tra comunità diverse. Queste metriche operano in modo totalmente "cieco" rispetto al significato delle tabelle, valutando la bontà della partizione basandosi solo sulla teoria delle reti.

- **Metriche semantiche**: Ignorano completamente i vincoli strutturali e valutano i cluster esclusivamente dal punto di vista del significato. Sfruttando il concetto di similarità vettoriale degli *embedding* generati dall'LLM all'interno dello spazio latente, queste metriche verificano se le tabelle assegnate allo stesso gruppo condividano effettivamente il medesimo dominio di business (es. "Vendite" o "Risorse Umane").

- **Metriche ibride**: Rappresentano il punto di convergenza del framework di valutazione. Sono state implementate con lo scopo di fondere le due prospettive precedenti, cercando di quantificare la sinergia e i compromessi (*trade-off*) tra la coesione strutturale e la coerenza semantica, restituendo un indicatore olistico della qualità dei domini funzionali individuati.

Nelle sezioni seguenti verranno analizzate in dettaglio le formulazioni matematiche e le logiche operative di ciascuna di queste metriche, fornendo gli strumenti concettuali necessari per interpretare criticamente i risultati sperimentali che verranno presentati nel capitolo conclusivo.

5.1 Metriche topologiche

La prima categoria di valutazione adottata in questo studio ignora completamente le informazioni semantiche e si focalizza unicamente sulla qualità strutturale della partizione. Un database relazionale ben progettato dovrebbe presentare domini applicativi in cui i moduli interni risultano fortemente interconnessi tra loro, mantenendo al contempo legami più sporadici con i moduli esterni. Le metriche topologiche hanno proprio l'obiettivo di misurare questa dinamica, esaminando la distribuzione degli archi (i vincoli di Foreign Key) rispetto ai confini dei cluster individuati. Nell'implementazione realizzata, la valutazione topologica di ciascun modello di clustering restituisce un insieme di tre metriche principali: Modularità, Coverage e Mean Conductance.

5.1.1 Modularità

Come ampiamente discusso nei capitoli precedenti, la modularità (Q) rappresenta il la valutazione per eccellenza della Community Detection. Essa misura la differenza tra la frazione di archi che cadono all'interno delle comunità e la frazione che ci si aspetterebbe se gli archi fossero distribuiti casualmente (mantenendo inalterato il grado dei nodi). Riprendendo il concetto in breve, sia $G = (V, E)$ il grafo non orientato rappresentante lo schema del database, con matrice di adiacenza A. Data una partizione in comunità $C = \{C_1, C_2, \ldots, C_k\}$, la modularità è definita come:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

dove m è il numero totale di archi, k_i e k_j sono i gradi dei nodi i e j, c_i e c_j sono le comunità di appartenenza, e la funzione delta di Kronecker $\delta(c_i, c_j)$ vale 1 se i due nodi appartengono allo stesso cluster, 0 altrimenti. Un valore elevato di Q (prossimo a 1) indica una partizione eccellente in cui le tabelle di uno stesso cluster sono fortemente interconnesse. Al contrario, un valore prossimo allo 0 o negativo indica che la divisione in cluster non è migliore di una assegnazione casuale. Nell'architettura di valutazione, questa metrica è calcolata utilizzando la funzione nativa `modularity` della libreria NetworkX.

5.1.2 Copertura

La copertura (*Coverage*) è una metrica intuitiva e diretta che quantifica la percentuale globale di vincoli strutturali preservati all'interno delle comunità rispetto al totale delle connessioni del database. In termini pratici, misura quanta parte della "complessità relazionale" viene catturata internamente ai cluster creati dal modello. Matematicamente, la copertura di una partizione C è definita come il rapporto tra la somma degli archi interni a tutte le comunità e il numero totale di archi del grafo m:

$$\text{Coverage}(C) = \frac{\sum_{i=1}^{k} |E(C_i)|}{m}$$

dove $E(C_i)$ rappresenta l'insieme degli archi i cui nodi estremi appartengono entrambi alla medesima comunità C_i. Nell'algoritmo sviluppato (`compute_coverage`), questa metrica viene computata estraendo per ogni cluster il relativo sottografo indotto e sommandone gli archi. Un valore di Coverage pari a 1.0 implicherebbe l'assenza totale di connessioni tra cluster diversi (partizioni isolate), un caso limite irrealistico nei database reali. L'obiettivo pratico è ottenere un valore di Coverage elevato, indicando che la maggior parte delle operazioni di JOIN avverrebbe presumibilmente all'interno dello stesso modulo logico, limitando le dipendenze inter-dominio.

5.1.3 Conduttanza Media

Se Modularity e Coverage valutano la coesione interna globale, la Conduttanza (*Conductance*) sposta l'attenzione sui confini delle comunità, misurando l'efficacia del "taglio" (*cut*) operato dall'algoritmo. In teoria dei grafi, un buon cluster è un gruppo di nodi che ha molte connessioni interne ma "disperde" pochissime

connessioni verso l'esterno. La conduttanza formalizza questo concetto per una singola comunità $S \subset V$:

$$\Phi(S) = \frac{\text{cut}(S, V \setminus S)}{\min(\text{vol}(S), \text{vol}(V \setminus S))}$$

dove:

- $\text{cut}(S, V \setminus S)$ è il numero di archi che connettono un nodo dentro S a un nodo fuori da S.
- $\text{vol}(S)$ è il volume della comunità, definito come la somma dei gradi di tutti i nodi appartenenti a S ($\sum_{u \in S} \deg(u)$). Il denominatore utilizza il minimo tra il volume del cluster e il volume del resto del grafo per normalizzare la metrica ed evitare distorsioni legate alla dimensione della comunità.

A differenza di Q e Coverage, la conduttanza è una metrica di costo: valori più bassi indicano partizioni migliori. Una conduttanza vicina allo 0 significa che il cluster è quasi completamente isolato, comportandosi come un "collo di bottiglia" informativo. Nel framework implementato (`mean_conductance`), l'algoritmo itera su ogni singola comunità generata dal modello, ne calcola la conduttanza specifica ignorando i casi limite (cluster vuoti o coincidenti con l'intero grafo), e restituisce la media aritmetica di tali valori. Questa operazione permette di valutare se l'algoritmo tenda a produrre tagli netti sui confini funzionali delle tabelle, o se invece generi raggruppamenti "permeabili" e frammentati.

5.2 Metriche semantiche

Mentre le metriche topologiche si focalizzano esclusivamente sulla presenza o assenza di vincoli strutturali (Foreign Keys), le metriche semantiche valutano la qualità del clustering da una prospettiva puramente legata al dominio informativo. In un database aziendale, tabelle che appartengono allo stesso dominio funzionale (ad esempio, *Dipendenti*, *Stipendi*, *Presenze*) dovrebbero condividere un vocabolario e un significato affini, indipendentemente da come sono fisicamente collegate nello schema. Per quantificare questa coerenza, le metriche semantiche operano non sul grafo, ma sullo spazio vettoriale degli embedding testuali (X_{sem}) estratti in fase di pre-elaborazione (es. tramite LLM). La misura fondamentale alla base di queste metriche è la Similarità del Coseno (*Cosine Similarity*), che valuta l'allineamento direzionale tra due vettori $\mathbf{x}_i$ e $\mathbf{x}_j$ nello spazio latente:

$$S_{ij} = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}$$

Nell'implementazione sviluppata (`compute_sem_metrics`), a partire dalla matrice di similarità globale $S \in \mathbb{R}^{N \times N}$, vengono calcolate due grandezze intermedie fondamentali:

1. **Similarità Intra-cluster (Sim_{intra})**: Misura la coesione semantica interna delle comunità. Per ogni cluster identificato, viene calcolata la similarità media tra tutte le coppie distinte di nodi al suo interno. Il valore globale Sim_{intra} è la media aritmetica di queste similarità per tutti i cluster validi. Un valore elevato indica che le tabelle nello stesso gruppo parlano dello stesso "argomento".

2. **Similarità Inter-cluster (Sim_{inter})**: Misura la sovrapposizione semantica tra comunità diverse. Per calcolare questa grandezza in modo computazionalmente efficiente (evitando di esplorare tutte le possibili coppie $N \times (N-1)/2$), l'algoritmo adotta una tecnica di *negative sampling*: estrae casualmente un campione rappresentativo (fino a 50.000 coppie) di nodi appartenenti a cluster *differenti* e ne calcola la similarità media. Un valore basso indica che i cluster descrivono concetti ben distinti.

A partire da queste due grandezze, il framework di valutazione definisce due metriche di sintesi principali, utilizzate per confrontare le prestazioni dei vari modelli: la Semantic Difference e il Semantic Ratio.

5.2.1 Semantic Difference

La Differenza Semantica è una metrica assoluta che quantifica il margine di separazione tra la coesione interna dei cluster e la loro vicinanza esterna. È definita semplicemente come la differenza algebrica tra le due similarità medie:

$$S_{sem\_diff} = Sim_{intra} - Sim_{inter}$$

Questa metrica risponde alla domanda: "Quanto i nodi all'interno di un cluster sono più simili tra loro rispetto a quanto lo siano con i nodi degli altri cluster?".

- Un valore positivo e alto è l'obiettivo ideale: significa che l'algoritmo ha raggruppato tabelle semanticamente molto affini e ha tenuto separati argomenti diversi.
- Un valore prossimo allo zero indica che l'assegnamento ai cluster è semanticamente casuale: la similarità tra due tabelle nello stesso gruppo è statisticamente indistinguibile da quella di due tabelle prese a caso nell'intero database.
- Un valore negativo indicherebbe un clustering patologico, in cui nodi semanticamente distanti vengono raggruppati insieme, mentre nodi affini vengono separati.

5.2.2 Semantic Ratio

Mentre la differenza semantica fornisce una misura lineare del margine, il Rapporto Semantico (*Semantic Ratio*) valuta la proporzione relativa tra la coesione e la separazione. È definito come:

$$S_{sem\_ratio} = \frac{Sim_{intra}}{Sim_{inter} + \epsilon}$$

dove ϵ (impostato a 1×10^{-9} nell'implementazione) è una piccola costante di stabilità numerica aggiunta per prevenire divisioni per zero nel caso teorico in cui i cluster siano perfettamente ortogonali tra loro. Questa metrica amplifica le differenze prestazionali tra i modelli, specialmente quando i valori assoluti di similarità sono molto schiacciati (cosa che accade spesso con gli embedding linguistici ad alta dimensionalità, dove i vettori tendono a occupare un cono ristretto dello spazio, portando tutti i valori di similarità del coseno a essere positivi). Un $S_{sem\_ratio} > 1$ certifica la validità semantica del clustering, e valori via via maggiori indicano partizioni in cui l'isolamento concettuale dei moduli applicativi è sempre più netto e ben definito.

5.3 Metriche ibride

Se le metriche topologiche e semantiche offrono visioni ortogonali e isolate del problema, le metriche ibride rappresentano il punto di convergenza del framework di valutazione. Negli schemi di database complessi, la verità funzionale risiede quasi sempre nell'intersezione tra la struttura relazionale e il significato dei dati. Per quantificare questa sinergia, sono state implementate due metriche ibride basate su approcci matematici differenti: la prima agisce direttamente a livello di grafo pesato (*Semantic-Topological Modularity*), mentre la seconda opera una combinazione convessa a posteriori dei punteggi (*Attributed Modularity*). Entrambi gli approcci richiedono l'utilizzo combinato del grafo originale $G = (V, E)$ e della matrice degli embedding semantici normalizzati X.

5.3.1 Semantic-Topological Modularity

Questa prima metrica fonde topologia e semantica operando una trasformazione strutturale sul grafo originale prima del calcolo della modularità. L'assunto di base è che non tutti i vincoli di Foreign Key abbiano lo stesso "peso funzionale": un arco che collega due tabelle semanticamente identiche dovrebbe avere un valore maggiore rispetto a un arco che collega due tabelle concettualmente distanti (es. una dipendenza tecnica verso un dizionario generico). L'algoritmo (`semantic_topological_modularity`) procede in due fasi:

1. **Costruzione del Grafo Pesato Semantico**: A partire dal grafo originale non orientato G, viene generato un nuovo grafo $H = (V, E, W)$ recante la medesima topologia. Tuttavia, a ogni arco $e = (u, v)$ esistente viene assegnato un peso continuo w_{uv} pari alla similarità del coseno tra gli embedding dei nodi collegati:

$$w_{uv} = \frac{\mathbf{x}_u \cdot \mathbf{x}_v}{\|\mathbf{x}_u\|\|\mathbf{x}_v\|}$$

2. **Calcolo della Modularità Pesata**: Sulla base della partizione C prodotta dal modello di clustering, viene calcolata la modularità standard di Newman sul grafo H, specificando il peso degli archi.

La formula della modularità pesata diviene quindi:

$$Q_{ST} = \frac{1}{2W_{tot}} \sum_{i,j} \left(w_{ij} - \frac{s_i s_j}{2W_{tot}} \right) \delta(c_i, c_j)$$

dove W_{tot} è la somma di tutti i pesi degli archi nel grafo e s_i è la forza (strength) del nodo i, definita come la somma dei pesi degli archi a esso incidenti ($\sum_j w_{ij}$). Questa metrica premia partizioni che non solo preservano un elevato numero di connessioni interne, ma in cui tali connessioni legano tabelle con una forte affinità semantica.

5.3.2 Attributed Modularity

Il secondo approccio (`attributed_modularity`) adotta una strategia di fusione di tipo *late-fusion*. Invece di alterare il grafo di partenza, calcola separatamente un punteggio di eccellenza topologica e un punteggio di eccellenza semantica, per poi combinarli linearmente. La metrica si compone di due elementi:

1. **Modularità Topologica (Q_{topo})**: La classica modularità calcolata sul grafo non pesato G.

2. **Omofilia Semantica (Hom_{sem})**: Un indicatore continuo che misura l'omogeneità vettoriale della partizione. Analogamente alla metrica *Semantic Difference* vista nella sezione precedente, l'algoritmo calcola la similarità coseno per tutte le coppie di nodi possibili, separando quelle intra-cluster da quelle inter-cluster. L'omofilia è definita come la differenza tra le medie aritmetiche di questi due insiemi:

$$Hom_{sem} = \text{Media}(\text{Sim}_{intra}) - \text{Media}(\text{Sim}_{inter})$$

Il punteggio finale Q_{attr} è calcolato come una combinazione convessa governata dall'iperparametro $\alpha \in [0, 1]$:

$$Q_{attr} = \alpha \cdot Q_{topo} + (1 - \alpha) \cdot Hom_{sem}$$

Nell'ambito degli esperimenti condotti in questa tesi, il parametro è stato impostato a $\alpha = 0.5$, assegnando così uguale importanza (50% e 50%) alla struttura relazionale imposta dallo sviluppatore del database e alla semantica estratta dai dati. Questa formulazione risulta particolarmente robusta: un modello otterrà un punteggio elevato di Q_{attr} solo se sarà in grado di massimizzare simultaneamente la coesione strutturale e la separazione concettuale dei cluster, fornendo l'indicatore definitivo per confrontare algoritmi puramente topologici (come Leiden) con architetture neurali avanzate (come DMON e DGCluster).

Capitolo 6

Esperimenti e risultati

Nei capitoli precedenti è stato tracciato un percorso teorico e architetturale finalizzato ad affrontare il complesso tema della modularizzazione dei database relazionali. A partire dalla formulazione del problema in termini di grafo e dal suo arricchimento semantico tramite Large Language Models, il lavoro ha progressivamente orientato l'attenzione verso la progettazione di modelli avanzati di *Deep Graph Clustering*, come DMON e DGCluster, per giungere infine alla definizione di rigorose metriche di valutazione di tipo topologico, semantico e ibrido. Lo scopo di questo sesto capitolo è trasferire l'intero impianto metodologico dal livello teorico a quello empirico, verificandone efficacia e robustezza tramite una mirata campagna sperimentale.

Per comprendere a fondo il comportamento, i punti di forza e i limiti delle diverse tecniche proposte, i modelli sono stati testati su schemi di database specifici, scelti per la loro varietà intrinseca e per la fitta rete di vincoli di integrità referenziale che li caratterizza. La trattazione seguirà il naturale flusso logico della pipeline di analisi, partendo proprio dalla presentazione dei dataset oggetto di studio, per poi scendere nel dettaglio del setup sperimentale. In questa fase cruciale verrà posta particolare attenzione alle diverse strategie di *text embedding* adottate per tradurre le descrizioni testuali in vettori continui, mettendo a confronto l'elaborazione dell'intera descrizione della tabella con tecniche più granulari basate sul partizionamento in *batch* (o *chunking*).

Successivamente, verrà descritto il processo di calibrazione dei modelli, illustrando le metodologie di *Grid Search* impiegate per l'ottimizzazione degli iperparametri e le tecniche di post-elaborazione applicate per raffinare gli assegnamenti, come l'integrazione del clustering K-Means.

Infine, il capitolo culminerà nell'analisi quantitativa e qualitativa dei risultati. Attraverso le metriche di valutazione precedentemente definite, sarà possibile tracciare un confronto analitico e diretto tra le diverse architetture e configurazioni.

Questo esame permetterà di evidenziare quale approccio garantisca il miglior compromesso tra la conservazione della topologia strutturale del database e la coerenza funzionale dei domini applicativi estratti, traendo così le conclusioni definitive sull'efficacia del *Graph Representation Learning*.

6.1 Dataset utilizzati

Per condurre una validazione empirica rigorosa e valutare la capacità di generalizzazione delle architetture di clustering proposte, la fase di test è stata progettata sfruttando tre distinti schemi di database relazionali. La selezione di questi dataset non è stata casuale, ma ha seguito una precisa logica di progressione dimensionale e di complessità strutturale. L'obiettivo è osservare il comportamento dei modelli nel passaggio da scenari ideali e controllati fino ad arrivare a schemi enterprise altamente frammentati.
Nello specifico, i primi due dataset sono di natura *open-source* (estratti da repository pubblici su GitHub) e fungono da *benchmark* controllati: il primo rappresenta un database di dimensioni estremamente ridotte (funzionale come *baseline* o "toy dataset"), mentre il secondo introduce una media complessità con 71 tabelle. Il terzo e ultimo dataset costituisce invece il vero banco di prova industriale dell'intero progetto: si tratta di uno schema relazionale reale e in produzione, fornito direttamente da un cliente interessato ai servizi della piattaforma Dataslice. Questo database, che gestisce il dominio assicurativo e legale dei sinistri stradali, presenta un'enorme complessità topologica, contando 1027 tabelle interconnesse da 440 vincoli relazionali.
Di seguito vengono analizzate nel dettaglio le caratteristiche strutturali e il dominio informativo di ciascun dataset, partendo dal caso di studio più semplice.

6.1.1 Dataset 1: Sistema di Noleggio (Aviano DB)

Il primo dataset analizzato, denominato convenzionalmente "Aviano DB" [20], rappresenta un classico schema relazionale progettato per la gestione di un'attività di noleggio di veicoli ed equipaggiamenti. Essendo un database di dimensioni contenute, esso fornisce un ambiente di test ideale in cui l'aggregazione logica delle tabelle è umanamente interpretabile e facilmente verificabile, permettendo di validare il corretto funzionamento delle pipeline di estrazione testuale e di calcolo delle metriche prima di scalare su sistemi più ampi.
Dall'analisi del file di schema (JSON), il grafo relazionale estratto risulta composto dalle seguenti metriche topologiche:

- **13 Nodi (Tabelle)**: `location`, `customer`, `fuel_option`, `insurance`, `equipment_type`, `equipment`, `vehicle_type`, `vehicle`, `rental`,

`rental_has_equipment_type`, `rental_has_insurance`, `vehicle_has_equiment`, `rental_invoice`.

- **15 Archi (Vincoli di Foreign Key)**: che definiscono le dipendenze gerarchiche e associative tra le entità.

Nonostante le dimensioni ridotte, lo schema presenta una topologia interessante per gli algoritmi di clustering. Si possono infatti distinguere chiaramente tre macro-aree semantiche e funzionali:

1. **Dominio Inventario/Flotta**: centrato sulle tabelle `vehicle`, `equipment` e le rispettive tabelle anagrafiche e di tipologia.
2. **Dominio Transazionale (Noleggio)**: il cuore del database, rappresentato dalla tabella *hub* `rental` (fortemente connessa) e dalle sue tabelle di collegamento (es. `rental_has_insurance`, `rental_has_equipment_type`).
3. **Dominio Anagrafico e Fatturazione**: comprendente entità periferiche ma cruciali come `customer` (cliente), `location` (sedi fisiche) e `rental_invoice` (fatturazione).

L'inclusione di questo dataset nella campagna sperimentale ha lo scopo primario di verificare se i modelli di *Deep Graph Clustering* siano in grado di non effettuare *oversmoothing* su grafi molto piccoli, riuscendo a mantenere separati domini logicamente distinti anche in presenza di una distanza topologica (numero di "salti" o *hop*) molto ridotta tra le tabelle.

6.1.2 Dataset 2: Sistema Gestionale ERP (AdventureWorks DB)

Il secondo dataset selezionato per la campagna sperimentale rappresenta un significativo salto in termini di complessità e dimensioni. Estrapolato e parsato a partire dal noto database di benchmark Microsoft AdventureWorks [21], questo schema modella l'architettura informativa di un sistema ERP (*Enterprise Resource Planning*) completo, progettato per gestire le operazioni di una grande azienda manifatturiera.
L'inclusione di questo dataset risponde alla necessità di testare i modelli su una struttura relazionale che presenti una gerarchia profonda, con tabelle altamente connesse (veri e propri hub logici) e numerose tabelle satellite o di dizionario. Dall'analisi del file JSON elaborato durante la fase di preprocessing, il grafo estratto presenta le seguenti caratteristiche topologiche:

- **71 Nodi (Tabelle)**: che spaziano da anagrafiche complesse (`Customer`, `Employee`, `Product`) a log di sistema (`DatabaseLog`, `ErrorLog`) fino a entità puramente transazionali (`SalesOrderHeader`, `PurchaseOrderDetail`).

- **104 Archi (Vincoli di Foreign Key)**: che formano una fitta rete di dipendenze incrociate, rendendo la separazione dei cluster topologici una sfida non banale per gli algoritmi standard.

A differenza del dataset "Aviano", questo schema ERP possiede una strutturazione funzionale nativa particolarmente preziosa ai fini della valutazione. Le tabelle al suo interno sono infatti formalmente suddivise in sei *schema* logici distinti a livello di database, i quali fungono da veri e propri moduli applicativi di base. Nello specifico, l'architettura informativa si articola attorno a un modulo **Person**, dedicato alla gestione dell'anagrafica centralizzata e dei contatti aziendali. Il ciclo transazionale è diviso in due aree speculari: da un lato lo schema **Sales** governa l'intero ciclo attivo, occupandosi dei clienti, dei territori, delle vendite e delle offerte; dall'altro, lo schema **Purchasing** gestisce il ciclo passivo, tracciando i rapporti con i fornitori e gli ordini di acquisto. Il nucleo operativo e manifatturiero risiede invece nel modulo **Production**, l'area più vasta del database, progettata per orchestrare il catalogo prodotti, le distinte base e l'inventario fisico. A supporto della struttura organizzativa interna interviene lo schema **HumanResources**, incaricato di gestire le informazioni relative ai dipendenti, all'organizzazione in reparti e ai turni aziendali. Infine, l'architettura si completa con il modulo di sistema **dbo**, che raggruppa le tabelle di servizio necessarie per il monitoraggio interno, come i log degli errori e delle operazioni di database.

Questa chiara compartimentazione nativa fornisce un'eccellente "verità di base" (*ground truth*) per valutare l'accuratezza degli algoritmi, permettendo di verificare se i modelli riescano effettivamente a dedurre questi domini logici a partire dai soli vincoli relazionali e dall'analisi semantica L'obiettivo dei test condotti su questo schema da 71 tabelle è valutare la capacità delle architetture DMON e DGCluster di districare questo "groviglio" relazionale. In particolare, si intende verificare se le metriche ibride (topologiche-semantiche) riescano a guidare l'algoritmo nel ricostruire una frammentazione coerente con i 6 domini logici nativi, evitando il collasso di tabelle funzionalmente diverse (es. `Sales` e `Purchasing`) in un'unica gigantesca comunità solo perché indirettamente connesse tramite anagrafiche condivise.

6.1.3 Dataset 3: Sistema Assicurativo Reale (Caso Industriale)

Il terzo e ultimo dataset costituisce il principale banco di prova del progetto, rappresentando uno scenario industriale autentico e non un ambiente di *benchmark* pre-confezionato. Fornito direttamente da un cliente reale per validare le potenzialità della piattaforma Dataslice, il database modella il dominio estremamente complesso e stratificato della gestione dei sinistri stradali e del relativo ecosistema assicurativo.

L'esportazione originale (fornita tramite file SQL) comprendeva un'architettura massiva composta da oltre 1000 tabelle e circa 440 relazioni. Tuttavia, trattandosi di un'estrazione parziale progettata unicamente per valutare i risultati preliminari del tool, lo schema presentava una forte criticità strutturale: la documentazione dei vincoli relazionali era incompleta. Di conseguenza, un numero cospicuo di entità risultava "spaiato", ovvero completamente privo di collegamenti di Foreign Key verso il resto del database.
Nel contesto del *Graph Representation Learning*, la presenza di un numero molto elevato di nodi isolati (cioè nodi con grado topologico nullo) all'interno di un grafo fortemente frammentato compromette in maniera significativa la qualità delle predizioni. Oltre ad alterare in modo sostanziale le metriche di modularità e conduttanza, tali nodi ostacolano la corretta propagazione dei segnali (il *message passing*) nelle GNN, di fatto riducendo il problema del clustering su grafo a una semplice clusterizzazione nello spazio vettoriale. Per ovviare a questo problema e garantire la coerenza dell'esperimento, è stata implementata una rigorosa fase di *preprocessing* topologico. Il grafo grezzo è stato sottoposto a un filtro che ha rimosso sistematicamente tutti i nodi disconnessi e le componenti isolate irrilevanti.

A seguito dell'operazione di pulizia, il dataset si è consolidato in un'unica componente relazionale interconnessa formata da: 116 Nodi (Tabelle) e 128 Archi (Vincoli strutturali). Nonostante la drastica riduzione dimensionale, questo nucleo rappresenta il cuore nevralgico della logica di business del cliente. Analizzando la semantica a grandi linee, il grafo raggruppa tabelle che gestiscono l'intero ciclo di vita del sinistro assicurativo. Le informazioni modellate spaziano dall'articolazione territoriale della compagnia (gerarchie di agenzie e sub-agenzie logiche) alla registrazione degli eventi accidentali. Sono inoltre incluse ampie sezioni dedicate al tracciamento delle coperture di polizza, all'autorizzazione dei pagamenti, alla gestione delle spese mediche e di perizia, fino al tracciamento storico delle pratiche e dei complessi contenziosi legali.

A differenza del dataset AdventureWorks, in cui i moduli erano esplicitamente dichiarati a livello di *schema*, in questo caso industriale i confini tra i domini funzionali non sono noti a priori. L'applicazione dei modelli DMON e DGCluster su questo dataset ha quindi l'obiettivo di dimostrare come l'Intelligenza Artificiale possa far emergere ordine e partizioni logiche di business da uno schema *legacy* stratificato e apparentemente caotico.

6.2 Setup sperimentale

Definiti i dataset di validazione e le loro peculiarità topologiche, è fondamentale delineare l'ambiente di test e le metodologie operative impiegate per condurre gli esperimenti. Nel contesto del *Deep Graph Clustering*, l'addestramento dei modelli

neurale rappresenta solo il nucleo centrale di una complessa pipeline analitica. La qualità dei risultati finali e la loro riproducibilità dipendono infatti in larga misura dalle scelte ingegneristiche effettuate nelle fasi di preparazione dei dati, di calibrazione del modello e di estrazione delle partizioni.

Per garantire un confronto equo e oggettivo tra le diverse architetture (DMON, DGCluster e le euristiche di base come Leiden), il setup sperimentale è stato standardizzato seguendo un flusso di elaborazione diviso in tre fasi sequenziali, le quali verranno esplorate nel dettaglio nelle sottosezioni successive.

6.2.1 Strategie di text embedding

Come anticipato nel primo capitolo, il passaggio fondamentale per abilitare l'apprendimento delle Graph Neural Networks su grafi attribuiti è la conversione dei metadati testuali in vettori numerici continui. Per questa delicata fase di codifica semantica, ci si è avvalsi dei servizi cloud forniti da AWS Bedrock, impiegando in particolar modo il modello *State-of-the-Art* `amazon.titan-embed-text-v2:0`. Per garantire un compromesso ottimale tra espressività semantica ed efficienza computazionale durante il training delle reti, la dimensionalità dell'embedding di output è stata fissata a 512 per tutte le sperimentazioni.

Prima di invocare il modello di embedding, si è resa necessaria una profonda riflessione sulla natura dell'input. Passare le descrizioni generate dall'LLM in formato testuale grezzo e destrutturato avrebbe rischiato di diluire l'informazione tecnica, rendendo difficile per il modello di embedding pesare correttamente nomi di colonne rispetto a descrizioni generiche. Per massimizzare la precisione dei vettori risultanti, è stata quindi implementata una rigorosa funzione di formattazione. Questa procedura programmaticamente estrae i campi chiave dal dizionario dei metadati di ogni nodo e li concatena utilizzando dei *tag* espliciti, creando un *template* standardizzato e altamente informativo.

Il formato adottato prevede la seguente struttura logica per ogni tabella:

1. [**TABLE**]: Il nome fisico della tabella nel database.
2. [**DESCRIPTION**]: La descrizione sintetica del suo ruolo.
3. [**BUSINESS**]: L'area funzionale o di business dedotta dall'LLM.
4. [**COLUMNS**]: L'elenco concatenato dei nomi delle colonne, corredato dalle rispettive descrizioni (separate da punto e virgola).

A titolo esemplificativo, l'elaborazione dei metadati relativi a una tabella anagrafica telefonica del terzo dataset (il caso industriale) produce la seguente stringa formattata, pronta per essere vettorializzata:

```
1 [TABLE]: TELEFONO
2 [DESCRIPTION]: Stores telephone numbers associated with
     individuals or entities involved in insurance claims,
     facilitating contact and communication during the claims
     management process.
3 [BUSINESS]: Claims Communication Management
4 [COLUMNS]:
5   COD_NOMINATIVO: Identifier code for the individual or entity
      associated with the telephone number.
6   COD_TELEFONO: Unique identifier or code for the telephone number
      assigned to an individual or entity.
```

Questa strutturazione rigida garantisce che il modello Titan v2 riceva un contesto chiaro e gerarchico, migliorando significativamente la qualità dello spazio latente semantico. A valle di questa standardizzazione del testo, per valutare l'impatto della lunghezza e della frammentazione dell'informazione sulle prestazioni di clustering, sono state implementate e confrontate tre differenti strategie di *text embedding*: l'approccio *Full description* e l'elaborazione in *batch* (o *chunking*).

Full description

La strategia *Full description* rappresenta l'approccio *baseline* più diretto e intuitivo. In questa configurazione, non viene applicata alcuna logica di partizionamento o troncamento del testo. L'intera stringa formattata prodotta nel passaggio precedente, comprensiva di tutte le colonne e delle relative descrizioni, viene trattata come un singolo documento monolitico.
Operativamente, per ogni singola tabella (nodo) presente nel grafo, il testo completo viene inviato al modello `amazon.titan-embed-text-v2:0` tramite una singola chiamata API ad AWS Bedrock. Il modello elabora l'intero contesto e restituisce un unico vettore denso di dimensione 512. Questo singolo embedding cattura olisticamente il significato della tabella, mediando tra l'informazione generale del business e i dettagli granulari delle singole colonne.

Una volta ottenuto il vettore, la pipeline di *preprocessing* lo inietta direttamente all'interno della struttura dati del grafo *NetworkX*, memorizzandolo come attributo del nodo sotto la chiave `"embedding"`. Al termine dell'elaborazione di tutti i nodi, l'intero grafo arricchito topologicamente e semanticamente viene serializzato e salvato in un *bucket* AWS S3. Questo salvataggio garantisce la persistenza del dato e permette alle successive fasi di training (condotte in PyTorch Geometric) di scaricare e caricare in memoria i tensori già computati, azzerando la latenza e i costi di calcolo legati a invocazioni ripetute all'LLM durante la Grid Search.

Elaborazione batch (Chunking)

La seconda strategia esplorata, denominata *Elaborazione batch* o *Chunking*, nasce per far fronte a un potenziale limite dell'approccio monolitico: il rischio di diluizione semantica. Nei database reali, specialmente in schemi industriali come il terzo dataset analizzato, alcune tabelle possono contenere decine o centinaia di colonne. L'invio di una stringa formattata così estesa al modello di embedding rischia di generare una rappresentazione vettoriale "appiattita", in cui il peso semantico di colonne specifiche e critiche (ad esempio chiavi esterne particolari o flag di stato) viene oscurato dal contesto generale del documento.
Per ovviare a questa problematica e ottenere vettori più densi di informazione locale, si è deciso di frammentare la descrizione di ogni singola tabella in segmenti testuali più piccoli (*chunk*), prima di procedere alla vettorializzazione. A tale scopo è stata adottata la libreria `langchain_text_splitters`, e in particolare la classe `RecursiveCharacterTextSplitter`. Questo strumento è progettato per non troncare il testo in modo cieco (che potrebbe spezzare a metà il nome di una colonna), ma per rispettare la struttura logica della stringa, provando a dividere sequenzialmente in corrispondenza di ritorni a capo, punteggiatura e spazi.

La configurazione del partizionatore ha richiesto una calibrazione accurata dei parametri. La dimensione massima di ogni frammento (`chunk_size`) è stata impostata a 300 caratteri testuali. Questa dimensione, volutamente contenuta, forza il modello AWS Titan v2 a generare embedding altamente focalizzati su sottoinsiemi ristretti di informazioni (es. la descrizione di business isolata, oppure gruppi di due o tre colonne alla volta).
Per scongiurare il rischio di perdere il contesto logico ai margini del taglio testuale, è stato introdotto un parametro di sovrapposizione (`chunk_overlap`) pari a 32 caratteri. Questa finestra di intersezione garantisce la continuità semantica tra un *chunk* e il successivo, assicurando che nessuna parola chiave venga compromessa da una divisione netta.

Operativamente, la stringa formattata di ciascuna tabella viene processata dal metodo `split_text`, che restituisce un array di sotto-testi. A questo punto, il processo entra in una logica di elaborazione a *batch*: la funzione
`add_embeddings_to_graph_with_chunks` itera su tutti i frammenti generati, interrogando il Large Language Model per ciascuno di essi. Il risultato è che a un singolo nodo del grafo non viene più associato un solo vettore, ma un insieme di vettori a 512 dimensioni (ognuno rappresentante una specifica sfaccettatura informativa della tabella). Analogamente alla strategia precedente, anche il grafo così arricchito viene archiviato in un bucket S3, rendendolo immediatamente disponibile per le future sessioni di training dei modelli. Questa strategia multi-vettoriale prepara il terreno per valutare se un'analisi semantica granulare (successivamente aggregata tramite operazioni di *pooling*) offra un vantaggio metrico o topologico

nell'individuazione dei domini applicativi rispetto alla singola *Full description*.

6.2.2 Grid search degli iperparametri

L'impiego di architetture neurali nel contesto del *Graph Representation Learning* non supervisionato richiede una fase di calibrazione critica per garantire che i modelli convergano verso soluzioni topologicamente e semanticamente stabili. Tra tutte le variabili architetturali, la sfida operativa più complessa riguarda il modello DMON. A differenza di algoritmi gerarchici dinamici (come DGCluster con BIRCH) o euristiche puramente *data-driven*, l'approccio *Soft Clustering* di DMON impone un vincolo strutturale rigido: il numero di cluster K da individuare deve essere obbligatoriamente prefissato a priori, in quanto definisce la dimensionalità del layer di proiezione finale (MLP) della rete. Poiché in scenari reali (come il database industriale analizzato nel terzo dataset) il numero di domini funzionali ottimali è un'incognita, si è reso necessario implementare una strategia di *Grid Search* esaustiva. L'obiettivo di questa procedura è esplorare iterativamente un intervallo definito di possibili valori per K (governato dai parametri di `start` ed `end` nel codice di training), addestrando un modello DMON indipendente per ciascuna configurazione. Il processo di esplorazione, eseguito in ambiente cloud tramite AWS SageMaker, prevede per ogni iterazione i seguenti passaggi:

1. **Inizializzazione e Training**: Viene istanziato un modello DMON con un numero di cluster target pari a i (con $i \in [\text{start}, \text{end})$). La rete viene addestrata per un numero fisso di epoche, minimizzando la funzione di *loss* composita (Collapse Penalty e Modularità differenziabile).

2. **Estrazione dei Risultati**: Al termine del training, la rete produce in output la matrice degli embedding aggregati (`x_pooled`) e il vettore contenente l'assegnazione discreta di ogni tabella al rispettivo cluster (`assignments`).

3. **Valutazione Ibrida**: Per decidere oggettivamente quale valore di K produca il partizionamento migliore, non ci si è limitati a osservare il solo valore della *loss* di training, che risulterebbe parziale. Si è invece adottata la metrica ibrida **Attributed Modularity (Q_{attr})**, definita nel Capitolo 5.

La scelta della metrica *Attributed Modularity* come criterio di selezione finale (*model selection*) rappresenta uno dei punti di forza del setup sperimentale. Come illustrato nello script di addestramento (`train_pool_entry.py`):

```
1 #Caricamento dell'embedding da S3 per il calcolo della
     metrica topologica+semantica
2 s3 = boto3.client("s3")
3
```

```
bucket_name = "amzn-s3-datasliceai-gnn-tesi-silvestri"
key = f"dataset/{args.name_dataset}/for_metrics/{args.name_dataset}_emb.pkl"
local_path = "/opt/ml/input/data/emb.pkl"

s3.download_file(bucket_name, key, local_path)

with open(local_path, "rb") as f:
    state_dict = pickle.load(f)

desc_embd = np.array([state_dict[node] for node in G.nodes()])

for i in range(args.start, args.end):
    print(f"--- Cluster: {i} ---")

    assignments, _, _, x_pooled = train_dmon_with_pooling(
        X_padded, mask, edge_index,
        num_clusters=i,
        epochs=args.epochs,
        hidden_channels=args.hidden_channels,
        embed_dim=args.embed_dim,
        lr=args.lr,
        collapse_lambda=args.collapse_lambda,
        pool_hidden=args.pool_hidden
    )

    sem_mod = attributed_modularity(G, desc_embd, assignments)

    print(f"  Semantic Topological Modularity: {sem_mod:.4f}")

    if best_assignments_pool is None or sem_mod > best_mean_mesures_pool:
        best_assignments_pool = assignments
        best_mean_mesures_pool = sem_mod
        best_x_pooled = x_pooled
        best_num_clusters_pool = i

print("\n--- Best Result ---")
print(f"Best number of clusters: {best_num_clusters_pool}")
print(f"Best mean composite measure: {best_mean_mesures_pool:.4f}")
print(f"Num clusters: {np.unique(best_assignments_pool)}")
```

Listing 6.1: Ciclo di addestramento iterativo per il modello DMON. La configurazione ottimale del numero di cluster viene determinata massimizzando la metrica *Attributed Modularity*.

per ogni iterazione l'algoritmo scarica i vettori semantici originali (`desc_embd`) direttamente dal bucket AWS S3. Successivamente, la funzione `attributed_modularity` valuta la partizione appena generata da DMON operando una combinazione convessa ($\alpha = 0.5$) tra:

- La **Modularità topologica** pura, calcolata sui vincoli di *Foreign Key* del grafo G.
- L'**Omofilia semantica**, calcolata come differenza tra la similarità coseno media intra-cluster e inter-cluster degli embedding testuali originali.

Ad ogni ciclo della *Grid Search*, il punteggio ottenuto viene confrontato con il massimo storico (`best_mean_mesures_pool`). Al termine dell'esplorazione, l'infrastruttura di training seleziona automaticamente il modello che ha registrato il valore di *Attributed Modularity* più elevato, eleggendo il relativo K come "numero di cluster ideale". Infine, i risultati definitivi di questo modello ottimale (assegnamenti e vettori latenti) vengono serializzati in formato JSON e tensore PyTorch (`.pt`) per le successive fasi di analisi e confronto.

6.2.3 Clustering con K-Means

Prima di procedere alla fase conclusiva di valutazione e confronto dei risultati, il setup sperimentale è stato arricchito con l'introduzione di un'ulteriore metodologia, inserita con la funzione di *baseline* algoritmica: l'applicazione del partizionamento K-Means. L'inclusione di questo approccio tradizionale risponde a una precisa domanda di ricerca: è possibile ottenere partizioni logiche di alta qualità affidandosi esclusivamente a metodi semplici e versatili, sfruttando unicamente la potenza rappresentativa dei moderni Large Language Models e ignorando del tutto la topologia relazionale del database? Applicando il K-Means direttamente ai vettori continui normalizzati generati dalla strategia *Full description*, l'esperimento isola volontariamente la sola componente semantica. Questa scelta crea un termine di paragone fondamentale per quantificare il reale valore aggiunto apportato successivamente dalle Graph Neural Networks, le quali hanno invece l'onere e il vantaggio di fondere il significato testuale con i vincoli strutturali di *Foreign Key*.

Esattamente come riscontrato nell'architettura DMON, l'algoritmo K-Means è parametrico e richiede la definizione a priori del numero di cluster (K) in cui suddividere lo spazio vettoriale. Essendo il numero di domini applicativi reali

sconosciuto a priori, la determinazione di questo iperparametro non poteva essere lasciata a un'impostazione manuale. Si è pertanto adottata una strategia di ottimizzazione quantitativa basata sul Silhouette Score. L'analisi della Silhouette è un metodo ampiamente riconosciuto per l'interpretazione e la validazione della coerenza all'interno dei cluster. Per ogni singola tabella, questa metrica calcola un punteggio compreso tra -1 e +1, valutando quanto l'elemento sia vicino ai punti della propria comunità (coesione intra-cluster) e, contestualmente, quanto sia distante dai punti della comunità adiacente più prossima (separazione inter-cluster). Un valore medio elevato indica che i cluster sono densi e ben separati nello spazio latente semantico. Operativamente, la ricerca del parametro ottimale è stata condotta eseguendo l'algoritmo in modo iterativo su un intervallo predefinito di possibili valori di K. Per ciascuna partizione generata in questa fase esplorativa, è stato calcolato il Silhouette Score medio su tutti i nodi.

Il sistema ha quindi identificato in modo del tutto *data-driven* la configurazione in grado di massimizzare tale metrica globale, eleggendo il relativo K come numero di cluster ideale per il database in esame.Sulla base di questo parametro ottimizzato, l'algoritmo K-Means è stato eseguito un'ultima volta per generare l'assegnamento definitivo. Il partizionamento risultante fornisce così una classificazione "pura", basata esclusivamente su ciò che le tabelle "significano" e non su come esse sono collegate. I risultati ottenuti tramite questa *baseline* semantica, unitamente a quelli derivanti dalle euristiche puramente topologiche (come l'algoritmo di Leiden), costituiranno i pilastri di confronto necessari per valutare in modo critico le architetture ibride nel prosieguo del capitolo.

6.3 Analisi dei risultati

Conclusa la disamina del *setup* sperimentale, questa sezione costituisce il fulcro valutativo dell'intero lavoro di tesi. Verrà presentata un'analisi critica e comparativa dei risultati ottenuti dai diversi modelli di clustering (Leiden, K-Means semantico, DMON e DGCluster) applicati ai tre dataset oggetto di studio. L'esposizione seguirà un ordine di complessità topologica crescente: si partirà da un ambiente di validazione controllato e di ridotte dimensioni, per passare a un tipico gestionale aziendale strutturato, fino ad arrivare alla frammentazione di uno schema industriale reale e stratificato. Per ciascun caso di studio verranno analizzati i punteggi ottenuti nelle metriche topologiche, semantiche e ibride, al fine di determinare quale architettura riesca a isolare con maggiore accuratezza i domini funzionali del database.

Prima di presentare i risultati definitivi relativi all'estrazione dei cluster, occorre precisare che le architetture GNN sono state sottoposte alla procedura di *Grid Search* sopra descritta per individuare la configurazione parametrica ottimale. Le

impostazioni di addestramento, applicate in modo uniforme a tutti i database per i due paradigmi di *text embedding*, sono le seguenti:

- **Modello GNN con Full Description**: Per la rete addestrata sugli embedding monolitici (senza partizionamento del testo), il processo di ottimizzazione ha fissato la durata del training a `EPOCHS = 200`, con un *learning rate* (`LR`) pari a 1×10^{-3}. La capacità rappresentativa della rete è stata dimensionata impostando sia i canali dei layer nascosti (`HIDDEN_CHANNELS`) sia la dimensione dello spazio latente finale (`EMBED_DIM`) a 128. Per l'architettura DMON, la ricerca del numero ideale di cluster (K) ha esplorato l'intervallo compreso tra `START = 2` ed `END = 20`.

- **Modello GNN con Elaborazione Batch (Pooling)**: Per l'architettura incaricata di elaborare l'informazione suddivisa in *chunk*, sono stati mantenuti invariati i parametri di base per garantire la comparabilità dei risultati (`EPOCHS = 200`, `LR = 1e-3`, `HIDDEN_CHANNELS = 128`, `EMBED_DIM = 128`, range di esplorazione $K \in [2, 20]$). Tuttavia, a questa configurazione si aggiunge il parametro `POOL_HIDDEN = 128`, necessario per dimensionare il layer di *attention* interno che si occupa di aggregare i vettori multipli di ogni singolo nodo prima del passaggio ai layer convoluzionali.

6.3.1 Risultati sul dataset Aviano DB

Il primo scenario di validazione è stato condotto sul database "Aviano", un sistema gestionale per il noleggio di veicoli di dimensioni estremamente contenute (13 tabelle e 15 vincoli relazionali). Nonostante la sua semplicità, questo dataset rappresenta un banco di prova fondamentale per verificare che le architetture neurali non incorrano in fenomeni di *oversmoothing* o collasso, riuscendo a mantenere separati i tre domini logici principali (Inventario, Transazionale, Anagrafica) anche su una rete molto piccola.

Per presentare in modo organico i risultati dell'esperimento, tutte le metriche (topologiche, semantiche e ibride) calcolate per i cinque modelli testati sono state aggregate nella Tabella 6.1.

Metodo	K	Metriche Topologiche			Metriche Semantiche		Metriche Ibride	
		Mod. (Q)	*Coverage*	*Cond.* (Φ) ↓	*Sim. Diff*	*Sim. Ratio*	*Sem. Mod.*	*Attr. Mod.*
`kmeans_single_desc`	5	0.2622	0.5333	0.2067	0.2493	1.5127	0.3184	0.2460
`leiden_communities`	4	**0.3511**	0.6667	0.2121	0.1747	1.3480	0.3483	0.2176
`gnn_full_desc`	8	0.1956	0.3333	0.2354	**0.2768**	**1.5502**	0.2180	0.2414
`gnn_pool_desc`	4	0.3467	0.6000	0.2167	0.1723	1.3486	**0.3605**	**0.2556**
`gnn_dgcluster`	**3**	0.3422	**0.7333**	**0.1603**	0.1343	1.2659	0.3418	0.2051

Tabella 6.1: Risultati della valutazione sul dataset Aviano DB. Il simbolo ↓ indica che per la Conduttanza media valori minori rappresentano prestazioni migliori. I valori in grassetto evidenziano i risultati ottimali per ciascuna metrica.

Le *Baseline*: Topologia vs Semantica

Dall'analisi dei risultati delle *baseline* emerge con nettezza la contrapposizione tra struttura e significato. L'algoritmo di Leiden (`leiden_communities`), fondandosi esclusivamente sulla topologia del grafo, individua 4 cluster e raggiunge, come prevedibile, il valore più alto di modularità standard ($Q = 0.3511$). Al contrario, il K-Means (`kmeans_single_desc`), che lavora unicamente sui vettori semantici senza considerare gli archi, produce 5 cluster e ottiene un'eccellente separazione testuale (Semantic Ratio pari a 1.51), ma non riesce a rispettare adeguatamente i vincoli del database (la modularità cala a 0.26 e il Coverage si attesta al 53%). Questi risultati confermano l'ipotesi iniziale: nessuno dei due approcci, se utilizzato in modo isolato, è sufficiente per ottenere una corretta modularizzazione dello schema.

Il comportamento di DMON e l'impatto del *Chunking*

Il dato più interessante dell'esperimento su Aviano riguarda l'impatto della codifica testuale sulle reti GNN. Il modello DMON addestrato sugli embedding monolitici (`gnn_full_desc`) manifesta un evidente fenomeno di *over-fragmentation*. Il modello divide il piccolo grafo in ben 8 cluster: questo produce punteggi semantici altissimi (Sim Ratio di 1.55, il più alto in assoluto) perché le tabelle vengono quasi isolate, ma distrugge letteralmente la topologia. La modularità crolla a 0.19 e il Coverage si attesta a un misero 33%, indicando che quasi tutti i vincoli di *Foreign Key* sono stati tagliati. La situazione si ribalta drasticamente introducendo la strategia a *batch* con livello di *attention* (`gnn_pool_desc`). Fornendo alla rete l'informazione frazionata in *chunk*, DMON stabilizza il proprio spazio latente, selezionando il 4 come numero di cluster ideale.

Topologicamente, il modello si allinea quasi perfettamente all'euristica di Leiden ($Q = 0.3467$), ma il vero successo si osserva nelle **metriche ibride**: questo modello registra il picco massimo sia nella *Semantic-Topological Modularity* (0.3605)

che nell'*Attributed Modularity* (0.2556). Ciò dimostra che la rete ha trovato il compromesso matematico perfetto, aggregando le tabelle che sono simultaneamente vicine nel grafo e semanticamente coerenti.

Le performance di DGCluster

Infine, l'architettura DGCluster si distingue per la sua robustezza e coesione spaziale. Sfruttando la soglia dell'algoritmo BIRCH e senza alcun bisogno di forzare un parametro K prestabilito, DGCluster converge autonomamente sull'individuazione di **3 macro-comunità**. Questa tripartizione rispecchia perfettamente i tre domini concettuali attesi analizzati nella descrizione del dataset (Inventario, Transazionale, Anagrafica). Dal punto di vista topologico, DGCluster eccelle nella qualità del partizionamento: detiene il valore più alto di *Coverage* (73.33% degli archi totali assorbiti internamente) e la *Mean Conductance* più bassa in assoluto (0.1603), segno inequivocabile di confini netti e ben delimitati. Tuttavia, i punteggi semantici leggermente più bassi rispetto agli altri modelli (Sim Diff 0.13) non rappresentano un fallimento, bensì una conseguenza matematica della granularità. Avendo creato solo 3 grandi cluster, ciascun contenitore ingloba una maggiore varianza di vocaboli rispetto a un partizionamento in 8 micro-comunità.
In sintesi, i risultati su questo primo *toy dataset* confermano la validità dell'architettura neurale rispetto ai metodi tradizionali: DMON (nella variante *pooled*) dimostra la miglior capacità di bilanciamento ibrido, mentre DGCluster si afferma come l'approccio più affidabile per dedurre i reali macro-moduli funzionali operando tagli topologici ottimali.

6.3.2 Risultati sul dataset AdventureWorks DB

Il secondo scenario di test sposta l'analisi su un livello di complessità decisamente superiore. Il dataset "AdventureWorks", rappresentando un sistema gestionale ERP completo (71 tabelle e 104 vincoli relazionali), costituisce un banco di prova ideale per valutare la scalabilità dei modelli.

I risultati derivanti dall'applicazione dei modelli su questo dataset, esposti integralmente nella Tabella 6.2, permettono di misurare la capacità degli algoritmi di ricostruire questa compartimentazione latente.

Metodo	K	Metriche Topologiche			Metriche Semantiche		Metriche Ibride	
		Mod. (Q)	*Coverage*	*Cond.* (Φ) ↓	*Sim. Diff*	*Sim. Ratio*	*Sem. Mod.*	*Attr. Mod.*
`kmeans_single_desc`	9	0.3153	0.4556	0.3376	**0.1670**	**1.5751**	0.3839	0.2392
`leiden_communities`	8	**0.6158**	0.7667	0.1351	0.1414	1.4759	0.6132	**0.3574**
`gnn_full_desc`	9	0.5864	0.7111	0.1470	0.1222	1.4094	0.6018	0.3498
`gnn_pool_desc`	9	0.5948	0.7111	0.1439	0.1262	1.4209	0.6076	0.3538
`gnn_dgcluster`	**6**	0.5991	**0.8222**	**0.1086**	0.1364	1.4637	**0.6156**	0.3555

Tabella 6.2: Risultati della valutazione sul dataset AdventureWorks DB. Il simbolo ↓ indica che per la Conduttanza media valori minori rappresentano prestazioni migliori. I valori in grassetto evidenziano i risultati ottimali per ciascuna metrica.

Le *Baseline*: I limiti dell'approccio semantico isolato

L'aumento dimensionale del grafo rende ancora più evidenti le criticità dei metodi non ibridi. Il partizionamento puramente semantico di K-Means (`kmeans_single_desc`), pur registrando le metriche semantiche più elevate (Sim Ratio di 1.57), collassa completamente sotto il profilo strutturale.
Con un Coverage del 45.56% e una Modularità di appena 0.3153, questo approccio smembra letteralmente il database, separando tabelle fortemente connesse solo perché il loro vocabolario descrittivo risulta differente.
Di contro, l'euristica topologica di Leiden si dimostra eccellente nell'ottimizzare la struttura, registrando il picco assoluto di Modularità standard ($Q = 0.6158$) e un ottimo valore di Attributed Modularity (0.3574).
Tuttavia, tendendo a massimizzare la densità interna, Leiden frammenta la rete in 8 cluster, discostandosi dai 6 moduli architetturali reali del database.

La stabilità di DMON e il confronto tra le codifiche

Su questo grafo di media complessità, le due varianti dell'architettura DMON mostrano un comportamento coerente e stabile. Entrambe le strategie di embedding convergono (tramite Grid Search guidata dall'Attributed Modularity) su un partizionamento in 9 cluster.
Confrontando le due codifiche testuali, si nota come l'introduzione del *chunking* (`gnn_pool_desc`) garantisca ancora una volta un lieve ma misurabile vantaggio rispetto alla descrizione monolitica (`gnn_full_desc`) su quasi tutti i fronti: la Modularità passa da 0.586 a 0.594, la Conduttanza diminuisce e le metriche ibride registrano un miglioramento.
Sebbene l'*over-fragmentation* osservato su Aviano con l'approccio *Full Description* non si ripeta qui con la stessa gravità, l'elaborazione a *batch* si conferma la strategia più solida per le reti GNN basate su loss spettriali.

L'analisi qualitativa di DGCluster e l'allineamento con i moduli nativi

Il risultato quantitativamente più rilevante dell'intero esperimento su AdventureWorks è fornito dall'architettura DGCluster. Senza ricevere in input alcun parametro prefissato, il modello neurale, supportato dal clustering gerarchico BIRCH, identifica autonomamente **esattamente 6 cluster**. Dal punto di vista puramente numerico, questo dato è sorprendente, in quanto coincide perfettamente con il numero di *schema* nativi progettati dai creatori del database.

Tuttavia, un'ispezione qualitativa e visiva del partizionamento generato, illustrato nel grafo in Figura 6.1, permette di apprezzare alcune sfumature fondamentali sul reale comportamento della rete.
Dall'osservazione topologica emerge che il modello eccelle nell'individuare e isolare con estrema precisione le macro-aree funzionali nevralgiche dell'azienda. I moduli **Person** (anagrafiche), **Sales** (ciclo attivo), **Production** (inventario e manifattura) e **HumanResources** (personale) vengono aggregati in comunità dense e semanticamente coerenti. Di contro, la rete mostra alcune imprecisioni nella delimitazione dei due domini rimanenti: **Purchasing** (ciclo passivo) e **dbo** (tabelle di log e sistema).

Questa parziale "confusione" algoritmica è tuttavia giustificabile sia a livello strutturale che semantico:

- Il modulo *Purchasing* possiede fortissime dipendenze topologiche verso *Production* (es. il legame tra fornitori e prodotti) e verso le anagrafiche, portando la rete a sfumare i confini di questo dominio e ad assorbirne alcune tabelle nei cluster adiacenti.

- Il modulo *dbo*, contenendo tabelle di servizio (`ErrorLog`, `DatabaseLog`), possiede una semantica generalista e una connettività spesso anomala o isolata, rendendone complessa la classificazione come dominio di business a sé stante da parte di una GNN.

Nonostante queste lievi "sbavature" nell'allineamento semantico 1:1, le metriche globali certificano la superiorità della partizione estratta da DGCluster. Il modello domina incontrastato nella delimitazione strutturale delle comunità: incapsula al proprio interno l'82.22% dei vincoli di *Foreign Key* (il *Coverage* più alto in assoluto) e abbatte la *Mean Conductance* a 0.1086, dimostrando che i 6 moduli individuati, seppur con lievi differenze rispetto al *design* originale umano, risultano matematicamente molto più coesi e indipendenti. Anche il primato nella *Semantic-Topological Modularity* (0.6156) conferma che la fusione *early-stage* tra rappresentazione testuale e connettività implementata da DGCluster costituisce l'approccio più solido per gestire ecosistemi ERP complessi.

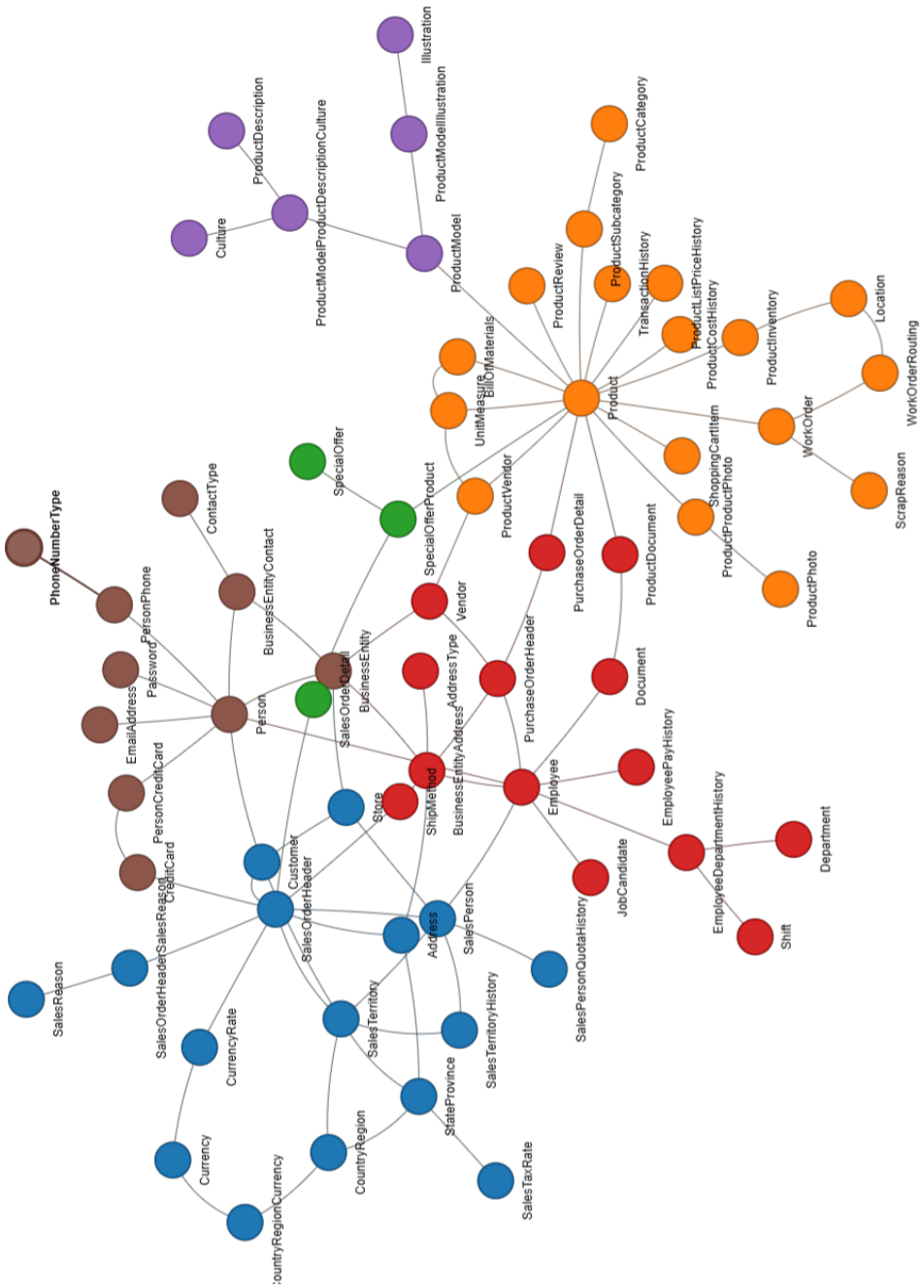

Figura 6.1: Visualizzazione del clustering prodotto da DGCluster sul dataset AdventureWorks. I colori dei nodi rappresentano le partizioni individuate dall'algoritmo. Sebbene i macro-domini principali siano ben isolati, si nota una sovrapposizione nelle aree relative agli acquisti (*Purchasing*) e alle tabelle di sistema (*dbo*).

6.3.3 Risultati sul dataset industriale

Il terzo e ultimo scenario sperimentale affronta la sfida più complessa e realistica del progetto: l'applicazione dei modelli di clustering su un database aziendale autentico e in produzione, privo di una documentazione architetturale esplicita.
Come descritto nella Sezione 6.1, questo grafo da 116 tabelle e 128 vincoli relazionali modella il dominio stratificato dei sinistri stradali (polizze, pagamenti, spese legali, anagrafiche, ecc.).
Non essendoci una *ground truth* predefinita (nessuna divisione in moduli nota a priori), la validazione si affida unicamente all'oggettività delle metriche matematiche per stabilire quale algoritmo abbia estratto i confini funzionali più plausibili e coerenti.

I risultati dell'esperimento su questo caso di studio industriale sono consolidati nella Tabella 6.3.

Metodo	**K**	**Metriche Topologiche**			**Metriche Semantiche**		**Metriche Ibride**	
		Mod. (Q)	*Coverage*	*Cond.* (Φ) ↓	*Sim. Diff*	*Sim. Ratio*	*Sem. Mod.*	*Attr. Mod.*
`kmeans_single_desc`	9	0.2654	0.3906	0.2789	**0.1652**	**1.4542**	0.3473	0.1870
`leiden_communities`	11	**0.7769**	**0.8984**	0.0671	0.1296	1.3501	**0.7985**	0.4251
`gnn_full_desc`	12	0.7413	0.8438	0.0961	0.1209	1.3272	0.7622	0.4192
`gnn_pool_desc`	15	0.7433	0.8281	0.0908	0.1223	1.3296	0.7760	0.4211
`gnn_dgcluster`	10	0.7695	0.8906	**0.0664**	0.1134	1.3063	0.7882	**0.4289**

Tabella 6.3: Risultati della valutazione sul dataset industriale. Il simbolo ↓ indica che per la Conduttanza media valori minori rappresentano prestazioni migliori. I valori in grassetto evidenziano i risultati ottimali per ciascuna metrica calcolata sul database reale.

L'alta correlazione strutturale nei database *legacy*

L'analisi delle *baseline* su questo schema reale porta alla luce una dinamica tipica dei sistemi informativi storicizzati (*legacy*).
A differenza dei database di test, qui l'euristica di Leiden ottiene risultati eccezionali non solo nella topologia (Modularità record di 0.7769), ma anche nelle metriche ibride e semantiche (*Semantic Modularity* pari a 0.7985). Questo fenomeno si spiega con il fatto che, in un gestionale industriale stratificato negli anni, le tabelle fortemente connesse tra loro da vincoli di *Foreign Key* tendono fisiologicamente a descrivere lo stesso processo di business.
Al contrario, il K-Means semantico isolato fallisce in modo drastico: pur ottenendo il miglior distacco vettoriale, la sua Modularità crolla a 0.2654 con un Coverage del 39%, confermando l'inadeguatezza degli approcci puramente testuali quando si tratta di fare *reverse engineering* su architetture complesse.

Il comportamento di DMON

Applicando i modelli di *Deep Graph Clustering*, si osserva come l'architettura DMON continui a offrire partizioni di alto livello, sebbene con una tendenza a una maggiore frammentazione. La variante addestrata sulle descrizioni intere (`gnn_full_desc`) individua 12 cluster, mentre l'approccio con *chunking* (`gnn_pool_desc`) ne estrae ben 15. Anche su questo dataset reale si conferma il *trend* osservato in precedenza: la strategia a *batch* con layer di aggregazione performa costantemente meglio rispetto al documento monolitico. Il modello `pool_desc` abbassa la Conduttanza, alza la Modularità e migliora il punteggio di Attributed Modularity (0.4211 contro 0.4192). Tuttavia, la creazione di 15 cluster per un grafo di 116 nodi potrebbe risultare eccessivamente granulare per un architetto software alla ricerca di macro-moduli applicativi.

DGCluster: il bilanciamento definitivo per l'esplorazione del dominio

In questo scenario esplorativo senza verità nota, DGCluster emerge ancora una volta come la soluzione algoritmica più solida e bilanciata in assoluto.
Individuando autonomamente **10 cluster**, il modello propone una ripartizione architetturale del business estremamente plausibile e gestibile (ad esempio: un modulo per i pagamenti, uno per le entità legali, uno per le agenzie sul territorio, ecc.). Ciò che decreta la superiorità di DGCluster in questo esperimento sono due metriche cruciali:

1. **Il primato nella separazione comunitaria**. DGCluster registra la *Mean Conductance* più bassa di tutti i modelli testati (**0.0664**). Questo significa che i tagli operati nel grafo recidono pochissimi legami esterni rispetto alla densità interna. Le 10 comunità trovate sono quindi moduli software isolati, ideali per essere potenzialmente migrati verso architetture a microservizi con il minimo rischio di dipendenze incrociate.

2. **Il picco nell'Attributed Modularity**. Mentre Leiden ottiene la modularità topologica pura più alta, DGCluster vince nella metrica ibrida definitiva (*Attributed Modularity* pari a **0.4289**). Questo dato dimostra matematicamente che la GNN non si è limitata a seguire ciecamente la struttura degli archi, ma ha corretto e perfezionato la partizione guidata dagli embedding generati da AWS Titan, raggiungendo il miglior compromesso globale possibile tra topologia relazionale e affinità semantica del dominio assicurativo.

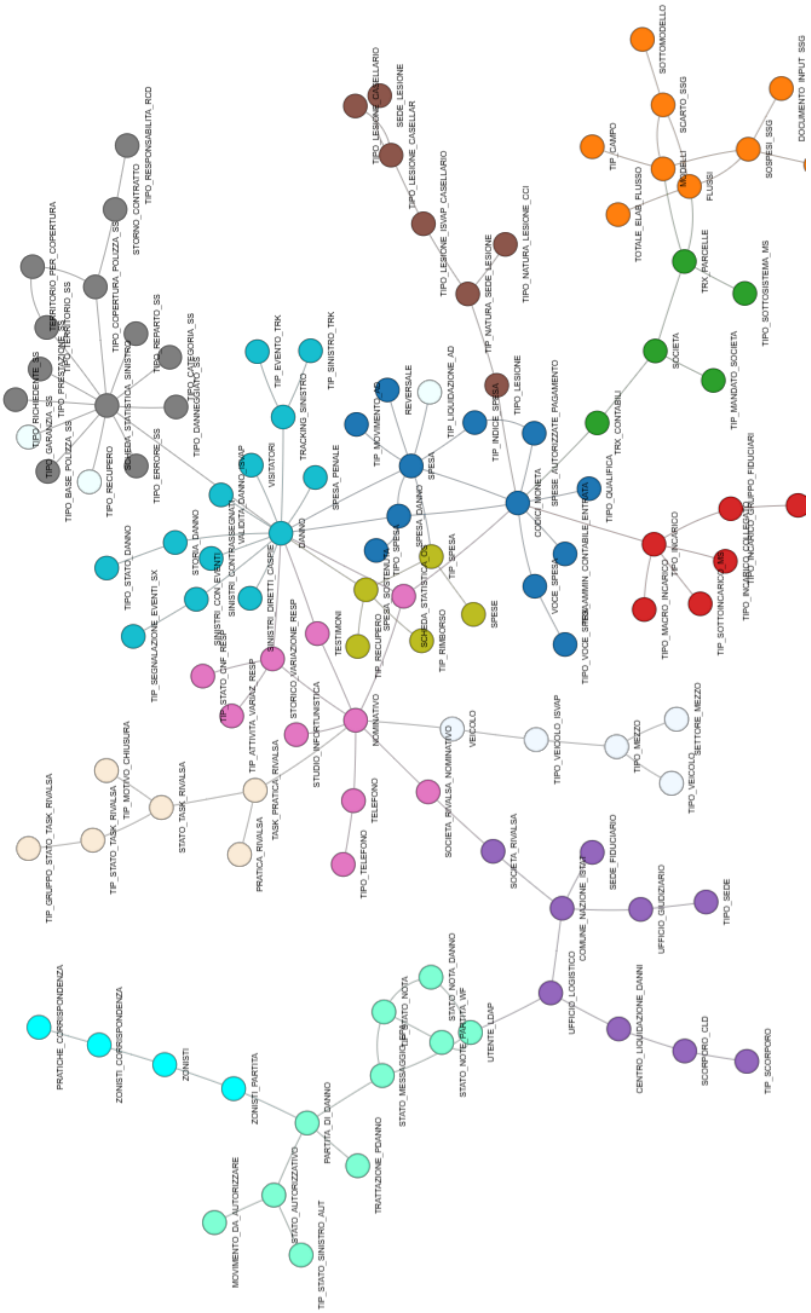

Figura 6.2: Visualizzazione del clustering prodotto dal modello DMON (variante pooled) sul dataset industriale. La frammentazione in 15 cluster produce un'eccessiva segmentazione del dominio, dividendo visivamente aree ad alta densità relazionale.

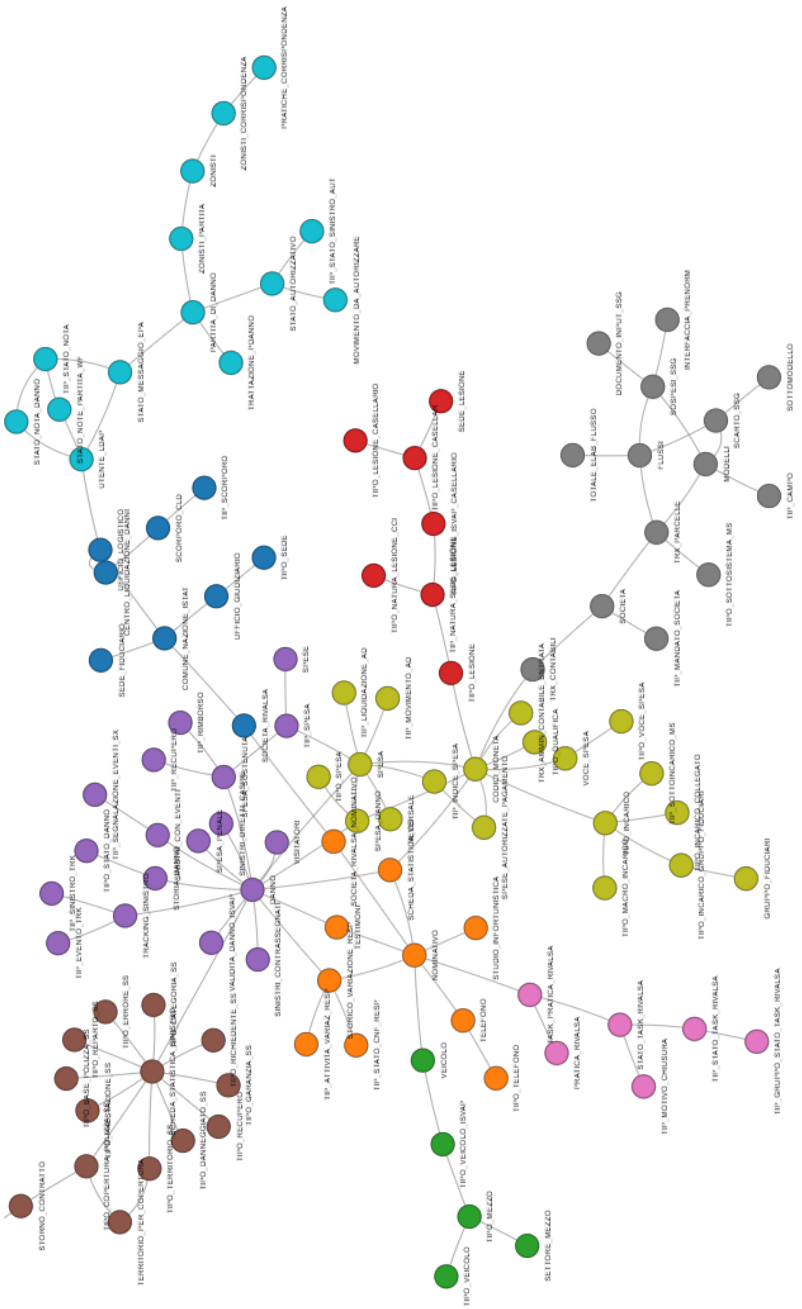

Figura 6.3: Visualizzazione del clustering prodotto dal modello DGCluster sul dataset industriale. I 10 cluster individuati risultano topologicamente compatti e semanticamente omogenei, definendo macro-moduli di business chiari e ben isolati.

Analisi visiva delle partizioni: DMON vs DGCluster

A supporto dell'evidenza puramente numerica, un'analisi visiva dei grafi partizionati permette di apprezzare concretamente la differenza qualitativa tra le due architetture neurali più performanti.

Come illustrato nella Figura 6.2, il grafo generato dall'algoritmo DMON (`gnn_pool_desc`) evidenzia i limiti operativi legati all'impostazione rigida del parametro K.
Nonostante le buone metriche ibride, la suddivisione in 15 cluster produce un evidente fenomeno di *over-fragmentation*. Visivamente, si nota come diverse aree del grafo, pur essendo caratterizzate da un'alta densità di archi interni (segno di una forte dipendenza funzionale), vengano spezzate in micro-comunità distinte. Questo comportamento risulta poco utile in uno scenario di *software engineering*, in quanto isola eccessivamente le tabelle, generando moduli troppo piccoli per rappresentare un reale dominio di business indipendente.

Al contrario, l'osservazione della Figura 6.3 conferma la superiorità qualitativa dell'architettura DGCluster. L'estrazione autonoma di 10 macro-comunità si traduce in un grafo i cui cluster risultano nettamente più omogenei e coesi. I confini tra le diverse aree tematiche (ad esempio gestione legale, anagrafiche, contabilità dei sinistri) sono delineati in modo molto più pulito. Questa pulizia visiva è la diretta conseguenza matematica del valore record di *Mean Conductance* (0.0664) registrato dal modello: essendoci pochissimi archi che "tagliano" partizioni diverse, le comunità cromatiche appaiono dense al loro interno e debolmente connesse all'esterno, rispecchiando la struttura ideale di un'architettura a microservizi ben progettata.

In conclusione, i risultati estratti da quest'ultimo database certificano come l'integrazione tra Large Language Models, architetture Graph Neural Networks (nello specifico DGCluster) e metriche ibride rappresenti una soluzione all'avanguardia per l'ingegneria del software, capace di rivelare i confini applicativi latenti anche all'interno dei sistemi informativi aziendali più opachi e complessi.

Capitolo 7

Conclusioni

Il presente lavoro di tesi si è posto l'obiettivo di affrontare una delle sfide più complesse nell'ambito dell'ingegneria del software moderna: la comprensione, la modularizzazione e il *reverse engineering* di database relazionali *legacy*. Attraverso la piattaforma Dataslice AI, è stata progettata e validata una pipeline innovativa in grado di fondere la potenza generativa dei Large Language Models (LLM) con le avanzate capacità di *pattern recognition* strutturale offerte dalle Graph Neural Networks (GNN).

L'ampia campagna sperimentale, condotta su scenari di complessità crescente fino ad arrivare a un caso industriale reale di oltre cento tabelle, ha permesso di trarre conclusioni significative in merito all'efficacia delle diverse strategie di *clustering* su grafi attribuiti.

Da un lato, l'analisi ha evidenziato il valore e la solidità delle euristiche tradizionali puramente topologiche. L'algoritmo di Leiden, in particolare, ha dimostrato prestazioni sorprendenti: pur essendo un metodo matematicamente semplice e privo della capacità di "leggere" nativamente il significato delle tabelle, ha registrato punteggi di modularità standard estremamente elevati in tutti gli scenari. Il suo principale vantaggio risiede nell'efficienza: Leiden non necessita di addestramento, non richiede costose chiamate API per il calcolo degli *embedding* (se si esclude la fase di valutazione ibrida) e presenta un *overhead* computazionale e costi di implementazione minimi. Tuttavia, affidarsi unicamente alla topologia ha mostrato limiti evidenti sul piano pratico. Sebbene matematicamente valide, le partizioni generate da Leiden sono risultate, da un punto di vista visivo e architetturale, meno "precise" e meno coerenti. L'algoritmo tende infatti a inglobare tabelle con funzioni di business completamente diverse solo perché connesse da un percorso strutturale, producendo cluster ottimali per la teoria dei grafi, ma spesso inadeguati per un architetto software alla ricerca di domini logici per la migrazione a microservizi.

Dall'altro lato, i risultati empirici confermano come i metodi neurali basati sul *Deep Graph Clustering* (in particolar modo DMON e DGCluster) rappresentino l'approccio qualitativamente superiore e definitivo per questo genere di task. Nonostante i maggiori costi computazionali legati all'estrazione degli *embedding* semantici e alle tempistiche di addestramento, le GNN hanno dimostrato una capacità unica: quella di apprendere un bilanciamento dinamico tra la topologia relazionale e il significato intrinseco dei dati. Da una parte, architetture come DGCluster, operando sull'intero contesto semantico delle tabelle, sono riuscite a estrarre in modo del tutto autonomo i moduli logici nativi degli schemi (es. i 6 *schema* di AdventureWorks) e a isolare comunità altamente omogenee e coese nel dataset assicurativo. Dall'altra, l'impiego di strategie di frammentazione del testo (*text chunking*), convalidate con successo nel modello DMON *pooled*, ha dimostrato come un'elaborazione più granulare dell'informazione possa stabilizzare lo spazio latente, offrendo un'alternativa robusta per la gestione di descrizioni molto estese.

In sintesi, pur riconoscendo a metodi come Leiden il ruolo di eccellenti e rapide *baseline* strutturali, si ritiene che l'investimento nell'implementazione di Graph Neural Networks sia ampiamente giustificato dai risultati. Le metriche ibride introdotte (*Attributed Modularity*) e l'ispezione visiva dei grafi certificano che le GNN offrono un livello di comprensione olistica del database irraggiungibile dai metodi classici, restituendo partizioni che rispecchiano fedelmente il reale dominio applicativo e fornendo uno strumento di valore inestimabile per la modernizzazione dei sistemi informativi complessi.

Bibliografia

[1] Anthony Cleve e Jean-Luc Hainaut. «Co-transformations in Database Applications Evolution». In: a cura di Ralf Lämmel, João Saraiva e Joost Visser. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. ISBN: 978-3-540-46235-4. DOI: 10.1007/11877028_17. URL: https://doi.org/10.1007/11877028_17 (cit. a p. 2).

[2] Alon Halevy, Anand Rajaraman e Joann Ordille. «Data integration: the teenage years». In: *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment. 2006 (cit. a p. 3).

[3] Michael Stonebraker e Ihab F. Ilyas. «Data Integration: The Current Status and the Way Forward». In: *IEEE Data Engineering Bulletin* 41.2 (2018) (cit. a p. 3).

[4] Réka Albert e Albert-László Barabási. «Statistical mechanics of complex networks». In: *Reviews of Modern Physics* 74.1 (gen. 2002). ISSN: 1539-0756. DOI: 10.1103/revmodphys.74.47. URL: http://dx.doi.org/10.1103/RevModPhys.74.47 (cit. a p. 10).

[5] M. E. J. Newman. «The Structure and Function of Complex Networks». In: *SIAM Review* 45.2 (gen. 2003). ISSN: 1095-7200. DOI: 10.1137/s00361445 0342480. URL: http://dx.doi.org/10.1137/S003614450342480 (cit. alle pp. 10, 14).

[6] Santo Fortunato. «Community detection in graphs». In: *Physics Reports* 486.3–5 (feb. 2010). ISSN: 0370-1573. DOI: 10.1016/j.physrep.2009.11.002. URL: http://dx.doi.org/10.1016/j.physrep.2009.11.002 (cit. alle pp. 11, 13, 15).

[7] Santo Fortunato e Claudio Castellano. *Community Structure in Graphs*. 2007. arXiv: 0712.2716 [physics.soc-ph]. URL: https://arxiv.org/abs/0712.2716 (cit. alle pp. 11, 17).

[8] Herbert A. Simon. «The Architecture of Complexity». In: *Proceedings of the American Philosophical Society* 106.6 (1962), pp. 467–482 (cit. a p. 11).

[9] M. E. J. Newman e M. Girvan. «Finding and evaluating community structure in networks». In: *Physical Review E* 69.2 (feb. 2004). ISSN: 1550-2376. DOI: 10.1103/physreve.69.026113. URL: http://dx.doi.org/10.1103/PhysRevE.69.026113 (cit. alle pp. 12, 14, 21).

[10] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang e Philip S Yu. «A comprehensive survey on graph neural networks». In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24 (cit. a p. 18).

[11] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte e Etienne Lefebvre. «Fast unfolding of communities in large networks». In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (ott. 2008), P10008. ISSN: 1742-5468. DOI: 10.1088/1742-5468/2008/10/p10008. URL: http://dx.doi.org/10.1088/1742-5468/2008/10/P10008 (cit. alle pp. 22, 23).

[12] V. A. Traag, L. Waltman e N. J. van Eck. «From Louvain to Leiden: guaranteeing well-connected communities». In: *Scientific Reports* 9.1 (mar. 2019). ISSN: 2045-2322. DOI: 10.1038/s41598-019-41695-z. URL: http://dx.doi.org/10.1038/s41598-019-41695-z (cit. alle pp. 24, 25, 27).

[13] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner e Gabriele Monfardini. «The Graph Neural Network Model». In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605 (cit. a p. 32).

[14] Thomas N. Kipf e Max Welling. «Semi-Supervised Classification with Graph Convolutional Networks». In: (2017). arXiv: 1609.02907 [cs.LG]. URL: https://arxiv.org/abs/1609.02907 (cit. alle pp. 33, 35).

[15] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals e George E. Dahl. «Neural Message Passing for Quantum Chemistry». In: (2017). arXiv: 1704.01212 [cs.LG]. URL: https://arxiv.org/abs/1704.01212 (cit. a p. 33).

[16] Anton Tsitsulin, John Palowitch, Bryan Perozzi e Emmanuel Müller. «Graph Clustering with Graph Neural Networks». In: (2023). arXiv: 2006.16904 [cs.LG]. URL: https://arxiv.org/abs/2006.16904 (cit. alle pp. 38, 40).

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser e Illia Polosukhin. «Attention is all you need». In: *Advances in neural information processing systems*. Vol. 30. 2017 (cit. a p. 45).

[18] Aritra Bhowmick, Mert Kosan, Zexi Huang, Ambuj Singh e Sourav Medya. «DGCLUSTER: A Neural Framework for Attributed Graph Clustering via Modularity Maximization». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2024 (cit. alle pp. 48, 52).

[19] Tian Zhang, Raghu Ramakrishnan e Miron Livny. «BIRCH: an efficient data clustering method for very large databases». In: *ACM SIGMOD Record* 25.2 (1996), pp. 103–114 (cit. a p. 50).

[20] Hursh Desai. *Aviano-db: Simple Car Rental System Database Design Project*. `https://github.com/hurshd0/aviano-db`. Consultato: Marzo 2026. 2019 (cit. a p. 63).

[21] Microsoft. *AdventureWorks Sample Databases*. `https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/adventure-works`. Consultato: Marzo 2026. 2026 (cit. a p. 64).