UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

DOCUMENTAZIONE PROGETTO DI INTELLIGENZA ARTIFICIALE

Data Augmentation di strutture di dialoghi Multi-Party con Generazione Vincolata e Validazione GNN

TEAM

Francesco Giorgione Matricola: 0522501741

Simona Lo Conte Matricola: 0522501786

Marta Napolillo Matricola: 0522501787

Anno Accademico 2024-2025

Abstract

Questo lavoro presenta lo sviluppo di una pipeline innovativa per la Data Augmentation di strutture di dialoghi multiparte, combinando l'uso di Large Language Models (LLM) e Graph Neural Networks (GNN). L'approccio proposto si articola in diverse fasi: un'analisi approfondita dei dataset di riferimento (STAC, Minecraft e Molweni), una generazione vincolata di dialoghi sintetici tramite LLM e una validazione strutturale mediante GNN, con focus su due architetture principali, Graph Attention Network (GAT) e GraphSAGE. La coerenza delle relazioni tra EDU viene validata attraverso un task di Link Prediction, selezionando i contenuti più adatti per l'ampliamento dei dataset. La pipeline, eseguita sul dataset di Minecraft, ha prodotto risultati che mostrano un significativo miglioramento della capacità di generalizzazione dei modelli di parsing dei dialoghi, valutata tramite l'applicazione del modello Seq2Seq-DDP sul dataset originale e sui dataset aumentati. Questo framework fornisce un approccio scalabile e generalizzabile per l'analisi di dialoghi complessi, suggerendo ulteriori sviluppi futuri, tra cui l'ottimizzazione di parametri della pipeline e l'estensione di quest'ultima a dataset più ampi e diversificati.

Indice

CAPITOLO 1

Introduzione

Questo progetto si propone di sviluppare una pipeline per la *Data Augmentation* di strutture di dialoghi multi-parte, sfruttando modelli di tipo *Large Language Model (LLM)*. L'obiettivo principale è generare dialoghi sintetici che rispettino rigorosamente i vincoli strutturali e semantici definiti dai corpora di:

- **Stac**: è un corpus di conversazioni di chat strategiche annotate manualmente con informazioni relative alla negoziazione relative a 45 giochi che coinvolgono tre o più giocatori. Ogni gioco è suddiviso in dialoghi, o sottosezioni del gioco che comprendono una o più sessioni di contrattazione.
- **Minecraft Structured Dialogue Corpus (MSDC)**: è un dataset creato per analizzare le interazioni linguistiche tra giocatori in scenari collaborativi all'interno di Minecraft. Sono registrate le azioni eseguite dai giocatori, come la costruzione di strutture o la modifica dell'ambiente, per correlare le istruzioni verbali con le azioni fisiche nel gioco.
- **Molweni**: è un dataset che contiene su conversazioni che coinvolgono più partecipanti, riflettendo scenari di interazione complessi e di vario genere. La media degli interlocutori per dialogo è di 3,52.

I dataset analizzati contengono una collezione di grafi, più precisamente DAG (Directed Acyclic Graphs), utilizzati per rappresentare i dialoghi. In particolare:

- I nodi corrispondono alle unità del dialogo (EDU, Elementary Discourse Units) e includono informazioni come lo speaker e il contenuto testuale della frase.
- Gli archi descrivono le relazioni tra le diverse unità del dialogo, come risposte dirette o connessioni tematiche tra le frasi.

Per garantire la coerenza delle relazioni strutturali nei dialoghi sintetici generati, si utilizza una Graph Neural Network (GNN) per il task di Link Prediction. Nello specifico, la GNN, presi in input i DAG, li analizza per produrre node embedding arricchiti delle informazioni di contesto, i.e. caratteristiche di nodi ed archi, proprietà strutturali. Gli embedding prodotti vengono utilizzati per eseguire il task di link prediction, in modo da verificare che le strutture generate rispettino i vincoli logici e le regole presenti nei dati originali. Dopo la fase di validazione, si ottiene il dataset aumentato. Infine, il training di *Dialogue Discourse Parsing as Generation* (vedi repository GitHub cliccando qui) viene eseguito sul dataset di partenza e su quelli aumentati, in modo da confrontare i risultati ottenuti e verificare se l'augmentation ha determinato un miglioramento delle performance.

1.1 Struttura del documento

Il documento è articolato in diversi capitoli, ognuno dei quali approfondisce aspetti specifici del lavoro svolto. In particolare:

- Il Capitolo 2 fornisce una panoramica dei principali lavori già presenti in letteratura, trattanti la tematica affrontata in tale studio;
- Il Capitolo 3 descrive il modello di Data Augmentation adottato e l'implementazione della Graph Neural Network (GNN) utilizzata per effettuare il task di Link Prediction. Viene inoltre illustrata l'intera pipeline sviluppata nell'ambito del progetto. La fase finale riguarda poi la valutazione del dataset aumentato di generalizzare a nuovi modelli;

- Il Capitolo 4 presenta un'analisi approfondita dei risultati ottenuti attraverso la validazione del modello GNN e del relativo Link Predictor, evidenziandone le performance e i limiti. Inoltre vengono analizzati i risultati ottenuti mediante l'esecuzione del modello Seq2Se2-DDP sui dataset originale e aumentato;
- Il Capitolo 5 riassume le principali conclusioni emerse dal lavoro, proponendo al contempo idee per futuri sviluppi e possibili miglioramenti.

CAPITOLO 2

Stato dell'arte

2.1 Data Augmentation

La **Data Augmentation** è una tecnica utilizzata per aumentare la diversità di un dataset senza dover raccogliere e annotare nuovi dati. Nello specifico, quello che viene fatto è applicare varie trasformazioni ai dati esistenti per creare nuove versioni modificate, aiutando il modello a imparare a generalizzare meglio. Queste trasformazioni vengono aggiunte al dataset iniziale, preservando l'etichetta originale dei dati e garantendo quindi che i dati aumentati rimangano rilevanti e utili per l'addestramento. Quindi, la Data Augmentation è un'area cruciale per migliorare la qualità e la varietà dei dati utilizzati nel training dei modelli, favorendo la capacità di generalizzare su nuovi dati e riducendo il rischio di overfitting.

In merito alla tematica di Data Augmentation, sono stati analizzati alcuni dei lavori pre-esistenti in letteratura, al fine di comprendere al meglio possibili sfide e miglioramenti da apportare.

Nello studio condotto da Chen et al. [1] viene introdotto un framework per il miglioramento della generazione conversazionale utilizzando un modello LLM (ConvAug).

La metodologia condotta parte dalla generazione di conversazioni multi-livello, catturando la diversa natura dei contesti conversazionali. Questo viene fatto attraverso modelli LLM, definendo un prompt ispirato alla cognizione umana che limiti la generazione di falsi positivi, falsi negativi e allucinazioni. Inoltre, viene sviluppato un filtro di campioni adatti alla difficoltà che seleziona campioni impegnativi per conversazioni complesse, dando così al modello uno spazio di apprendimento più ampio. Gli esperimenti vengono poi condotti considerando quattro dataset e ConvAug si è dimostrato, sia in scenari normali che zero-shot, notevolmente efficace e generalizzabile.

Il lavoro condotto da Cimino et al.[2] affronta la sfida data dalla scarsità di dati nell'analisi dei dialoghi, proponendo un metodo non supervisionato per l'estrazione di strutture di discorso in dialoghi multiparte basato sulla coerenza. Utilizzando LLM open-source sintonizzati su dati conversazionali, il metodo identifica sotto-dialoghi coerenti tramite due algoritmi: DS-DP che utilizza una strategia di programmazione dinamica e DS-FLOW che segue un approccio greedy. La fase di valutazione è stata condotta sul dataset STAC e su un subset di MOLWENI, ottenendo prestazioni migliori per il secondo corpus. In generale, però, le prestazioni ottenute per entrambi i dataset, misurate in termini di F1-score, hanno superato quelle dei precedenti metodi non supervisionati.

La maggior parte dei lavori e studi condotti mira ad analizzare e migliorare l'applicazione della Data Augmentation in vari contesti, con l'obiettivo di aumentare la qualità e la diversità dei dati utilizzati nei modelli di apprendimento. Tuttavia, la diversità del nostro studio risiede nell'introduzione di un approccio che integra la Data Augmentation di strutture di dialoghi multiparte con una fase di validazione basata su Graph Neural Networks (GNN), fornendo un framework completo ed efficace. Il nostro metodo utilizza tecniche di generazione vincolata per garantire coerenza e rilevanza nei dati aumentati, applicandolo a dataset complessi come STAC, Minecraft e Molweni, con l'obiettivo di catturare la ricchezza strutturale dei dialoghi multiparte.

La validazione dell'efficacia della Data Augmentation proposta avviene attraverso un task di link prediction, implementato mediante Graph Neural Networks (GNN). In particolare, sono state implementate e confrontate due architetture: Graph Attention Network (GAT), che utilizza meccanismi di attenzione per attribuire pesi differenti ai vicini di un nodo, e GraphSAGE, che sfrutta metodi di aggregazione efficienti per generare rappresentazioni dei nodi. Questo confronto approfondito consente di valutare quindi quale architettura sia più efficace nel catturare le relazioni strutturali emerse dai dati aumentati. Il risultato finale è l'implementazione di una pipeline automatizzata, in grado di gestire sia la fase di Data Augmentation che quella di Validazione tramite GNN, fornendo una soluzione scalabile e generalizzabile per l'analisi e il miglioramento di dialoghi complessi.

CAPITOLO 3

Metodologia di implementazione

La metodologia proposta si articola quindi in cinque fasi principali, descritte dettagliatamente in questo capitolo. Innanzitutto, viene effettuata un'analisi dei dataset (STAC, Minecraft e Molweni) per comprendere le caratteristiche delle strutture discorsive e le relazioni tra le unità di dialogo (EDUs). Questa fase ha lo scopo di identificare i pattern ricorrenti e criteri strutturali fondamentali da preservare durante la generazione sintetica. Successivamente, si passa alla generazione vincolata di dialoghi attraverso i modelli di Large Language Model (LLM). Dopo aver ottenuto le rappresentazioni in embeddings delle EDUs dei dialoghi considerati, la terza fase prevede la validazione dei dialoghi generati al fine di scegliere la migliore EDU da inserire nel dataset aumentato. Questo avviene tramite Graph Neural Networks (GNN) addestrata su dati annotati. Viene quindi sviluppata una pipeline di augmentation che integra i moduli di generazione e validazione in un processo automatizzato. La fase finale prevede la valutazione del dataset ottenuto a seguito della fase di Data Augmentation da parte di un LLM differente da quello utilizzato durante l'implementazione, in modo da valutare la capacità dei dati di generalizzare a nuovi modelli.

In merito alla metodologia condotta, le fasi di Analisi e Generazione degli Embeddings sono state condotte nel contempo sui dataset di STAC, Minecraft e Molweni. Un'altra fase condotta su tutti e tre i dataset è quella di implementazione delle GNN, in particolare GAT e GraphSAGE, per ottenere quindi un modello pre-addestrato su cui poter effettuare Link Prediction. A causa delle limitazioni sul numero di chiamate all'API imposte dal modello GPT-4o mini utilizzato per la fase di Data Augmentation, gli step descritti nelle Sezioni 3.3, 3.4.3, 3.5 e 3.6 sono condotti solo sul dataset di Minecraft. Questo è dovuto al fatto che Minecraft ha dimensioni inferiori in merito al numero di dialoghi contenuti rispetto agli altri dataset utilizzati in tale studio.

3.1 Analisi dei dataset

I dataset analizzati includono Stac, Minecraft e Molweni, ciascuno progettato per rappresentare i dialoghi reali attraverso i DAG (Directed Acyclic Graph). Ogni dataset è suddiviso in tre sezioni (Train, Validation e Test set) ciascuno di essi contenente una lista di DAG. La Tabella 3.1 contiene informazioni sui numeri di DAG presenti in tre dataset (STAC, MOLWENI e MINECRAFT) per le diverse suddivisioni dei dati: train, test e validation.

Nome Dataset	Elementi di Train	Elementi di Validation	Elementi di Test
STAC	947	103	109
MOLWENI	9000	500	500
MINECRAFT	307	132	133

Tabella 3.1: Distribuzione dei DAG nei dataset di Train, Test e Validation.

Un DAG (Directed Acyclic Graph) è un grafo diretto che non contiene cicli. Formalmente, un DAG è una coppia $G = (V, E)$ dove:

- V: rappresenta un insieme finito di nodi. Nei dataset considerati, questi nodi corrispondono alle EDU (Elementary Discourse Units) che contengono le informazioni rilevanti, come lo speaker e il contenuto testuale della frase associata.

- $E \subseteq VxV$: è un insieme di archi diretti, dove ogni arco è una coppia ordinata (u, v) con $u, v \in V$. Nei dataset, questi archi rappresentano le relazioni tra le EDU, che possono includere differenti tipologie, meglio descritte nella Sezione 3.1.1.

La rappresentazione dei dialoghi tramite DAG consente di modellare in modo strutturale e semantico le interazioni all'interno di dialoghi multi-parte. La Figura 3.1 rappresenta un esempio di DAG per il dataset STAC.

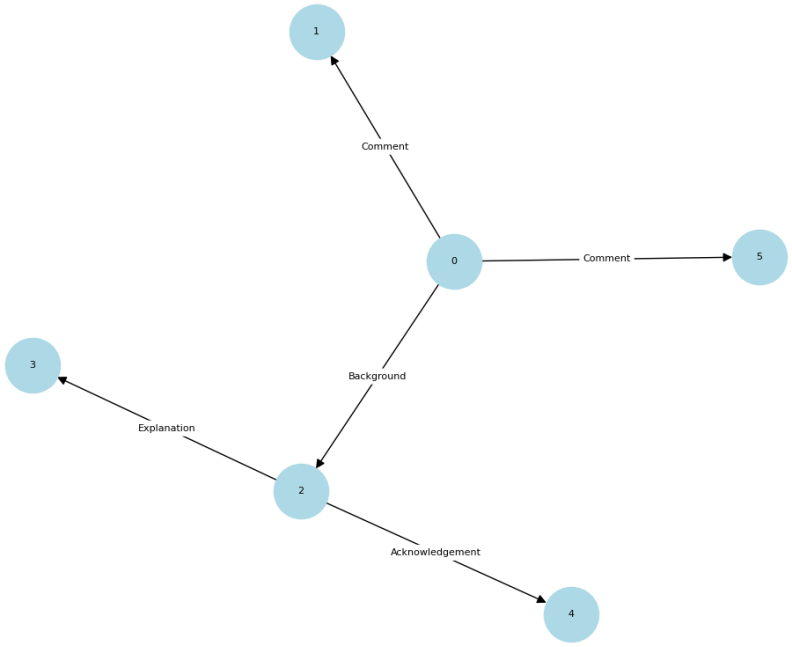


Figura 3.1: Esempio di DAG contenuto nel dataset di STAC

3.1.1 Analisi delle relazioni

I tre dataset considerati, oltre a contenere la lista di unità di dialogo componenti il discorso, contengono anche le relazioni tra le EDUs. Tali relazioni tra coppie di EDU possono essere di vario tipo, come riportato in [3]. Il seguente elenco riporta una breve definizione dei tipi di relazioni analizzate:

1. **Question_answer_pair**: Una EDU pone una domanda e l'altra fornisce una risposta;

2. **Comment**: Una EDU aggiunge un'opinione o una valutazione del contenuto dell'altra;
3. **Acknowledgement**: Una EDU riconosce o afferma le informazioni dell'altra, segnalata di solito da espressioni come OK, Right, Right then, Good, Fine;
4. **Continuation**: Entrambe le EDU elaborano o forniscono un contesto allo stesso segmento;
5. **Elaboration**: Una EDU fornisce ulteriori informazioni sull'eventualità introdotta nell'altra;
6. **Q_Elab**: Una EDU elabora la domanda posta dall'altra;
7. **Contrast**: Entrambe le EDU hanno strutture semantiche simili, ma presentano informazioni in contrasto tra loro;
8. **Explanation**: Una EDU fornisce una spiegazione o un ragionamento relativo all'altra;
9. **Clarification_question**: Una EDU chiede chiarimenti sulle informazioni contenute nell'altra;
10. **Result**: Una EDU descrive la causa dell'effetto descritto nell'altra;
11. **Correction**: Una EDU corregge o perfeziona le informazioni presentate nell'altra;
12. **Parallel**: Entrambe le EDU descrivono idee simili o parallele;
13. **Conditional**: Una EDU descrive una condizione necessaria affinché l'altra si verifichi;
14. **Alternation**: Una EDU presenta un'alternativa all'altra;
15. **Background**: Una EDU fornisce informazioni di base rilevanti per l'altra;
16. **Narration**: Una EDU descrive una sequenza di eventi o racconta una storia;

17. **Confirmation_question**: Una EDU cerca di confermare o convalidare le informazioni dell'altra;

18. **Sequence**: Una EDU descrive un evento che si verifica prima o dopo l'altra.

È da notare che tutti e tre i dataset contengono la maggior parte delle relazioni specificate nell'elenco soprastante, tenendo conto delle seguenti eccezioni:

- Q_Elab, Alternation sono presenti solo in STAC e Minecraft;
- Parallel, Background sono presenti solo in STAC e Molweni;
- Confirmation_question è presente solo in Minecraft;
- Question_answer_pair è presente in tutti e tre i dataset, ma con diverse notazioni (QAP in Molweni, Question_answer_pair in STAC e Question-answer_pair in Minecraft).

Le Figure 3.2, 3.3 e 3.4 rappresentano la distribuzione delle relazioni in ciascun dataset utilizzato. L'asse orizzontale mostra i tipi di relazioni presenti nel dataset, mentre l'asse verticale rappresenta la percentuale relativa di ciascuna relazione rispetto al totale.

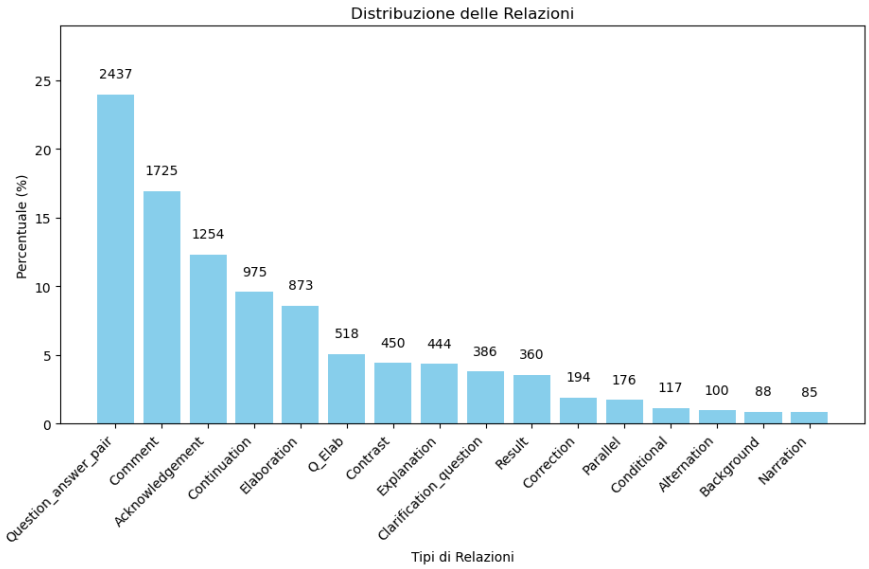


Figura 3.2: Frequenza relazioni in STAC

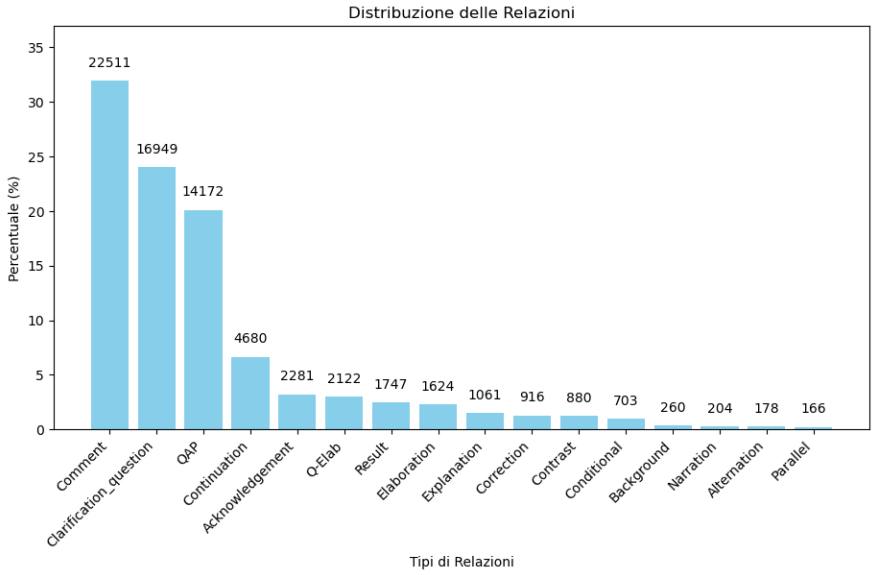


Figura 3.3: Frequenza relazioni in MOLWENI

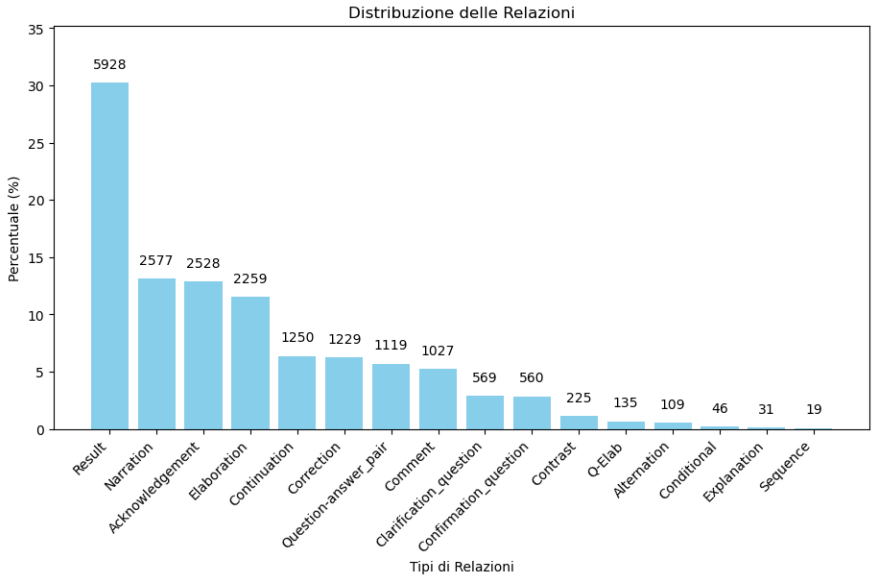


Figura 3.4: Frequenza relazioni in MINECRAFT

L'analisi condotta sui dataset e i grafici delle frequenze hanno permesso di individuare le EDU più rilevanti per il processo di Data Augmentation, scelte mediante la tecnica descritta nel Paragrafo 3.3.2.

3.2 Generazione Embeddings con SentenceTransformer

Gli **embeddings** costituiscono rappresentazioni vettoriali numeriche e consentono di mappare frasi, parole o documenti in uno spazio vettoriale continuo, riuscendo a catturare il significato semantico delle parole e facilitando quindi l'elaborazione del linguaggio naturale da parte di modelli di Machine Learning.

Nel contesto di questa analisi, la generazione degli embeddings per i tre dataset considerati è una fase cruciale, poiché fornisce una rappresentazione strutturata e significativa delle informazioni che sarà poi utilizzata nella fase di implementazione delle Graph Neural Networks (GNN).

Lo strumento utilizzato per la generazione degli embeddings è **SentenceTransformer**, una libreria offerta da offerto da HuggingFace e ottimizzata per la creazione di rappresentazioni vettoriali dense a partire da frasi o documenti. SentenceTransformer offre diversi modelli pre-trained, tra cui MiniLM e MPNet, ciascuno differente per alcune caratteristiche. Nel dettaglio le differenze principali tra i due modelli citati sono:

- **MiniLM** `(all-MiniLM-L6-v2)` è un modello leggero e veloce, progettato per generare embedding di dimensioni più contenute. È particolarmente adatto in scenari dove le risorse computazionali sono limitate o è richiesta una maggiore rapidità;
- **MPNet** `(all-mpnet-base-v2)` è un modello più grande e potente, che genera embedding di dimensioni maggiori rispetto a MiniLM. Questa caratteristica consente di catturare una quantità di informazioni semantiche più ricca e precisa, rendendolo ideale per task in cui la qualità degli embedding è prioritaria.

Vengono riportati nella Tabella 3.2 i differenti parametri tra i due modelli offerti da SentenceTransformer.

Parametri	MiniLM	MPNet
Max Sequence Length	256	384
Dimensions	384	768
Size	80MB	420MB
Speed	14200	2800
Avg Performance	58.80	63.30

Tabella 3.2: Confronto tra MiniLM e MPNet

La voce Avg Performance nella Tabella soprastante indica le performance medie ottenute considerando sia le performance di embeddings ottenuti su singole frasi che le performance in merito agli embeddings di query di ricerca e paragrafi. Inoltre, entrambi i modelli utilizzati sono stati addestrati su più di 1 Bilione di dati senza far riferimento a specifici task, quindi sono modelli general purpose.

Dopo un'analisi approfondita, è stato scelto il modello **MPNet** per la sua capacità di generare embeddings più ricchi e dettagliati. Nonostante la maggiore dimensionalità degli embeddings implichi un costo computazionale più alto, i vantaggi in termini di precisione e capacità rappresentativa sono stati decisivi per distinguere con efficacia le differenze semantiche tra le EDU (Elementary Discourse Unit) analizzate.

3.3 Data Augmentation con LLM

La fase di analisi del dataset, con particolare attenzione alla Sezione 3.1.1 relativa alla descrizione e esplorazione delle relazioni intrinseche tra i dati, ha posto le basi per la fase di **Data Augmentation** del dataset considerato. Questa fase implementativa si articola in due passaggi cruciali, descritti in dettaglio in questa sezione. Il primo consiste nella progettazione di un prompt ottimale, studiato per guidare l'LLM selezionato, assicurando che il modello comprenda appieno il task da eseguire. Successivamente, in stretta correlazione con il primo passaggio, si svolge la fase di generazione del testo. In questa fase, l'LLM produce contenuti basati sul prompt definito, con l'obiettivo di supportare efficacemente il processo di Data Augmen-

tation, generando esempi aggiuntivi che arricchiscono e diversificano il dataset di riferimento.

3.3.1 Prompt Engineering

Il **prompt** si basa sulla definizione di istruzioni dettagliate e vincoli strutturali progettati per guidare un modello linguistico nell'esecuzione di un compito specifico. L'ambito del **prompt engineering** studia come ottimizzare i prompt, ossia il modo in cui le richieste vengono formulate e strutturate, per sfruttare al massimo il potenziale dei modelli linguistici. Questa disciplina mira a definire istruzioni efficaci che riducano l'ambiguità e migliorino le prestazioni del modello, adattandolo a una vasta gamma di applicazioni, dalla generazione di contenuti alla risoluzione di problemi complessi. In particolare, l'ottimizzazione dei prompt implica l'uso di tecniche come la definizione di vincoli, l'inclusione di esempi concreti e l'esplicitazione di regole per massimizzare la precisione e l'affidabilità delle risposte del modello.

Un prompt ben strutturato quindi non solo facilita la generazione di dati coerenti e rilevanti, ma consente anche di sfruttare appieno il potenziale del modello, massimizzando la qualità e la varietà dei dati sintetici prodotti.

Il prompt engineering offre una vasta gamma di tecniche avanzate per definire un prompt ben strutturato. Tra queste, la tecnica utilizzata in questa analisi è il **Few-Shot Prompting**. Questa metodologia prevede di includere nel prompt non solo regole e vincoli chiari, ma anche esempi concreti relativi al task richiesto, al fine di guidare il modello nell'interpretazione e nell'esecuzione del compito. Attraverso gli esempi forniti, il modello apprende il contesto necessario per generare un nuovo EDU rispettando le regole e le relazioni semantiche definite. Questa tecnica permette al modello di risolvere il task in modo efficace, evitando al contempo il rischio di sovradattarsi agli esempi forniti. Gli esempi sono infatti generici e servono esclusivamente come guida, senza vincolare il modello a risposte predefinite. Inoltre, la vasta quantità di dati su cui i modelli linguistici avanzati sono addestrati consente loro di generare risposte originali senza limitarsi a riproporre gli esempi forniti, una problematica che

potrebbe invece verificarsi con modelli di dimensioni ridotte durante l'applicazione del Few-Shot Prompting.

Di seguito viene fornito un esempio del prompt definito al fine di generare dati sintetici validi.

Esempio di Prompt (1/2)

When one EDU is removed, you must generate a new EDU that replaces the removed one. The new EDU must be a strict paraphrase of the removed EDU, without referencing, modifying, or inferring any details from the other EDUs. The relationships between EDUs will remain implicitly preserved without needing to be explicitly mentioned in the generated EDU.

###Rules:

1. Each EDU represents a single, coherent idea.
2. Relationships between EDUs are defined in the format: [EDU1] $\rightarrow$ [EDU2]: <Relation>.
3. When an EDU is removed:
   - Generate a new EDU that is strictly a paraphrase of the removed EDU. The new EDU must only contain information present in the removed EDU and must not introduce, modify, or infer any details from other EDUs in the graph. Do not include any content from EDUs before or after it.
   - Ensure the generated EDU fits naturally with all connected EDUs.
   - Preserve all semantic relationships involving the missing EDU.

###Relationship Types:

1. **Result:** One EDU describes the cause of its effect described in the other.
2. **Acknowledgement:** One EDU acknowledges or affirms the information in the other, signaled by words like OK, Right, Right then, Good, Fine.

Esempio di Prompt (2/2)

###Example:

Input EDUs:

EDU1: *how is the sky?*

EDU2: *The sky is overcast.*

EDU3: *It might rain later.*

Relationships:

EDU1 → [EDU2]: Question_answer_pair

EDU2 → [EDU3]: Comment

The removed EDU is the EDU that appears in all relations specified in relationships.

EDU2 is removed. Generate a new EDU that logically replaces only EDU2, preserving all semantic relationships of EDU2 with other EDUs, defined in the previous list Relationships: *'The weather looks gloomy, which might indicate rain.'*

—

Now, process the following graph:

Input EDUs:

EDU1: *yes*

EDU2: *1bd1p 1bf1p*

EDU3: *now put one on top*

EDU4: *yes*

Relationships:

EDU1 → [EDU2]: Result

EDU2 → [EDU3]: Result

EDU2 → [EDU4]: Acknowledgement

EDU2 is removed. Generate a new EDU that logically replaces only EDU2, preserving all semantic relationships of EDU2 with other EDUs, defined in the previous list Relationships:

3.3.2 Scelta dell'LLM e Generazione dati

I **Large Language Models (LLM)**, sviluppati e resi disponibili da organizzazioni come Meta e OpenAI, rappresentano un'ampia gamma di strumenti avanzati che continuano a evolversi rapidamente. Le prestazioni di questi modelli sono in costante miglioramento, rendendoli sempre più efficaci nella risoluzione di una vasta gamma di task, che spaziano dalle attività più semplici, come il completamento di frasi, a quelle più complesse, come la comprensione semantica e la generazione di contenuti strutturati.

Nell'ambito della Data Augmentation, ci siamo concentrati sul task di generazione del testo. Questo task sfrutta la capacità degli LLM di creare output coerenti e rilevanti, ampliando i dataset e arricchendoli con esempi diversificati che rispettano i vincoli semantici e stilistici richiesti.

Tale fase è stata condotta, per ognuno dei tre dataset considerati, sui set di training. Nello specifico, per ciascun dialogo del dataset, rappresentato da una struttura a grafo descritta dettagliatamente nella Sezione 3.1, è stato scelto un nodo target in base al quale generare nuovi dati sintetici per ampliare il set di dati. Per la scelta della EDU target vengono inizialmente considerate le frequenze delle relazioni, ottenute in base a quanto descritto in 3.1.1. Dopo aver estratto le relazioni di un nodo in termini di archi entranti e uscenti, valutiamo la tipologia di relazioni con i suoi vicini. Viene quindi effettuata una somma dei pesi associati a ciascuna relazione del nodo (sulla base della frequenza). Tale valore è poi normalizzato per il numero di archi entranti e uscenti da quel nodo. Questo consente quindi di scegliere una EDU target non considerando il nodo con grado maggiore, ma selezionando il nodo le cui relazioni sono le più frequenti nel dataset.

I modelli testati in tale analisi sono **Llama-2**, proposto in [4] e sviluppato da Meta, e **GPT-4o Mini**, una versione compatta del modello GPT-4o di OpenAI.

Il modello Llama-2 (`meta-llama/Llama-2-7b-chat-hf`), offerto da HuggingFace, seppur pre-addestrato su circa 2 trilioni di token e fine-tuned su una grande

quantità di dati, non ha fornito risultati positivi per il compito richiesto, sebbene gli sia stato fornito un prompt molto preciso. Tale problematica può essere dovuta ad una diversità tra i dati utilizzati per l'addestramento del modello e i dataset utilizzati per la generazione del testo. Inoltre, al fine di bilanciare il trade-off tra complessità spaziale e temporale e prestazioni ottenute, il modello testato è quello con parameter size pari a 7B. Pertanto, utilizzando sempre il modello Llama-2 con un numero di parametri maggiore, le prestazioni sarebbero potute essere migliori.

Viceversa, il modello GPT-4o Mini (`gpt-4o-mini`) ha consentito una generazione di testo ottimale per il nostro task, garantendo maggiore efficienza e qualità delle EDU sintetiche generate. Rispetto a Llama-2-7B, GPT-4o Mini (in cui 'o' sta per omni) si è distinto per la sua capacità di comprendere meglio il contesto specifico del task e produrre testi coerenti e rilevanti, anche in presenza di prompt complessi. Questo risultato potrebbe essere attribuibile alla natura multimodale del modello e alla sua architettura ottimizzata, che consente di generalizzare meglio su dataset non direttamente correlati ai dati di training. Inoltre, GPT-4o Mini offre un compromesso ideale tra efficienza computazionale e qualità del testo generato, rendendolo particolarmente adatto per scenari in cui risorse computazionali limitate rappresentano un vincolo. Pertanto la scelta dell'LLM per la fase di Data Augmentation è ricaduta su tale modello.

Più nello specifico, per ogni dialogo e mediante un'API di GPT-4o Mini, vengono generate tre distinte EDU per il nodo target individuato. Successivamente, date le tre EDU generate, la EDU da inserire nel dataset aumentato è scelta mediante l'esecuzione di un task di link prediction. Maggiori dettagli in merito sono forniti nella Sezione 3.4.3.

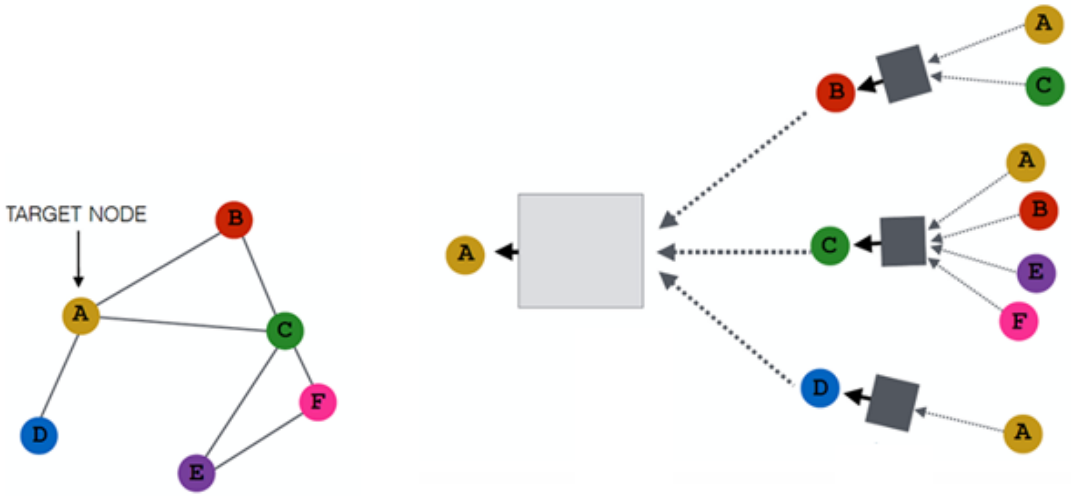
3.4 Validazione degli EDU generati tramite GNN

Le EDU generate secondo la strategia descritta in precedenza vengono validate mediante un task di link prediction eseguito tramite GNN. L'obiettivo è di verificare se la nuova EDU E', che sostituisce E, preserva le relazioni discorsive cui partecipava

E. Tra le EDU generate, viene effettivamente aggiunta al dataset quella risultata migliore in fase di validazione. Il criterio di scelta è descritto in dettaglio in Sezione 3.4.3.

3.4.1 Introduzione alle GNN

Le Graph Neural Networks (GNN) sono una famiglia di reti neurali progettate per apprendere dai grafi. Nello specifico, esse sono utilizzate per produrre, per ciascun nodo del grafo, una rappresentazione in forma di **embedding** che ne codifica le caratteristiche. L'obiettivo è quello di arricchire la rappresentazione iniziale di ogni nodo con informazioni di contesto che consentano di migliorare la qualità delle predizioni. La costruzione dei node embedding si basa sul meccanismo di **message passing**, eseguito sul grafo di computazione che si ottiene dalla visita del vicinato di ciascun nodo. Il numero di layer k della GNN definisce la profondità con la quale viene esplorato il grafo di input per la costruzione del grafo di computazione. La rappresentazione del nodo è aggiornata combinando, tramite il suddetto grafo di computazione, le feature dei nodi vicini. Non essendo definita una relazione d'ordine tra i nodi del grafo, la funzione di aggregazione utilizzata è invariante o equivariante alla permutazione (i.e. media, somma, max pooling). Ciò assicura che i node embedding prodotti non dipendano dall'ordine dei nodi considerato. L'intero processo è sinteticamente descritto in Figura 3.5.



(a) Grafo di input

(b) Grafo di computazione costruito per il message passing, k=2

Figura 3.5: Rappresentazione del message passing per la costruzione del node embedding di A. Le box nere rappresentano le funzioni di aggregazione (invarianti/equivarianti alla permutazione) dei vicini.

3.4.2 Implementazioni

Per lo sviluppo del progetto, abbiamo deciso di utilizzare le due diverse implementazioni GraphSAGE e Graph Attention Network (GAT), in modo da confrontarne i risultati e scegliere l'implementazione migliore.

GraphSAGE

GraphSAGE (Graph Sample and Aggregate) è una specifica implementazione di GNN, che consente di utilizzare, come funzione di aggregazione AGG, diverse funzioni, quali media, max pooling o aggregazione tramite LSTM (Long Short-Term Memory, variante architetturale di una RNN). Il message passing per il nodo v al layer l della rete è eseguito secondo la seguente formula di aggiornamento.

$$h_v^{(l)} = B_l \cdot h_v^{(l-1)} + W_l \cdot AGG(\{h_u^{(l-1)}, \forall u \in N(v)\})$$

Al layer l, l'embedding di v è aggiornato aggregando gli embedding dei vicini con la funzione di aggregazione AGG e pesando il risultato con una matrice di parametri

W_l. Per evitare di perdere le informazioni computate ai layer precedenti, il risultato è a sua volta aggregato con l'embedding di v al layer *l-1* (preventivamente moltiplicato per un'ulteriore matrice di pesi B_l). Si osservi che, non essendo definita una relazione d'ordine tra i nodi e non essendo noto a priori il numero di vicini, le matrici di parametri fanno inevitabilmente riferimento ai layer: a tutti i nodi dello stesso layer vengono applicati gli stessi pesi.

Anche in grafi di grandi dimensioni, GraphSAGE realizza in modo efficace il message passing fissando un limite superiore h al numero di vicini di cui aggregare le informazioni. Ciò evita che il message passing risulti eccessivamente oneroso in presenza di molti hub node (nodi aventi un gran numero di vicini). In presenza di un numero di vicini maggiore di h, gli h vicini da coinvolgere nel message passing vengono scelti tramite campionamento. Tuttavia, la nostra implementazione non necessita di tale meccanismo, in quanto i grafi considerati rappresentano relazioni discorsive tra EDU di brevi dialoghi, per cui i grafi considerati sono di dimensioni molto piccole.

Graph Attention Network

Diversamente da GraphSAGE, le Graph Attention Network (GAT) utilizzano meccanismi di *attention* per pesare in maniera adattiva l'importanza dei vicini durante l'aggregazione. Ciò consente di dare un peso maggiore ai vicini più importanti, migliorando la qualità degli embedding prodotti. Come evidenziato dalla seguente formula di aggiornamento, un coefficiente di attention $\alpha_v^{(l)}$ è utilizzato per pesare, dato un nodo v e un suo vicino u, l'importanza dell'embedding di u nell'aggiornamento della rappresentazione di v.

$$h_v^{(l)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(l)} W h_u^{(l-1)}\right)$$

I coefficienti di attention sono ottenuti nel modo seguente. Dati due nodi u e v, si calcola un valore grezzo e_{uv}, che quantifica il grado di relazione tra i due nodi.

$$e_{uv} = a\left(\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_v^{(l-1)}\right) = \text{Linear}\left(\text{Concat}\left(\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_v^{(l-1)}\right)\right)$$

Nello specifico, e_{uv} è ottenuto dalla combinazione lineare della concatenazione degli embedding dei due nodi, preventivamente moltiplicati per la matrice di parametri del layer. La funzione lineare utilizzata è tipicamente implementata come una rete single-layer, i cui pesi vengono appresi in fase di training. Infine, i valori grezzi vengono normalizzati tramite la softmax per ottenere i coefficienti da utilizzare nel calcolo.

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

Per migliorare la stabilità del modello, GAT usa l'attention **multi-head**. I moduli reali si compongono di n head di attention, ciascuna delle quali utilizza una diversa matrice di parametri e, di conseguenza, produce coefficienti differenti. I diversi valori prodotti vengono poi combinati tra loro, i.e. facendone la media. Ciò consente di catturare informazioni diverse da più prospettive.

Scelta del numero di layer k

Contrariamente a quanto avviene con altri modelli, l'incremento del numero di layer comporta tipicamente un peggioramento delle performance a causa dell'**oversmoothing**: i node embedding diventano tutti molto simili - o addirittura identici - finendo per non riuscire a discriminare i singoli nodi. Ne consegue una scarsa qualità delle predizioni. La causa è da individuare nella sovrapposizione dei campi recettivi: l'eccessivo incremento di k fa sì che i diversi nodi abbiano vicinati sovrapposti; i loro embedding, pertanto, risulteranno molto simili in quanto ottenuti dall'aggregazione di vicinati molto simili. Pertanto, soprattutto in presenza di grafi di piccole dimensioni - come i DAG che rappresentano le relazioni discorsive tra EDU - è opportuno che k sia molto piccolo. Tale parametro viene scelto a seguito della fase di validazione, in cui vengono definiti i parametri ottimali per ognuna delle due architetture GNN implementate.

3.4.3 Link Prediction

Una delle principali applicazioni delle GNN è la link prediction: dati due nodi, si vuole predire la probabilità che essi siano collegati da un arco. Come specificato ad

inizio sezione, la validazione degli EDU è modellata come un task di link prediction.

Siano E la EDU originariamente presente nel dataset, E' la nuova EDU generata in sostituzione di E. Affinché sia validato, E' è fornito al link predictor per verificare che preservi le relazioni cui partecipava E. Si noti che la GNN è uno strumento particolarmente indicato a tale scopo, in quanto le relazioni discorsive tra EDU sono rappresentate in forma di DAG (grafo diretto e aciclico). Affinché il DAG possa essere processato tramite la GNN, ogni nodo contiene l'embedding del rispettivo EDU (non la rappresentazione testuale), prodotto secondo la strategia descritta in precedenza.

Gli step dell'implementazione sono i seguenti.

1. **Utilizzo della GNN per la produzione di embedding arricchiti.** A partire dai word embedding iniziali, la GNN produce, per ciascun nodo, rappresentazioni arricchite grazie al meccanismo di message passing. I node embedding risultanti codificano, tra le altre cose, le relazioni esistenti tra le EDU.

2. **Utilizzo di un link predictor**. Per predire l'esistenza di un arco tra i due nodi, gli embedding dei nodi connessi dall'arco candidato vengono forniti ad un link predictor, che stima la probabilità di esistenza dell'arco. Se la probabilità stimata è ≥ 0.5, l'arco viene predetto come esistente, altrimenti come non esistente. La nostra implementazione utilizza una **rete neurale feedforward fully-connected** per la link prediction.

Come accennato in Sezione 3.3.2, date le tre EDU generate tramite GPT-4o Mini, la EDU da inserire nel dataset aumentato è scelta mediante l'esecuzione di un task di link prediction. L'obiettivo è quello di valutare la capacità di ciascuna nuova EDU di preservare le relazioni cui partecipava l'EDU target. Gli step effettuati sono i seguenti.

a) Si rimuovono nel DAG tutti gli archi incidenti sul nodo target (sia in entrata che in uscita).

b) Il DAG ottenuto è dato in input alla GNN per l'ottenimento di node embedding contestuali.

c) Per ogni arco rimosso, utilizzando gli embedding contestuali, se ne predice l'esistenza tramite il link predictor. Il risultato, per ognuno dei tre nuovi EDU,

è un valore di probabilità per ogni arco rimosso. Pertanto, ad ogni nuovo EDU è associato un vettore di probabilità.

d) Si scartano gli EDU il cui vettore di probabilità contiene almeno un valore < 0.5.

e) Tra gli EDU rimanenti, l'EDU avente la più alta probabilità media viene selezionato per essere inserito nel dataset aumentato.

Se, eseguendo lo step *(c)*, tutte e tre le EDU vengono scartate, allora lo step *(d)* è eseguito su tutte e tre le EDU, considerando i vettori di probabilità iniziali. Ne consegue che, per ogni dialogo originario, viene in ogni caso inserito nel dataset aumentato un nuovo dialogo.

Split del dataset

L'effettuazione di training, validazione e testing richiede di splittare il dataset in training set, validation set, test set. Per la link prediction - e più in generale per i task su grafo - individuare un criterio di split del dataset non è banale, a causa dell'esistenza di relazioni tra gli elementi. Con riferimento alla link prediction, la letteratura propone i seguenti criteri di split.

Figura 3.6: Split induttivo

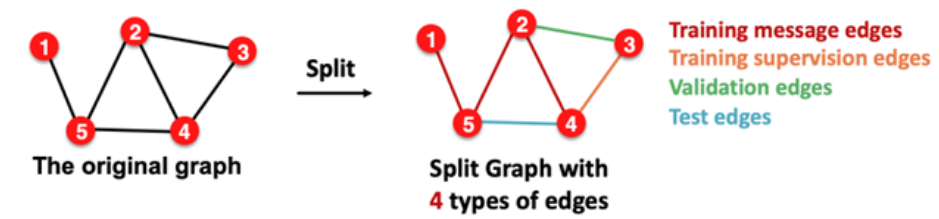


Figura 3.7: Split trasduttivo

- **Split induttivo** *(meno frequente)*. Supponendo di disporre di un dataset di 3 grafi, ciascuno di essi è utilizzato rispettivamente come training set, validation

set e test set. Inoltre, in ciascun grafo, gli archi sono suddivisi in *message edge* e *supervision edge*. Anche in fase di training, questi ultimi non vengono forniti alla GNN, ma sono comunque usati per il calcolo della loss. La strategia è rappresentata in Figura 3.6.

- **Split trasduttivo** *(più frequente)*. Supponendo di disporre di un unico grafo, i suoi archi vengono splittati in archi di training, validazione e testing, secondo quanto rappresentato in Figura 3.7.

A nostro avviso, le caratteristiche del nostro problema rendono sconvenienti entrambe le strategie. Infatti, i nostri dataset si compongono di migliaia di grafi (DAG che rappresentano le relazioni discorsive), ciascuno di dimensioni molto piccole e, di conseguenza, con un numero di archi molto ridotto (talvolta anche solo 1). Pertanto, applicare lo split sugli archi del singolo grafo è impraticabile. Disponendo già di dataset di training, validazione e testing (vedi Tabella 3.1), abbiamo banalmente eseguito il training sul training set, la validazione sul validation set, il testing sul test set. Di seguito forniamo una panoramica sulla modalità di effettuazione di training, validazione e testing dei modelli.

3.4.4 Definizione degli Iperparametri

Dopo aver definito le architetture di GNN da utilizzare e del relativo Link Predictor, è stata avviata un'analisi approfondita degli iperparametri rilevanti per ottimizzare le prestazioni del modello, al fine di individuare le configurazioni che garantiscono i migliori risultati per ciascun dataset. In particolare, l'analisi si è focalizzata principalmente sui seguenti iperparametri:

- **Dropout**: Questo parametro consente di bilanciare la prevenzione dell'overfitting con la capacità di apprendimento del modello. Valori bassi (0.1 - 0.3) sono utili se il modello è relativamente semplice o i dati di addestramento sono già ben regolarizzati. Valori più alti (0.4 - 0.6) sono preferibili per modelli più complessi, in modo da evitare l'overfitting senza compromettere troppo l'apprendimento;

- **Learning rate**: Determina la velocità con cui il modello aggiorna i pesi durante l'addestramento. In questo caso, si utilizza l'ottimizzatore Adam, che è noto per la sua efficienza nel trovare soluzioni ottimali in vari scenari;
- **Numero di layer**: Consente di definire la profondità della GNN. Ogni layer aggiuntivo consente di estrarre informazioni sempre più complesse dalle feature di partenza, ma un numero eccessivo di layer può causare problemi di oversmoothing (vedi Sezione 3.4.2), provocando un peggioramento delle performance;
- **Heads**: Questo iperparametro è specifico per il modello GAT (Graph Attention Network), analizzato in dettaglio nella Sezione 3.4.2, e riguarda la gestione dell'attenzione tra i nodi del grafo.

3.4.5 Training

Il training è stato eseguito separatamente per ciascuno dei dataset STAC, MOLWENI, MINECRAFT. Pertanto, sono state addestrate tre distinte GraphSAGE (una per dataset) e tre distinte GAT (una per dataset). Tale approccio è motivato dal fatto che l'augmentation di ognuno dei dataset è stata realizzato utilizzando la GNN addestrata su di esso. Per ragioni di efficienza, il dataset è stato suddiviso in mini-batch, in modo che il singolo aggiornamento dei pesi sia calcolato sulla loss media delle predizioni del singolo batch. Ogni DAG corrisponde ad un campione di training. A sua volta, il singolo campione si compone di un ugual numero di *sotto-campioni* positivi e negativi.

- I sotto-campioni positivi sono gli archi effettivamente presenti nel DAG;
- I sotto-campioni negativi vengono campionati, in egual numero a quelli positivi, tra gli archi non esistenti nel DAG.

La loss sul singolo DAG è data dalla media della loss sulle predizioni dei singoli sotto-campioni del DAG. La loss su un mini-batch è data dalla media della loss sui singoli DAG che lo compongono. Il singolo aggiornamento è calcolato in funzione

della loss di ogni mini-batch. La formula generica per la loss è la seguente:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\log(\texttt{pos\_pred}_i + \epsilon) - \frac{1}{M}\sum_{j=1}^{M}\log(1 - \texttt{neg\_pred}_j + \epsilon) \tag{3.4.1}$$

dove:

- `pos_pred` rappresenta le predizioni per archi esistenti (positivi);
- `neg_pred` rappresenta le predizioni per archi non esistenti (negativi);
- $\epsilon = 10^{-15}$ è un valore molto piccolo aggiunto per evitare problemi di logaritmi nulli.

Inoltre, il training è stato eseguito in modo incrementale. Ogni modello è stato inizialmente addestrato su 10 epoche in modo da valutarne la loss e le metriche ottenute in fase di validazione. In funzione dei valori ottenuti, si è optato per l'esecuzione di ulteriori epoche di training, nella misura ritenuta necessaria. Il numero di epoche di training complessivamente eseguite per le diverse reti e i diversi dataset è rappresentato in Tabella 3.3.

	STAC	**MOLWENI**	**MINECRAFT**
GraphSAGE	100	100	100
GAT	30	30	30

Tabella 3.3: Numero complessivo di epoche di training

3.4.6 Validazione

In questa fase, successiva al training, è stato eseguito il fine-tuning degli iperparametri, con particolare attenzione al learning rate α e al numero di layer k della GNN. Più nello specifico, ciascun modello addestrato è stato valutato sul validation set, calcolando accuracy, precision e recall sui risultati ottenuti. Analogamente a quanto fatto in fase di training, per ogni DAG del validation set sono stati utilizzati gli archi esistenti come campioni positivi, e un ugual numero di archi non esistenti è stato campionato in modo da ottenere i campioni negativi. In funzione delle metriche

calcolate, si è eventualmente stabilito di modificare uno o più iperparametri, e di rieseguire il training con la nuova combinazione. Gli iperparametri valutati sono stati descritti nella Sezione 3.4.4.

3.4.7 Testing

Entrambi i modelli, addestrati e validati, sono stati testati sui rispettivi test set, secondo la stessa strategia adottata in fase di training e validazione. Per ognuna delle due GNN implementate e per ciascun dataset, sono state calcolate metriche di qualità quali accuracy, precision, recall. Per il dataset di Minecraft (l'unico aumentato per le ragioni illustrate nell'introduzione del Capitolo 3), l'augmentation è stata realizzata utilizzando la GNN (GraphSAGE o GAT) che, in questa fase, consegue il più alto valore di accuracy.

3.5 Pipeline di Augmentation

La metodologia descritta rispettivamente nelle Sezioni 3.3 e 3.4.3 viene unificata mediante l'implementazione di una pipeline automatizzata di Data Augmentation, che unisce i vari step implementativi fornendo maggiore efficienza e una migliore gestione del codice.

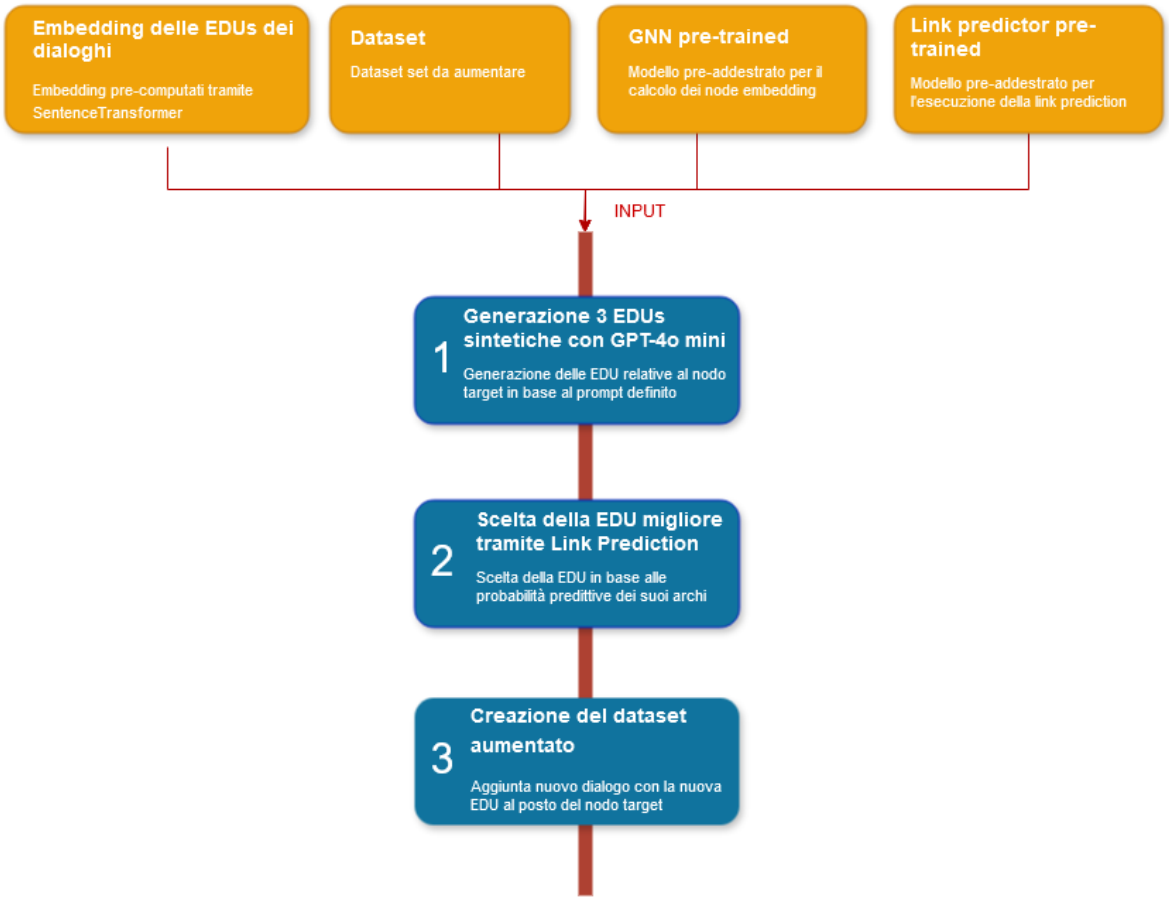


Figura 3.8: Pipeline di Data Augmentation

Come mostrato in Figura 3.8, gli step nella definizione della pipeline sono i seguenti.

1. **Generazione 3 EDUs sintetiche con GPT-4o mini.** Tale fase è condotta effettuando una chiamata all'API del modello GPT-4o mini, descritto in dettaglio nella Sezione 3.3.

2. **Scelta della EDU migliore con Link Prediction.** Nella fase di validazione del dataset aumentato tramite task di link prediction, viene effettuata la scelta di una delle tre EDU generate nel primo punto della pipeline. Questa selezione è fatta seguendo la tecnica descritta in 3.4.3;

3. **Creazione del dataset aumentato.** Lo step finale riguarda quindi l'ottenimento del nuovo dataset, contenente i dati sintetici validati.

In input alla pipeline vengono forniti i seguenti elementi:

- embedding delle EDU dei dialoghi (nella nostra implementazione pre-computati tramite MPNet, vedi dettagli Sezione 3.2);
- dataset da sottoporre al processo di Data Augmentation (nella nostra implementazione Minecraft, vedi dettagli in Sezione 3.1);
- GNN pre-addestrata per il calcolo dei node embedding contestuali (nella nostra implementazione GraphSAGE o una GAT, vedi dettagli in Sezione 3.4);
- link predictor pre-addestrato per la scelta, tra le tre generate, della EDU migliore (nella nostra implementazione una rete neurale feedforward, vedi dettagli in Sezione 3.4.3).

I tre step della pipeline sono implementati, rispettivamente, dalle funzioni `save_all_new_edus()`, `choose_edus()`, `save_all_new_edus_batch()` della classe `DataAugmentationPipeline`, presente nello script `main.py`. Quindi, le fasi di training della GNN e del relativo link predictor, così come la costruzione degli embeddings, vengono eseguite al di fuori del flusso di esecuzione della pipeline. L'implementazione della pipeline, pertanto, è indipendente dal tipo di embedding, dal dataset da aumentare, nonché dalla GNN e dal link predictor addestrati per l'effettuazione delle predizioni. Nella repository GitHub, forniamo i nostri dataset e i nostri script per la costruzione degli embedding ed il training, la validazione, il testing e l'utilizzo dei modelli. In ogni caso, tali elementi possono essere sostituiti, con la garanzia che la pipeline continui a funzionare correttamente. Infine, ogni funzione della pipeline salva il proprio output su file, in modo che ciascun metodo recuperi gli input necessari direttamente da file. Ciò consente di ottenere funzioni indipendenti tra loro, facilitando il riutilizzo del codice e migliorando la modularità.

La pipeline così definita consentirà, al termine dell'esecuzione, di ottenere il dataset aumentato contenente dati sintetici validi.

Nota: nello script `GPT4.py`, il client usato per interrogare l'API è istanziato con dei token di prova gratuiti (ora esauriti), che prevedono rate limit molto stringenti per l'invio di richieste. Per generare nuove EDU tramite la funzione `save_all_new_edus()`, è necessario inserire un nuovo token che consenta di interagire con l'API. È

opportuno utilizzare `save_all_new_edus()` nella pipeline solo se si dispone di un token che consente di interrogare l'API senza alcun limite di richieste. Viceversa, il processo di generazione sarà interrotto al raggiungimento del limite. Se non è possibile inviare richieste illimitate, è necessario interrogare l'API per piccoli mini-batch di dialoghi; nel nostro caso, la massima dimensione consentita dal token era di 3 dialoghi. Di conseguenza, abbiamo dovuto eseguire la generazione delle nuove EDU (step 1) al di fuori della pipeline. A tale scopo, abbiamo utilizzato la funzione `save_all_new_edus_batch()`. In questo caso, non si vuole includere la generazione delle nuove EDU nella pipeline, per cui quest'ultima sarà inizializzata con `all_new_edus_available = True`.

Listing 3.1: Utilizzo della pipeline con nuovi EDU già generati con *save_all_new_edus_batch()*

```
dataAugmenter = DataAugmentationPipeline(
    dataset_name='MINECRAFT',
    dataset_filename=MINECRAFT_dataset_filename,
    trained_GNN=trained_model,
    trained_link_predictor=trained_link_predictor,
    all_new_edus_available=True
)

dataAugmenter()
```

3.6 Valutazione dell'Efficacia

Una volta ottenuti i dataset aumentati validati, lo step finale è quello di valutare se tali dataset riescono ad ottenere performance superiori a quelle ottenute con il dataset iniziale. Ciò che viene fatto è quindi testare lo stesso modello con il dataset prima e dopo la Data Augmentation.

Un ulteriore scopo di tale fase è quello di valutare la capacità del dataset aumentato di generalizzare a nuovi modelli. Quindi, l'idea è sì quella di testare lo stesso modello

con entrambe le versioni dei dataset, ma di farlo su un nuovo modello non utilizzato all'interno della pipeline implementata.

Il modello scelto è **Seq2Seq-DDP**, un modello implementato da Li et al. e descritto in [5] (apri la repository GitHub cliccando qui). Tale modello, per sfruttare appieno la ricca conoscenza semantica e discorsiva degli LLM, propone di trasformare il tradizionale parsing del discorso (Discourse Dialogue Parsing - DDP) in un compito di generazione utilizzando un paradigma testo-testo (Seq2Seq), quindi prende in input una sequenza di testi grezzi e produce in output una sequenza di strutture come mostrato in Figura 3.9. Il modello Seq2Seq-DDP, utilizzando modelli pre-addestrati della famiglia T5, ha un'architettura standard encoder-decoder e inoltre l'approccio utilizzato non richiede modifiche all'architettura degli LLM. Dallo studio condotto, tale sistema ottiene risultati ragionevolmente buoni sui dataset STAC e Molweni.

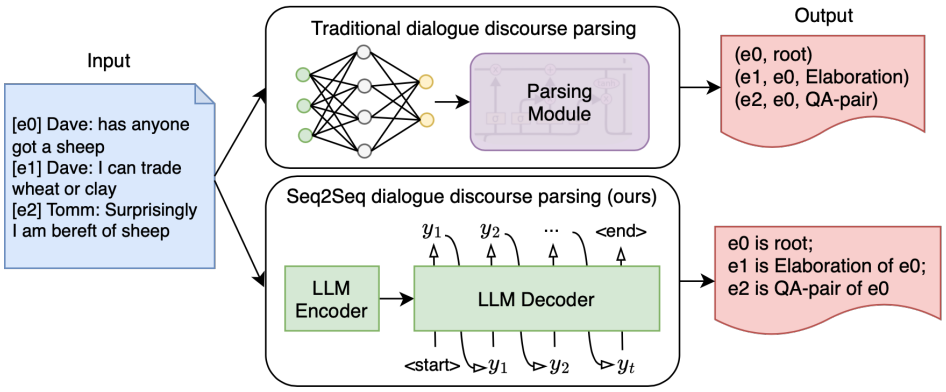


Figura 3.9: Parsing tradizionale vs. Parsing Seq2Seq del dialogo

Il modello Seq2Seq è stato implementato in modo da utilizzare i modelli pre-addestrati della famiglia T5. Nello specifico, il modello che abbiamo utilizzato in tale fase è T5-3B, una versione con 3 miliardi di parametri del modello T5, che si colloca tra il modello base T5 e le versioni più grandi come il T5-11B, offrendo un buon equilibrio tra performance e requisiti computazionali.

È stata quindi condotta la fase di training del modello rispettivamente sul training set originale di Minecraft, sul relativo set di dati aumentato utilizzando l'architettura GAT e su quello aumentato mediante il task di link prediction basato su GraphSAGE. A seguito, mediante il dataset di test, è stata eseguito il testing del modello.

CAPITOLO 4

Risultati

Nel contesto dell'implementazione, ciò che interessa maggiormente è esaminare e valutare i risultati conseguiti. Pertanto, in questo capitolo viene posta particolare attenzione ai risultati ottenuti durante le fasi di Data Augmentation e Validazione tramite task di Link Prediction della GNN. Si analizzano quindi nelle successive sezioni i dati sintetici generati tramite LLM, le prestazioni delle fasi di training, validation e testing della GNN e del relativo Link Predictor. Infine, il training di Dialogue Discourse Parsing as Generation, eseguito tramite il modello già esistente Seq2Seq, viene eseguito sul dataset di partenza e su quelli aumentati, in modo da confrontare i risultati ottenuti e verificare se l'augmentation ha determinato un miglioramento delle performance.

4.1 Training, Validation e Testing GNN

Sono state effettuate, per ognuno dei tre dataset considerati, rispettivamente le fasi di training, validation e testing della GNN implementata. Di seguito vengono riportati i risultati ottenuti, divisi per tipo di GNN - GAT o GraphSAGE - e per dataset considerato - STAC, Minecraft o Molweni - in termini di loss per le fasi di training e validation e di Accuracy, Precision e Recall per validation e testing.

4.1.1 GAT

La fase di testing della Graph Attention Network viene condotta utilizzando i modelli addestrati con 30 epoche, una batch size di 32 e un learning rate di 0.001 per ciascuno dei tre dataset considerati. Le Tabelle 4.1 e 4.2 riportano le configurazioni ottimali per ciascun dataset, sia per la GAT che per il relativo Link Predictor.

Dataset	Dim. Input	Dim. Hidden	Dim. Output	Num. Layers	Heads	Dropout
STAC	768	384	384	2	16	0.5
MINECRAFT	768	384	384	3	32	0.5
MOLWENI	768	384	384	2	32	0.5

Tabella 4.1: Configurazioni ottimali di GAT per i dataset STAC, MINECRAFT e MOLWENI.

Dataset	Dim. Input	Dim. Hidden	Dim. Output	Num. Layers	Dropout
STAC	384	192	1	2	0.5
MINECRAFT	384	192	1	3	0.5
MOLWENI	384	192	1	2	0.5

Tabella 4.2: Configurazioni ottimali di Link Predictor per i dataset STAC, MINECRAFT e MOLWENI.

Il numero di layer varia da 2 a 3, con una configurazione più profonda per il dataset Minecraft (3 layer) rispetto agli altri dataset (2 layer). Un aspetto rilevante è l'iperparametro Heads, che assume valori di 16 e 32 per i diversi dataset. In particolare, per il dataset Stac si è scelto di utilizzare un numero inferiore di Heads (16), poiché la dimensione dei singoli DAG (Directed Acyclic Graph) è più piccola rispetto agli altri dataset, e ciò consente di catturare le relazioni tra i nodi in modo efficace, evitando di complicare eccessivamente il modello.

STAC

Come mostrato in Figura 4.1, la loss ottenuta in fase di train si mantiene sempre nel range [1,1.5] ed è bassa anche nelle epoche iniziali. Dalla Figura 4.2 notiamo che,

anche se di poco, la loss è più bassa in fase di validation e quindi a seguito dello step di Hyperparameter tuning, garantendo l'ottimale scelta dei parametri. In particolare la loss media ottenuta è pari a 1.15.

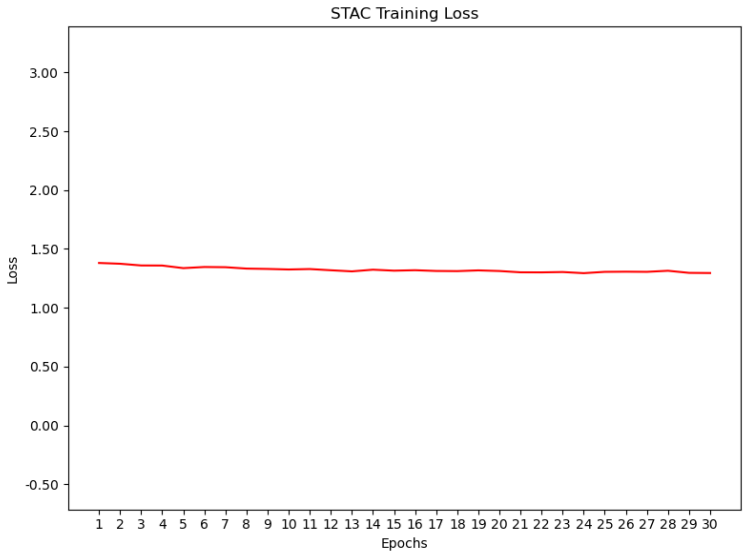


Figura 4.1: Grafico della loss in fase di training con GAT per il dataset STAC

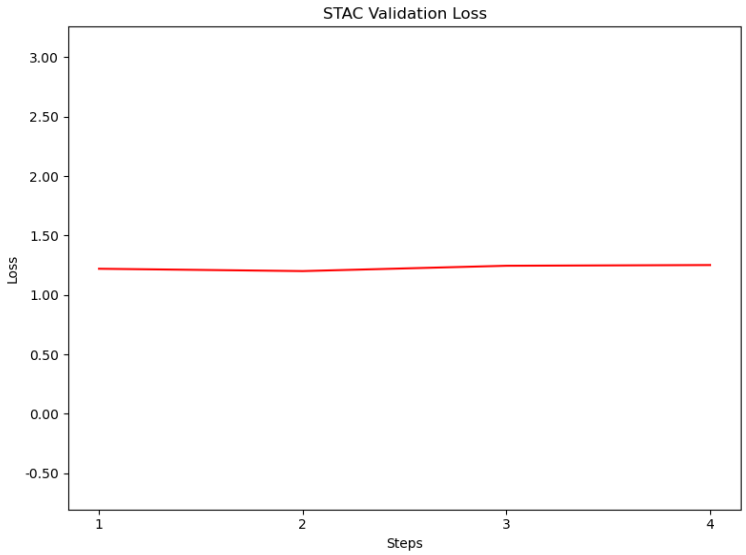


Figura 4.2: Grafico della loss in fase di validation con GAT per il dataset STAC

Vengono riportati in Tabella 4.3 i risultati ottenuti considerando le metriche di Accuracy, Precision e Recall rispettivamente nelle fasi di validazione e testing. Notiamo che per le tre metriche valori più alti sono ottenuti in fase di validazione, ma è da considerare che i valori in fase di testing non decrescono di molto, garantendo

comunque la capacità della GNN di generalizzare bene anche a nuovi dati non visti in fase di addestramento.

	Accuracy	**Precision**	**Recall**
Validation	77,61%	69,83%	97,22%
Testing	74,58%	67,35%	95,39%

Tabella 4.3: Metriche in fase di Validation e Testing per Stac con la GAT

Minecraft

Per il dataset di Minecraft i risultati della loss ottenuta rispettivamente in fase di training e validazione dei parametri sono mostrati nelle Figure 4.3 e 4.4. Notiamo che la loss assume comunque valori bassi e decrescenti in fase di training, illustrando il corretto funzionamento della rete e, a seguito della definizione dei parametri nella fase di validation, la loss ottenuta raggiunge il valore di 0.913.

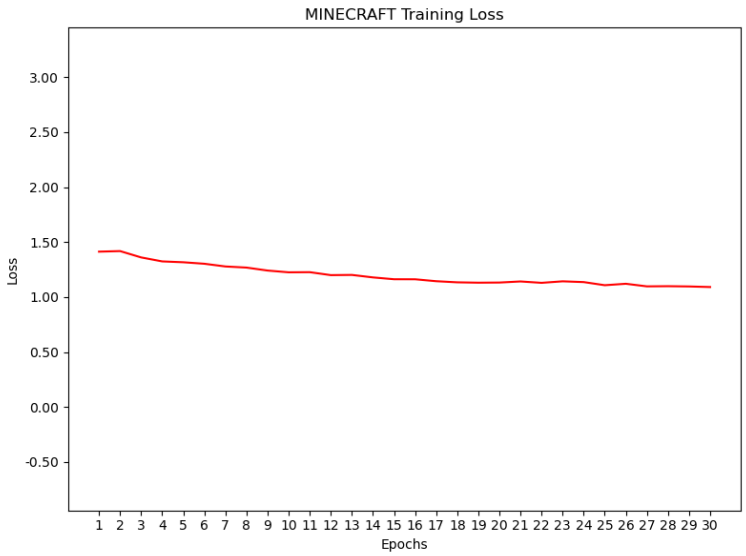


Figura 4.3: Grafico della loss in fase di training con GAT per il dataset Minecraft

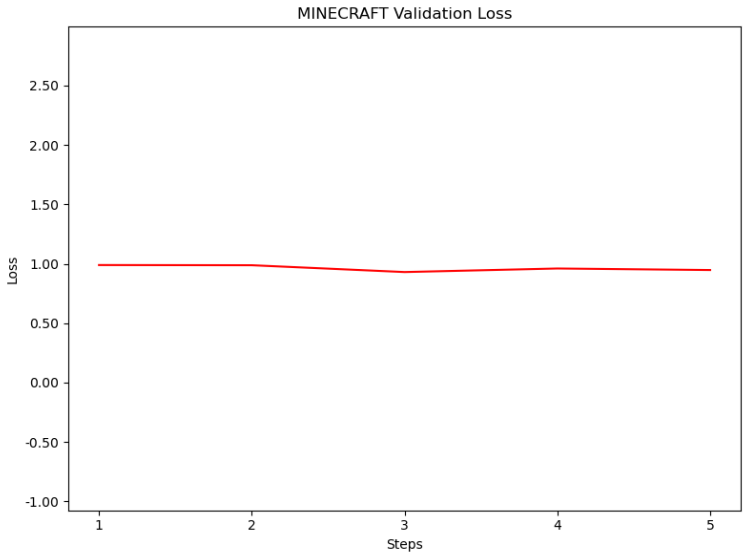


Figura 4.4: Grafico della loss in fase di validation con GAT per il dataset Minecraft

Analizzando i valori di Accuracy, Precision e Recall riportati nella Tabella sottostante, abbiamo valori pressoché simili nelle fasi di validazione e testing e questo indica una buona capacità del modello a generalizzare. Inoltre tutte le tre metriche assumono valori alti e la recall è la più alta, suggerendo che il modello riesce a identificare un numero elevato di campioni positivi.

	Accuracy	**Precision**	**Recall**
Validation	85,37%	78,05%	98,4%
Testing	85,42%	78,21%	98,21%

Tabella 4.4: Metriche in fase di Validation e Testing per Minecraft con la GAT

Molweni

Dopo aver effettuato la fase di testing e aver ottenuto la loss mostrata in Figura 4.5, variante anch'essa nel range [1,1.5], si è passati a definire i parametri ottimali della GAT implementata, ottenendo quindi una loss media di 1,22, il cui andamento è mostrato nella Figura 4.6. Quindi abbiamo anche in questo caso una loss che assume comunque valori bassi già analizzando le prime epoche condotte. Tale loss in fase di

training è comunque decrescente, mostrante quindi il corretto addestramento della GNN.

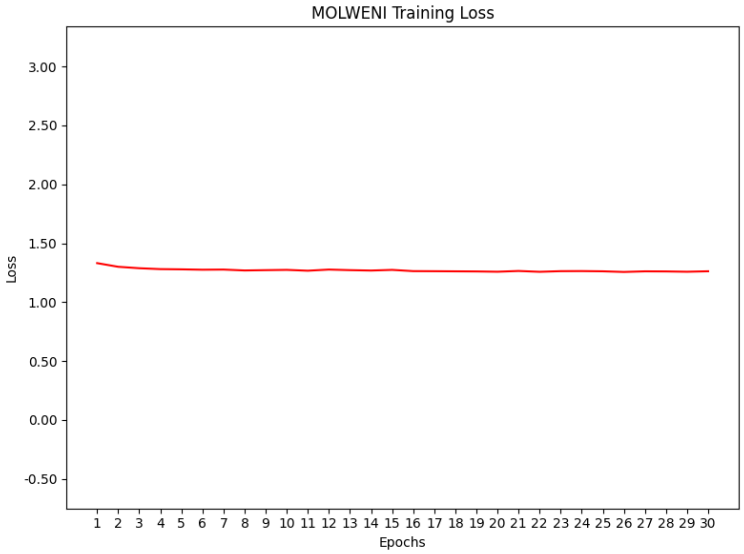


Figura 4.5: Grafico della loss in fase di training con GAT per il dataset Molweni

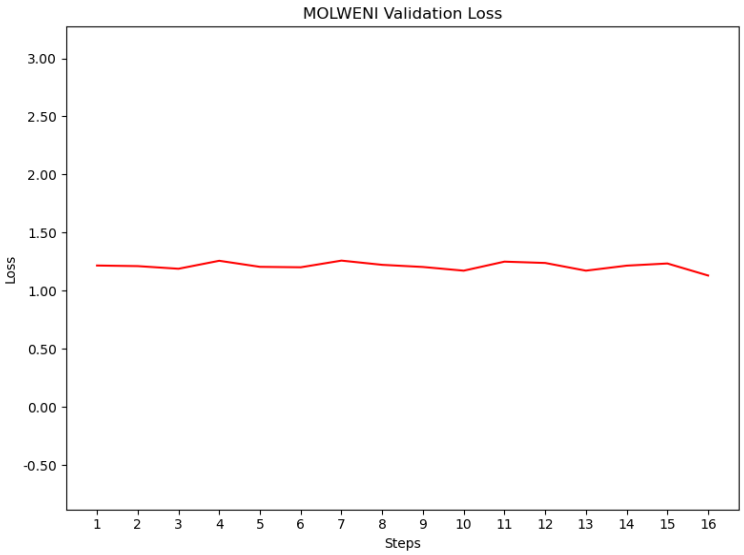


Figura 4.6: Grafico della loss in fase di validation con GAT per il dataset Molweni

Effettuando un confronto in merito alle metriche considerate rispettivamente in fase di validation e testing, abbiamo che i valori sono quasi uguali per tutte le metriche considerate e abbiamo valori molto alti per la recall, come mostrato nella Tabella sottostante. Viceversa, Accuracy e Precision assumono valori più bassi ma comunque non eccessivamente da considerarsi un problema.

	Accuracy	Precision	Recall
Validation	67,26%	60,44%	99,97%
Testing	67,02%	60,27%	99,89%

Tabella 4.5: Metriche in fase di Validation e Testing per Molweni con la GAT

Infine, come evidenziato nelle Figure 4.1, 4.3 e 4.5, si osserva che la loss in fase di training per il modello GAT assume un valore basso già nelle prime epoche, stabilizzandosi poi su valori poco più bassi man mano che il training prosegue. Questo comportamento è attribuibile al meccanismo di attenzione implementato dalla GAT, che consente al modello di calcolare embeddings di qualità superiore. Tale meccanismo migliora l'apprendimento, poiché permette al modello di dare maggiore importanza ai nodi e alle connessioni più rilevanti nel grafo, facilitando una rappresentazione più efficace dei dati fin dalle fasi iniziali del training.

4.1.2 GraphSAGE

La fase di testing della GraphSAGE viene condotta utilizzando i modelli addestrati con 100 epoche, una batch size di 32 e un learning rate di 0.0001 per ciascuno dei tre dataset considerati. Le Tabelle 4.6 e 4.7 riportano le configurazioni ottimali per ciascun dataset, sia per la GraphSAGE che per il relativo Link Predictor.

Dataset	Dim. Input	Dim. Hidden	Dim. Output	Num. Layers	Dropout
STAC	768	384	384	4	0.3
MINECRAFT	768	384	384	3	0.3
MOLWENI	768	384	384	4	0.3

Tabella 4.6: Configurazioni di GraphSAGE per i dataset STAC, MINECRAFT e MOLWENI.

Si osserva che per tutti i dataset è stato scelto un valore di dropout pari a 0.3, che risulta essere più basso rispetto a quanto utilizzato nei modelli GAT. In pratica, un dropout di 0.3 consente di ridurre il rischio di overfitting senza limitare eccessivamente la capacità del modello di generalizzare sui dati non visti. Inoltre, data una complessità inferiore della rete GraphSAGE rispetto alla GAT, si è deciso di aumenta-

Dataset	Dim. Input	Dim. Hidden	Dim. Output	Num. Layers	Dropout
STAC	384	192	1	4	0.3
MINECRAFT	384	192	1	3	0.3
MOLWENI	384	192	1	4	0.3

Tabella 4.7: Configurazioni di Link Predictor per i dataset STAC, MINECRAFT e MOLWENI.

re il numero di layer per permettere al modello di apprendere rappresentazioni più dettagliate dei dati.

STAC

La loss ottenuta in fase di training, come visibile in Figura 4.7, ha un andamento non del tutto decrescente, ma comunque i valori ottenuti sono da considerarsi buoni perché sempre al di sotto di 2. Abbiamo invece che in fase di validation, la loss assume come valore più basso 1.38. Pertanto possiamo comunque affermare che il modello si comporti bene a seguito della definizione dei parametri ottimali.

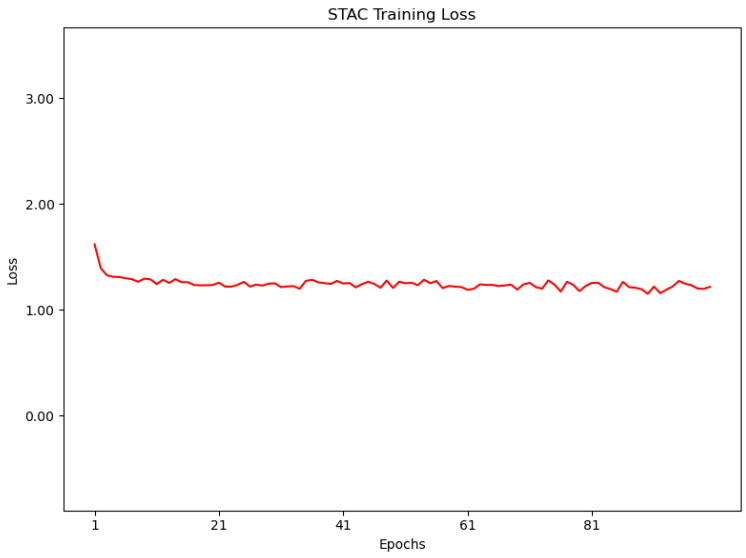


Figura 4.7: Grafico della loss in fase di training con GraphSAGE per il dataset STAC

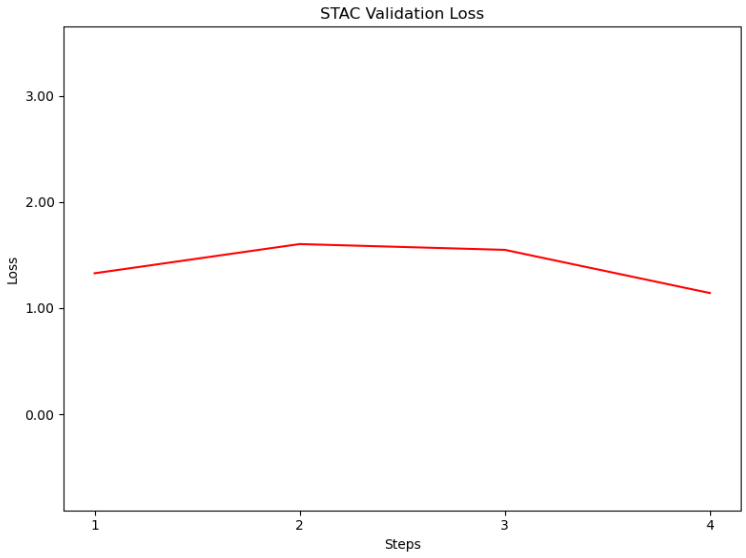


Figura 4.8: Grafico della loss in fase di validation con GraphSAGE per il dataset STAC

Esaminando i dati contenuti nella Tabella 4.8, la prima considerazione evidente è che le metriche in fase di training assumono valori molto superiori rispetto a quelli ottenuti nella validation. Questo può essere dovuto al fatto che, ottimizzando i parametri sul validation set, questo potrebbe essere stato reso più difficile per il modello rispetto al test set. Inoltre un'altra possibile alternativa è che i dati di test potrebbero essere più facili da classificare rispetto ai dati di validazione.

	Accuracy	**Precision**	**Recall**
Validation	58,84%	62,5%	44,19%
Testing	70,72%	68,03%	78,03%

Tabella 4.8: Metriche in fase di Validation e Testing per Stac con la GraphSAGE

Minecraft

Dal grafico mostrato in Figura 4.9 notiamo che inizialmente abbiamo valori più alti di loss, che tendono a decrescere all'aumentare delle epoche effettuate. Possiamo notare che i valori tendono a scendere fino a raggiungere valori prossimi a 1, indicante quindi che il modello si comporta comunque bene e non presenta un'eccessiva

perdita. Analizzando la loss in fase di validation, come mostrato in Figura 4.10, rimane comunque stabile intorno a 1 e il valore più basso raggiunto è proprio 1.09.

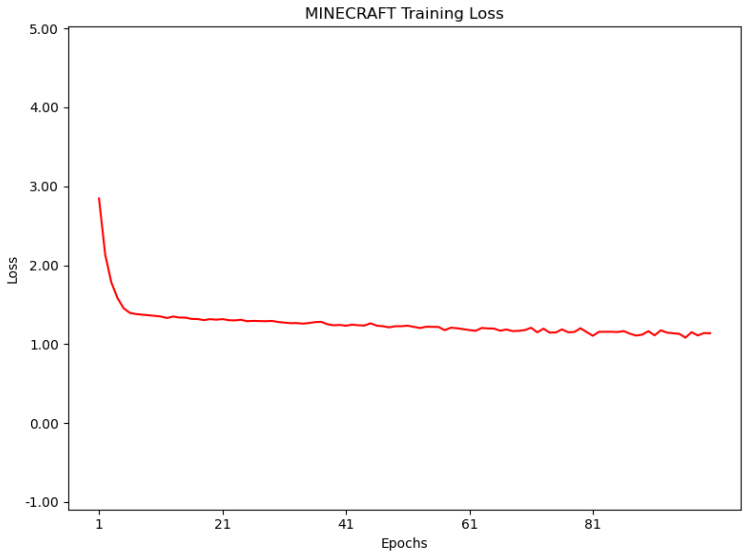


Figura 4.9: Grafico della loss in fase di training con GraphSAGE per il dataset Minecraft

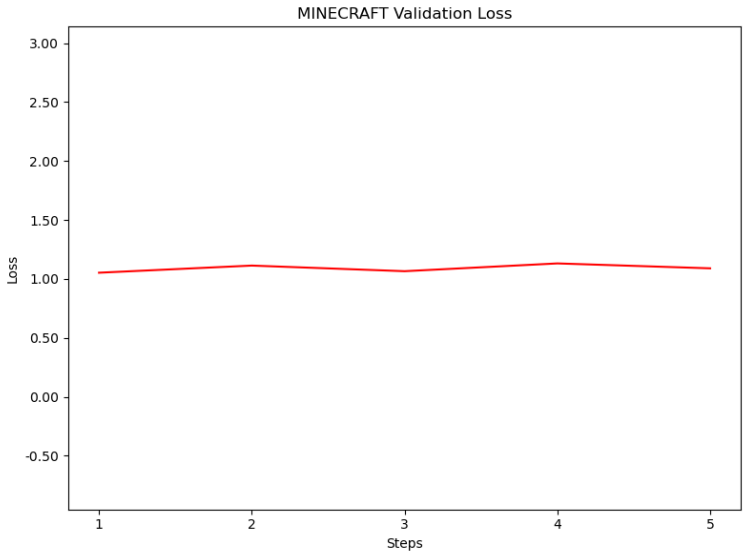


Figura 4.10: Grafico della loss in fase di validation con GraphSAGE per il dataset Minecraft

La Tabella 4.9 considerante i valori ottenuti a seguito della validazione e test sul dataset di Minecraft evidenzia che non ci sono particolari differenze tra i valori delle metriche ottenuti nelle due diverse fasi. Abbiamo comunque valori buoni per tutte e tre, e la recall, ossia la capacità di identificare i campioni positivi, è sempre il valore più alto.

	Accuracy	Precision	Recall
Validation	71,9%	67,94%	83%
Testing	71,34%	67,56%	82,11%

Tabella 4.9: Metriche in fase di Validation e Testing per Minecraft con la GraphSAGE

Molweni

Analizzando le Figure 4.11 e 4.12 notiamo che la loss anche in questo caso assume valori nel range [1,1.5] indicando di non ottenere una perdita eccessiva. All'interno della fase di validazione relativa alla definizione e tuning dei parametri considerati, la loss più bassa ottenuta è pari a 1.11.

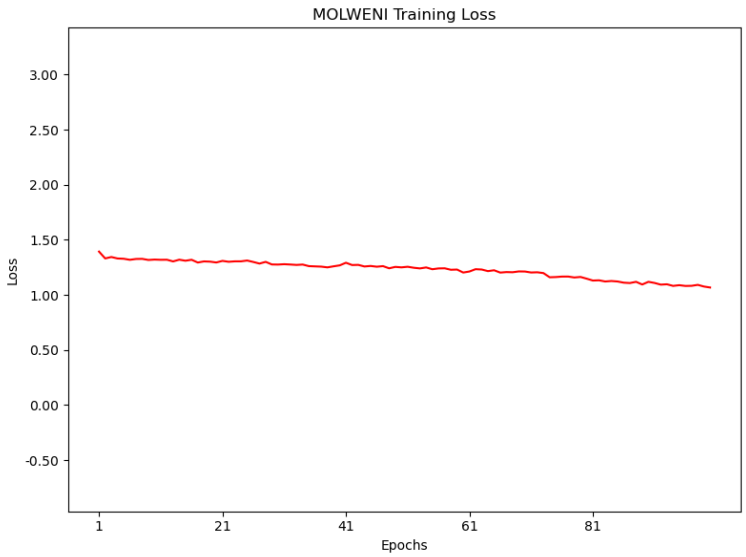


Figura 4.11: Grafico della loss in fase di training con GraphSAGE per il dataset Molweni

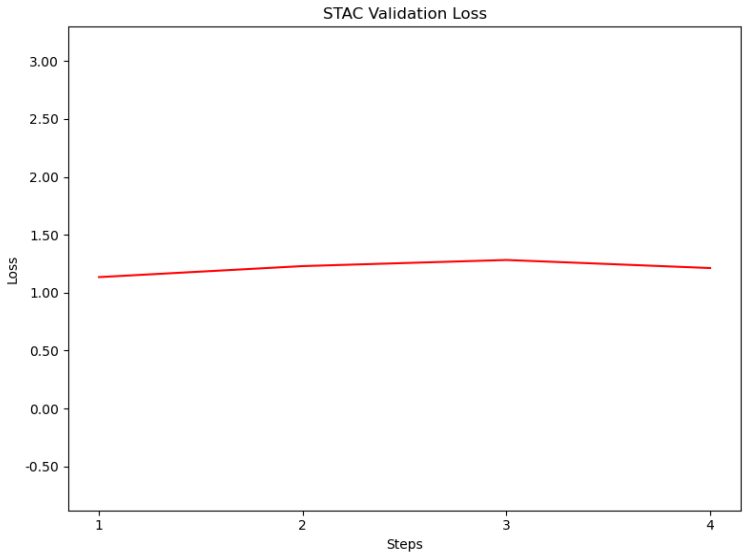


Figura 4.12: Grafico della loss in fase di validation con GraphSAGE per il dataset Molweni

I valori ottenuti in merito alle metriche di valutazione considerate sono migliori in fase di testing e questo evidenzia una buona capacità del modello di generalizzare a nuovi dati, purché superiori di poco rispetto a quelli della validazione. I valori più alti sono per le metriche di recall e accuracy, ma abbiamo valori più bassi per la precision, dimostrando che il modello genera anche una parte di falsi positivi (data dalla recall alta). Le prestazioni però sono comunque da ritenersi buone perché i valori ottenuti non sono eccessivamente bassi.

	Accuracy	**Precision**	**Recall**
Validation	70,9%	67,94%	79,43%
Testing	72,86%	69,73%	80,8%

Tabella 4.10: Metriche in fase di Validation e Testing per Molweni con la GraphSAGE

4.2 Link Predictor tramite pipeline

Il task di Data Augmentation effettuato eseguendo la fase di Link Prediction all'interno della pipeline viene condotto distintamente utilizzando le due architetture GNN implementate. Quindi vengono analizzati i risultati ottenuti valutando il dataset aumentato prodotto mediante l'utilizzo sia di GAT che di GraphSAGE.

4.2.1 Generazione Dati con GPT-4o mini

Come descritto in 3.3.2, vengono generate 3 EDU distinti mediante l'utilizzo del modello GPT-4o mini. Di seguito è riportato un esempio in cui, data una EDU target, vengono generate tre EDU sostitutive. Di seguito viene fornito un esempio di quanto fatto per ogni nodo target di ogni dialogo del dataset considerato.

EDU Target: starting one block up and over from the yellow block

- **Nuova EDU generata 1**: Place a yellow block at that position.
- **Nuova EDU generata 2**: Place a yellow block between the endpoints of the two farthest orange rows from you.
- **Nuova EDU generata 3**: Place a yellow block in the designated area.

4.2.2 Task di Link Prediction: Scelta della EDU

Tramite le due GNN addestrate sul task di Link Prediction, viene scelta una delle 3 EDU generate, in base al criterio definito nella Sezione 3.4.3. Tale task viene condotto quindi utilizzando distintamente la GNN di tipo GAT e di tipo GraphSAGE.

4.2.3 Costruzione dataset aumentato

L'utilizzo delle due tecniche proposte porterà a dataset aumentati di uguale dimensione, perché nella tecnica utilizzata la scelta è stata quella di aggiungere, per ogni dialogo del training set, un dialogo aggiuntivo in cui la EDU target è sostituita dalla EDU generata scelta, mantenendo però valide tutte le relative relazioni e attributi. Nel dettaglio, considerando che il dataset iniziale di training di Minecraft contiene 307 dialoghi, il dataset aumentato con i dati generati conterrà 614 dialoghi.

Le differenze tra i due dataset aumentati derivano dalle differenti prestazioni dei due Link Predictor utilizzati, il primo basato su GAT e l'altro su GraphSAGE.

Maggiori valutazioni sull'effettiva efficacia e sul miglioramento della qualità del dataset aumentato saranno effettuate nella fase successiva, durante la quale confron-

teremo le performance ottenute dal modello Seq2Seq rispettivamente sul dataset iniziale e sui due dataset aumentati prodotti.

4.3 Valutazione dell'efficacia

A seguito dell'esecuzione della pipeline sul dataset di Minecraft e del conseguente ottenimento dei due relativi dataset aumentati, procediamo ad analizzare i risultati ottenuti utilizzando tali dataset con il modello esistente Seq2Seq-DDP. Di seguito vengono mostrati i grafici indicanti l'andamento della loss durante la fase di fine-tuning del modello Seq2Seq-DDP, utilizzando i 3 dataset di Minecraft sopra citati. Il numero di epoche per la fase di training è pari a 5, valore pre-impostato nello script utilizzato.

La Figura 4.13 mostra la loss per il dataset di training iniziale di Minecraft. Notiamo un andamento decrescente della loss all'aumentare delle epoche, prova del fatto che il modello in fase di training riesce ad apprendere bene, con valori poco al di sotto di 1,5.

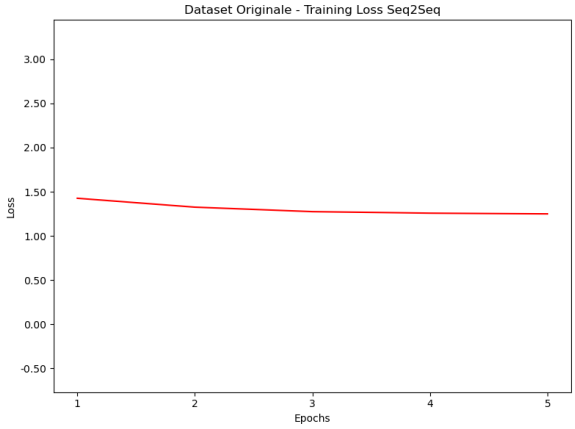


Figura 4.13: Training Loss per il dataset originale di Minecraft

I valori di loss ottenuti durante la fase di training condotta sul dataset di Minecraft aumentato utilizzando l'architettura GAT vengono mostrati in Figura 4.14. Abbiamo anche in tal caso un andamento decrescente e valori di poco superiori ad 1.

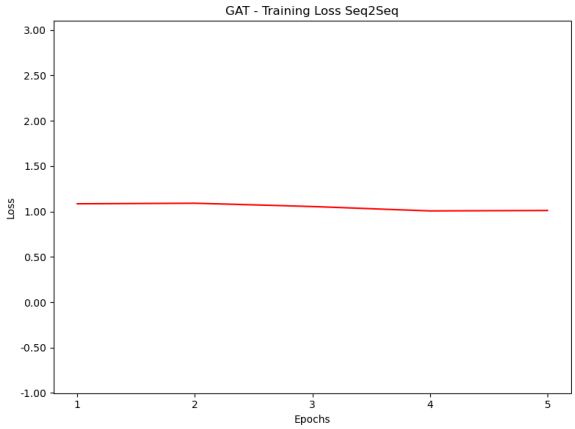


Figura 4.14: Training Loss per il dataset di Minecraft aumentato con GAT

Il training condotto utilizzando il set di dati di Minecraft aumentato mediante l'utilizzo di GraphSAGE ha prodotto i risultati mostrati nella Figura 4.15. I valori di loss ottenuti sono tutti vicini a 1, scendendo anche di poco al di sotto di tale valore e inoltre l'andamento della curva di loss è decrescente.

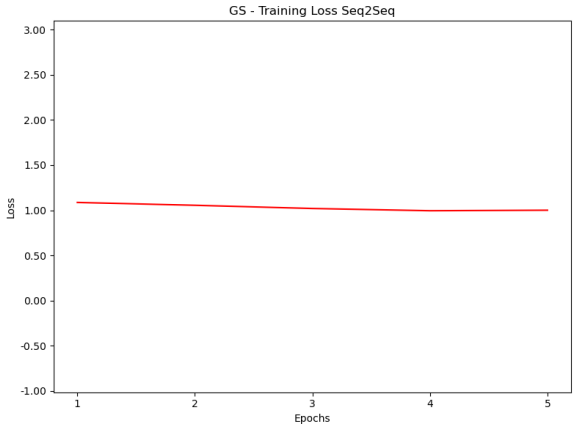


Figura 4.15: Training Loss per il dataset di Minecraft aumentato con GraphSAGE

Pertanto, analizzando le performance ottenute con i dataset considerati, è possibile evidenziare due importanti confronti:

- **Dataset Originale vs. Dataset Aumentati**: in termini di training loss, otteniamo valori inferiori e quindi prestazioni del modello migliori utilizzando i dataset di Minecraft aumentati rispetto a quello iniziale;
- **Dataset Aumentato con GAT vs. Dataset Aumentato con GraphSAGE**: anche avendo valori di training loss bassi in entrambi i casi, possiamo affermare che il

training condotto utilizzando il dataset aumentato con GraphSAGE riesce ad ottenere una perdita minore.

Una volta ottenuti i modelli pre-addestrati, il testing si è concentrato sul valutare le performance di test in termini di **F1-score**. A tal proposito, all'interno dei metodi implementati, tale metrica è stata calcolata considerando due misure fondamentali. La prima è **linkonly**, che nella metrica considera solo la predizione o meno corretta dell'arco, senza tener conto anche della relazione esistente tra i due nodi coinvolti, come, invece, effettuato da **link+rel**. Considerando l'importanza delle relazioni tra le EDU, abbiamo quindi scelto di considerare i valori di F1-score consideranti link+rel.

La fase di testing è stata condotta utilizzando in tutti e tre i casi il test set, facendo variare il modello pre-addestrato. Quindi vengono considerati i risultati della metrica F1-score ottenuti rispettivamente utilizzando il modello ottenuto dalla fase di training sul dataset originale, sul dataset aumentato con GAT e sul dataset aumentato con GraphSAGE.

Un ulteriore confronto riportato in tabella viene fatto riguardo la metrica F1-score calcolata sui dati **Raw** e **Post**, rappresentanti rispettivamente il set di test prima e dopo una fase di post-processing dei dati. Al fine di gestire eventuali errori generati da Seq2Seq-DDP (es.EDU mancanti o allucinazioni), la fase di post-processing applica regole di rifinitura producendo dati processati.

La Tabella 4.11 mostra i valori di F1-score ottenuti per i tre modelli considerati: il modello addestrato sul training set originale e i due modelli addestrati sui training set aumentati. I risultati indicano che, sebbene la F1-score rimanga complessivamente bassa, si registra un lieve miglioramento con l'uso dei dataset aumentati rispetto al dataset originale. Inoltre, tra i modelli analizzati, quello basato su GraphSAGE ottiene le prestazioni migliori, seppur molto basse.

Training Model	Type	F1-Score
Dataset originale	Raw	0.27%
	Post	0.23%
Dataset aumentato tramite GAT	Raw	1.01%
	Post	0.89%
Dataset aumentato tramite GraphSAGE	Raw	1.16%
	Post	1.02%

Tabella 4.11: Risultati F1-Score per il task Seq2Seq

Vengono condotte analisi più approfondite sui valori della metrica F1-score, con un focus specifico sui tipi di relazione. Questo passaggio permette di esaminare, per una determinata relazione, quali relazioni il modello ha erroneamente assegnato e con quale frequenza. Le Figure 4.16, 4.17, 4.18 mostrano, per ciascuno dei tre modelli considerati, la matrice di confusione che confronta le label con quelle generate dal modello.

Le righe della matrice fanno riferimento alle label, mentre le colonne si riferiscono alle predizioni dei modelli.

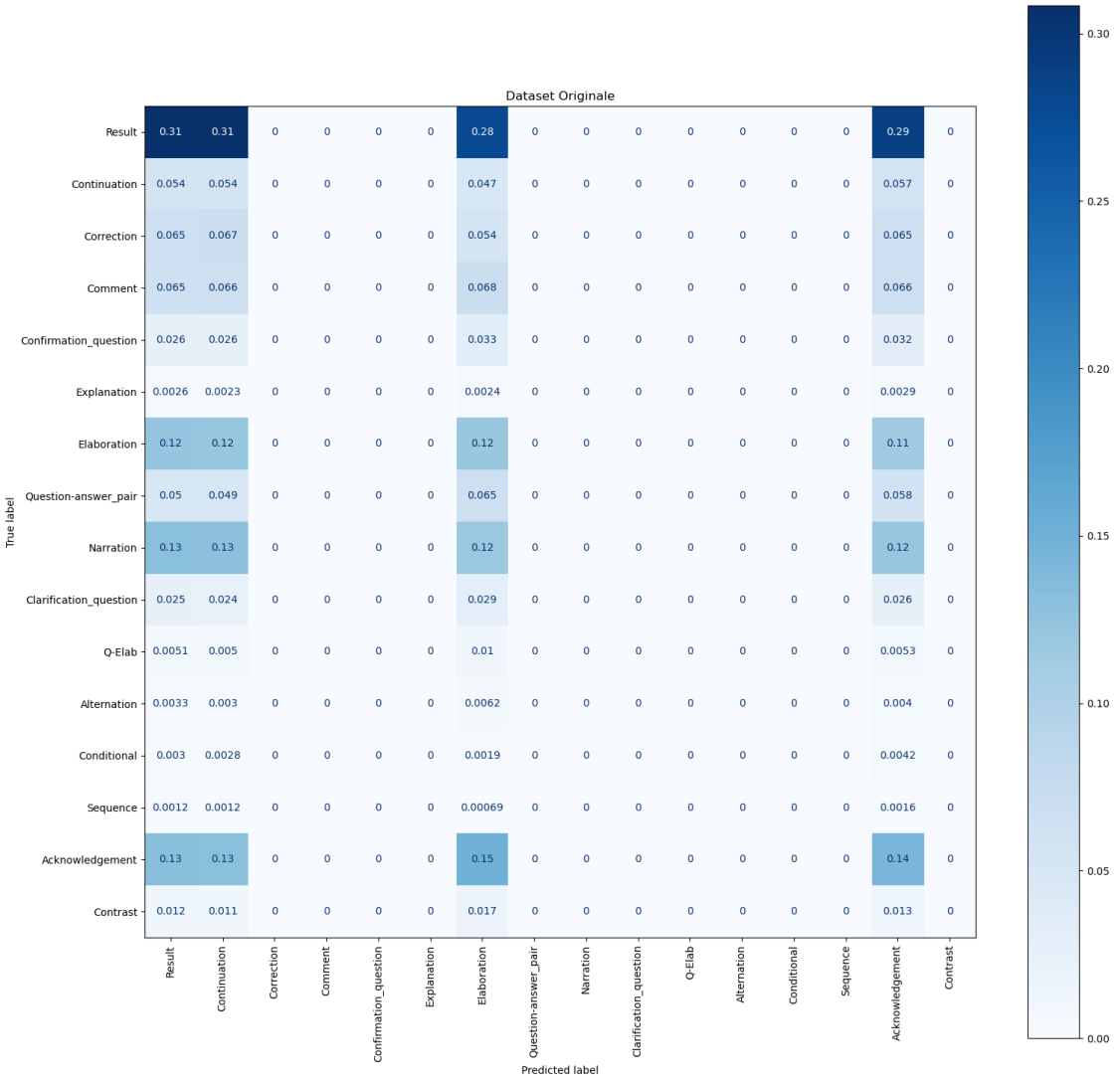


Figura 4.16: Matrice di confusione ottenuta per le predizioni del modello addestrato sul dataset originale

La matrice di confusione ottenuta per le predizioni del modello addestrato sul dataset naturale (Figura 4.16) evidenzia che il modello ha predetto sempre e soltanto le relazioni Result, Continuation, Elaboration, Acknowledgement, ignorando le restanti. Le performance migliori sono state ottenute sulla relazione **Result**, che risulta essere la più frequente nel training set e nel test set di Minecraft. Ciò è indicativo del fatto che il modello è riuscito ad apprendere le caratteristiche della relazione grazie alla possibilità di osservare un maggior numero di campioni.

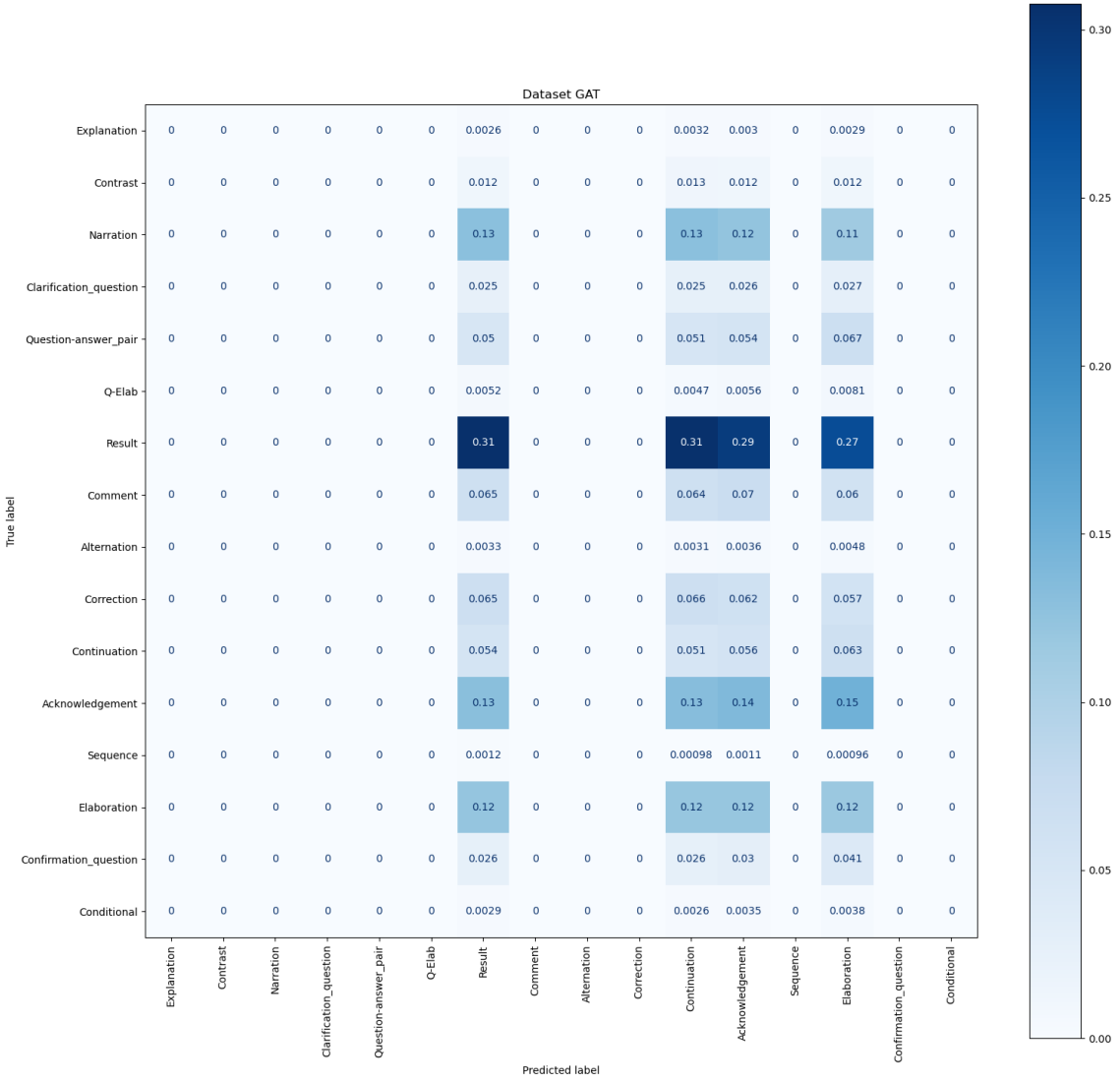


Figura 4.17: Matrice di confusione ottenuta per le predizioni del modello addestrato sul dataset aumentato con GAT

La matrice di confusione ottenuta per le predizioni del modello addestrato sul dataset aumentato con GAT (Figura 4.17) evidenzia, come prima, che il modello ha predetto sempre e soltanto quattro relazioni. In questo caso, si tratta delle relazioni di Result, Continuation, Acknowledgement, Elaboration. Anche in questo caso, le performance migliori sono state ottenute per la relazione Result, il che consente di trarre le stesse conclusioni di prima.

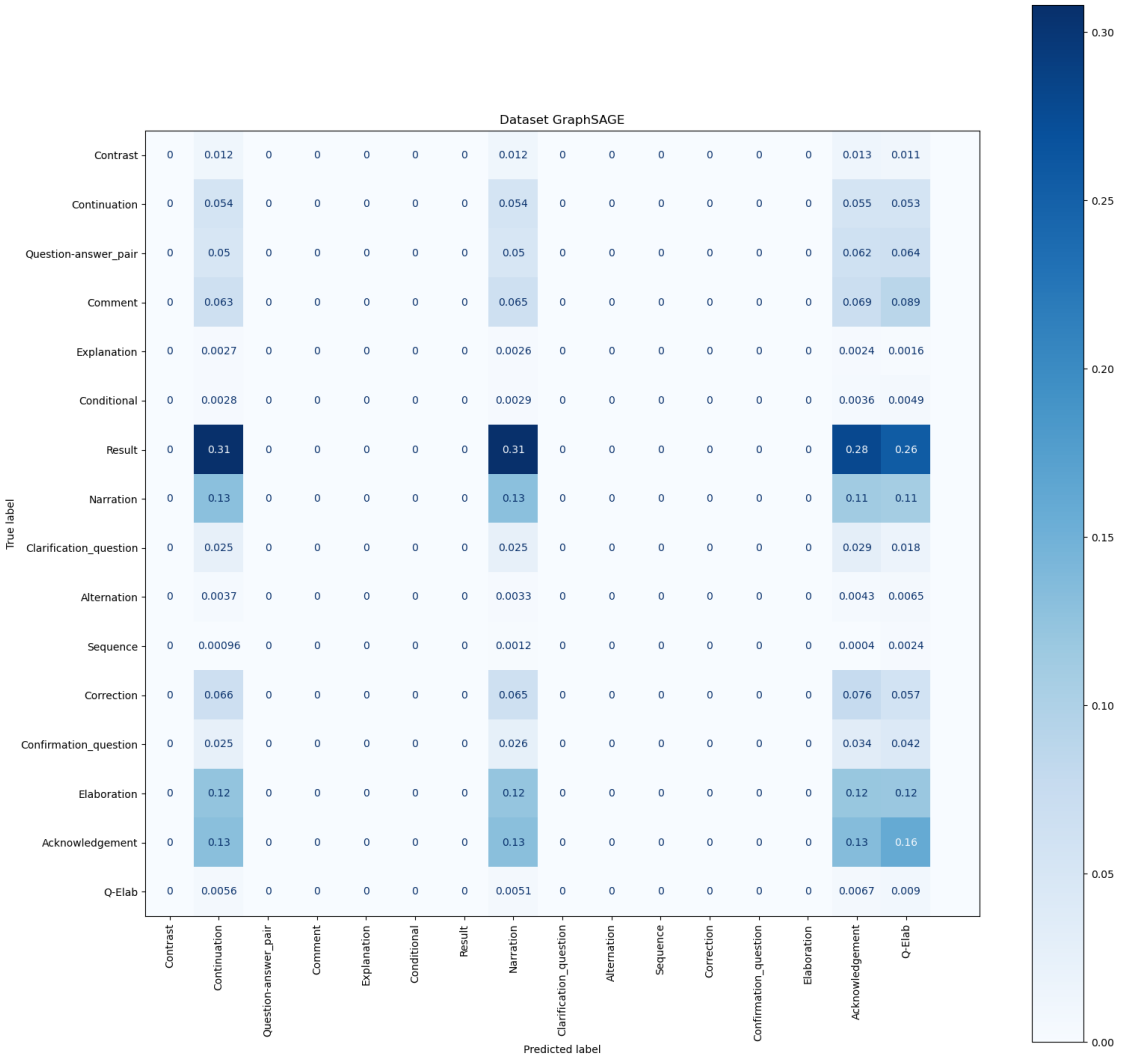


Figura 4.18: Matrice di confusione ottenuta per le predizioni del modello addestrato sul dataset aumentato con GS

La matrice di confusione ottenuta per le predizioni del modello addestrato sul dataset aumentato con GS (Figura 4.18) riporta dei risultati simili. In questo caso, le quattro relazioni predette sono Continuation, Narration, Acknowledgement, Q-Elab.

In definitiva, si può osservare che il predire frequentemente relazioni di tipo Result (la più frequente del training set) è un comportamento comune a tutti i modelli, in particolare per i primi due (addestrati rispettivamente sul dataset naturale e quello aumentato con GAT). Il terzo modello, invece, tende a generalizzare e a predire

Result meno frequentemente. Ciò motiva il fatto che quest'ultimo modello è risultato il migliore, in termini di **training loss** e di **F1-score**.

CAPITOLO 5

Conclusioni e sviluppi futuri

Questo studio si è focalizzato sullo sviluppo e l'implementazione di una pipeline innovativa per la Data Augmentation di dialoghi multi-parte, combinando modelli di linguaggio avanzati (LLM) con tecniche di validazione basate su Graph Neural Networks (GNN). L'obiettivo principale del progetto è stato quello di creare dati sintetici che rispettassero rigorosamente i vincoli semantici e strutturali presenti nei dataset originali, garantendo al contempo un miglioramento delle performance nel task di parsing del modello Seq2Seq-DDP. Nello specifico, il modello addestrato sul dataset aumentato con GraphSAGE è risultato il migliore, in quanto ha dimostrato una migliore capacità di generalizzazione rispetto ai restanti due. In generale, i risultati ottenuti, come evidenziato nel Capitolo 4, confermano la robustezza e l'efficacia dell'approccio adottato, offrendo al contempo numerosi spunti per ulteriori approfondimenti e ottimizzazioni future.

5.1 Sviluppi futuri

Per quanto riguarda gli sviluppi futuri, un primo passo cruciale consiste nell'estendere l'esecuzione della pipeline anche ai dataset STAC e Molweni, per i quali l'attuale implementazione si è fermata alla fase di addestramento della GNN e del

relativo Link Predictor. Per raggiungere questo obiettivo, è necessario disporre di un accesso esteso alle API di OpenAI utilizzate, attraverso una versione a pagamento o un piano che consenta un numero sufficiente di richieste. Una volta superato questo vincolo, sarà possibile generare dataset aumentati anche per STAC e Molweni e confrontare le prestazioni ottenute rispetto ai dataset originali.

Inoltre, un altro aspetto rilevante potrebbe essere l'adozione di strategie di ottimizzazione della pipeline, al fine di ridurre i costi computazionali e incrementarne la scalabilità, rendendola applicabile a dataset di dimensioni maggiori o con caratteristiche più complesse. Infine, ulteriori sperimentazioni su dataset provenienti da domini differenti potrebbero contribuire a validare ulteriormente la generalizzabilità e l'efficacia del framework sviluppato.

Bibliografia

[1] H. Chen, Z. Dou, K. Mao, J. Liu, and Z. Zhao, "Generalizing conversational dense retrieval via llm-cognition data augmentation," *arXiv preprint arXiv:2402.07092*, 2024. (Citato a pagina 4)

[2] G. Cimino, C. Li, G. Carenini, and V. Deufemia, "Coherence-based dialogue discourse structure extraction using open-source large language models," in *Proceedings of the 25th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2024, pp. 297–316. (Citato a pagina 5)

[3] N. Asher, V. Popescu, P. Muller, S. Afantenos, A. Cadilhac, F. Benamara, L. Vieu, and P. Denis, "Manual for the analysis of settlers data," *Strategic Conversation (STAC). Université Paul Sabatier*, 2012. (Citato a pagina 9)

[4] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023. (Citato a pagina 18)

[5] C. Li, Y. Yin, and G. Carenini, "Dialogue discourse parsing as generation: A sequence-to-sequence llm-based approach," in *Proceedings of the 25th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2024, pp. 1–14. (Citato a pagina 33)