



# UNIVERSITÀ DI PISA

Corso di Laurea Magistrale in Informatica Umanistica

## **Dante Similarity Search: un'applicazione web per ritrovare Dante nelle testimonianze dei sopravvissuti al Lager**

**Relatore:**

Prof.ssa Marina Riccucci

**Secondo relatore:**

Prof. Angelo Mario Del Grosso

**Candidata:**

Carla Congiu



# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 Il progetto <i>Voci dall'Inferno</i></b>	<b>2</b>
<b>2 Stato dell'arte</b>	<b>8</b>
2.1 <i>Verse Similarity for Multi-lingual Representations of the Qur'an</i> . . .	10
2.2 <i>Vulgate Verse Similarity Search</i> . . . . .	11
<b>3 Strumenti</b>	<b>13</b>
3.1 Linguaggio di programmazione Python . . . . .	13
3.2 Ambiente di sviluppo Google Colab . . . . .	14
3.3 Libreria spaCy . . . . .	15
3.4 Sentence Transformers . . . . .	15
3.4.1 LaBSE . . . . .	16
3.4.2 Paraphrase Mpnet Base V2 . . . . .	17
3.4.3 All Mpnet Base V2 . . . . .	18
3.5 Database vettoriale Weaviate . . . . .	18
3.6 Framework sviluppo web app Streamlit . . . . .	20
3.7 Sistema di deploy Cloudflare Tunnel . . . . .	21
3.8 Sistema di tracciamento GitHub . . . . .	22
<b>4 Creazione dei dataset dalla Commedia</b>	<b>23</b>
4.1 Segmentazione in terzine . . . . .	25
4.2 Segmentazione in versi . . . . .	27
4.3 Segmentazione in frasi . . . . .	28
<b>5 <i>Dante Similarity Search</i>: workflow</b>	<b>31</b>
5.1 Preparazione dei dati e creazione degli <i>embeddings</i> . . . . .	32
5.2 Connessione con Weaviate . . . . .	34
5.3 Utilizzo dei dati vettorializzati . . . . .	37
5.4 Creazione di un tunnel Cloudflare . . . . .	40
5.5 Esecuzione dell'applicazione Streamlit . . . . .	41
5.6 Streamlit web application . . . . .	42
5.7 Interfaccia grafica e interazione utente . . . . .	48

<b>6</b>	<b>Valutazione e discussione dell'approccio</b>	<b>52</b>
6.1	Citazioni esplicite . . . . .	54
6.2	Citazioni implicite . . . . .	62
<b>7</b>	<b>Integrazione dell'applicazione nel processo di codifica dei dati</b>	<b>67</b>
	<b>Conclusioni e sviluppi futuri</b>	<b>70</b>
	<b>Bibliografia</b>	<b>72</b>
	<b>Sitografia</b>	<b>75</b>

# Elenco delle figure

3.1	Interfaccia di Google Colab . . . . .	14
3.2	Architettura di Weaviate . . . . .	19
3.3	Funzionamento di Streamlit . . . . .	20
3.4	Architettura di Cloudflare tunnel . . . . .	21
3.5	Interfaccia di GitHub . . . . .	22
4.1	Codifica XML della Commedia . . . . .	24
5.1	Diagramma del workflow . . . . .	31
5.2	Interfaccia dell'applicazione Dante Similarity Search . . . . .	48
5.3	Menù di selezione della cantica . . . . .	49
5.4	Menù di selezione del canto . . . . .	49
5.5	Menù di selezione della metrica . . . . .	49
5.6	Esempio di ricerca di un'espressione esatta . . . . .	50
5.7	Esempio di ricerca di un'espressione non esatta . . . . .	51
6.1	Risultati ottenuti utilizzando LaBSE . . . . .	53
6.2	Risultati ottenuti utilizzando Paraphrase Mpnet Base V2: il riferi- mento viene colto nel quinto risultato . . . . .	53
6.3	Grafico Scatterplot che mostra i punteggi di similarità per ogni metrica	61
6.4	Esempio di ricerca con ChatGPT . . . . .	66

## Elenco delle tabelle

6.1	«Grandine grossa, acqua tinta e neve per l'aer tenebroso si riversa;» EA	54
6.2	«Le ripe eran grommate d'una muffa per l'alito di giù che vi s'appasta che con gli occhi e col naso facea zuffa» LG	55
6.3	«contraddizion che nol consente» LG	55
6.4	«La notte ch'io passai con tanta pièta» LG	56
6.5	«Temp'era dal principio del mattino e 'l sol che montava 'n sù con quelle stelle ch'eran con lui quando l'amor divino mosse di prima quelle cose belle;» LG	56
6.6	«"Miserere di me" gridai a lui "qual tu sii od ombra od omo certo?"» LG	57
6.7	«Lo giorno se n'andava, e l'aere bruno toglieva gli animai che sono in terra dalle fatiche loro» LG	57
6.8	«la bella scola» LG	58
6.9	«dentro di sé con la sua rabbia» LG	58
6.10	«E caddi come corpo morto cade.» NR	59
6.11	«male altrui» LS	59
6.12	«Dolce colore d'oriental zaffiro» LG	60
6.13	«fuor delle braccia del suo dolce amico» LG	60
6.14	«di spirito profetico dotato» LG	61
6.15	«Almeno mi pareva d'esser come Ciacco stasera, pochi minuti fa, quando tornavo in albergo dopo aver camminato per ore al buio, perdendomi e disperdendomi per la campagna, sotto una pioggia continua, affondando nel fango e nella fanghiglia» EA	62
6.16	«stare sempre agitato come selvaggina che può essere colta di sorpre- sa.» EA	63
6.17	«l'altro sonava di continuo la tromba di Barbariccia.» EA	64

# Listings

4.1	Dataset delle terzine . . . . .	25
4.2	Dataset dei versi . . . . .	27
4.3	Dataset delle frasi . . . . .	29
5.1	Preparazione dei dati e creazione degli embeddings . . . . .	32
5.2	Connessione con Weaviate . . . . .	34
5.3	Inserimento dei vettori . . . . .	38
5.4	Creazione di un tunnel Cloudflare . . . . .	40
5.5	Lancio dell'applicazione Streamlit . . . . .	41
5.6	Script dell'applicazione Streamlit . . . . .	43
7.1	Trasposizione dei risultati nelle codifiche delle testimonianze . . . . .	68





# Introduzione

Il presente lavoro di tesi è parte delle attività di sviluppo incardinate nell'ambito del progetto di ricerca dell'Università di Pisa *Voci dall'Inferno*, con l'obiettivo di creare un'applicazione web per il riconoscimento automatico di richiami al lessico dantesco all'interno delle testimonianze rese da sopravvissuti ai campi di concentramento.

Lo strumento software, denominato *Dante Similarity Search*, è progettato per individuare citazioni e allusioni al lessico dantesco presenti in tali testimonianze e per condurre un confronto tra gli enunciati dei sopravvissuti e i versi della *Commedia* di Dante Alighieri.

L'applicazione, realizzata impiegando il linguaggio di programmazione Python, utilizza diverse tecnologie di rappresentazione e gestione vettoriale dei dati testuali come *Weaviate*, una piattaforma open-source per la conservazione, la ricerca e la restituzione del testo, e *Streamlit*, un framework per lo sviluppo di applicazioni web perfettamente integrato in workflow di analisi del testo. Basandosi su metriche di *Sentence Similarity*, l'applicazione sfrutta modelli numerici per ottenere *embeddings* contestuali di frammenti testuali e in seguito misurarne la similarità.

# Capitolo 1

## Il progetto *Voci dall'Inferno*

*Voci dall'Inferno* è un progetto di ricerca dell'Università di Pisa avviato nel 2016 e coordinato dalla professoressa Marina Riccucci del Dipartimento di Filologia, Letteratura e Linguistica dell'Università di Pisa, con il supporto del professor Angelo Mario Del Grosso dell'Istituto di Linguistica Computazionale "A.Zampolli" del CNR di Pisa. L'iniziativa ha due principali obiettivi. Il primo mira a digitalizzare un ampio corpus di testimonianze non letterarie di deportati sopravvissuti ai campi di concentramento; il secondo obiettivo ambisce a identificare all'interno del repertorio la presenza di citazioni e/o allusioni al lessico di Dante.

Il corpus ad oggi consta di 23 testimonianze appartenenti a 18 testimoni, distinti in 17 deportati nei Lager (12 ebrei e 5 internati militari italiani) e 1 perseguitato. Esse sono pervenute sia in forma scritta (diari, memoriali, lettere) sia in forma orale (interviste conservate in microcassette e supporti non digitali), la maggior parte delle quali inedite da trattare.

Dall'analisi del corpus è emerso che molti testimoni si servono del vocabolario dantesco per poter esprimere le proprie esperienze concentrazionarie, altrimenti "*ineffabili*" (Riccucci, Calderini, 2020). Per definizione degli stessi superstiti, infatti, il Lager è indicibile: quindi portarne testimonianza equivale a narrare l'inesprimibile. Coloro che hanno voluto provare a raccontare la loro esperienza si sono ritrovati impotenti in quanto hanno riscontrato un limite linguistico dovuto alla mancanza di parole per esprimere cos'è il Lager. Fino a quando, a un certo punto, l'impasse è crollata, le parole sono uscite dalla mente, dalle labbra e dalle penne (della maggior

parte) dei sopravvissuti e il ‘dovere della testimonianza’ ha trovato la via, il canale del riferire. E seppure i modi, le forme e i toni siano differenziati, il minimo comun denominatore è uno solo: la definizione-descrizione del campo di concentramento come di un ‘*inferno*’, un inferno in terra. E quell’inferno è proprio l’inferno raccontato da Dante. Coloro che hanno paragonato il campo di concentramento all’Inferno dantesco, adottando le parole del poeta per descriverlo, lo hanno fatto non per un’esigenza estetica o per un’ambizione letteraria, ma per l’urgenza di trovare un mezzo di comunicazione. Tutti i sopravvissuti che hanno deciso di raccontare il Lager denunciano l’insufficienza della propria capacità espressiva per riuscire a descrivere una realtà che non somigliava a niente di cui si fosse mai stata fatta esperienza umana. Ed è proprio qui che la *Commedia* – e nel caso specifico la prima cantica – svolge un ruolo decisivo: quello di fornire il vocabolario per dare un nome a eventi e a realtà per le quali la lingua non aveva (o sembrava non avere) lemmi, concedendo così al vissuto delle vittime di farsi parola. Nello stesso tempo, noi ‘lettori’/‘uditori’ abbiamo potuto capire quello che le testimonianze riferivano grazie, proprio, alle parole dantesche: abbiamo potuto comprendere ciò che altrimenti non poteva essere compreso.

È interessante notare che la presenza di lemmi danteschi caratterizza le testimonianze sul vissuto concentrazionario a prescindere dall’età, dal genere, dal livello di istruzione (per molti di loro l’unico bagaglio culturale è quello fornito dalla scolarizzazione di base) e dalla nazionalità dei testimoni, anche se, in un contesto italiano, il semplice richiamo alla parola ‘inferno’ è già intriso della materia dantesca. Il patrimonio lemmatico collettivo, infatti, era fatto di parole dantesche ormai entrate nell’uso, nella vita di tutti i giorni, penetrato nella lingua del quotidiano, dentro il parlare della famiglia e della società, trasmesso di generazione in generazione come un’eredità. L’immaginario infernale di Dante rappresenta quindi una fonte da cui trarre ispirazione per individuare un linguaggio, le parole necessarie, per dire il Lager.

Questa connessione tra l’Inferno dantesco e il Lager nazista ha portato a una sovrapposizione semantica dei due termini, “inferno” e “Lager”, divenuti ormai quasi sinonimi. Si parla di nesso Lager-Inferno. Secondo la ricerca di Pisa il nesso è nuovo

e attiene all'età contemporanea: prima non esisteva, semplicemente perché prima il Lager non esisteva. Ciò che si è creata è dunque una simmetria tra il luogo immaginario di Dante e quello reale dei campi, entrambi trasmessi attraverso la scrittura. Le tessere dantesche rinvenute nelle testimonianze del corpus di Voci dall'Inferno si sono presentate attraverso diverse forme che sono state puntualmente classificate in:

- citazioni, sia in forma implicita sia in forma esplicita, di determinati versi della *Commedia*;
- allusioni o riferimenti più generici all'opera;
- vocaboli danteschi.

Dei 18 testimoni che costituiscono il corpus, 11 fanno uso di citazioni, allusioni, o termini danteschi provenienti dalla *Commedia*, per un totale di 68 tessere dantesche. Di queste, 15 sono citazioni esplicite, 4 sono citazioni implicite, 10 sono allusioni e 39 sono vocaboli danteschi. I dati che emergono dall'archivio digitale di Voci dall'Inferno confermano quanto osservato da Federico Siragusa nel suo lavoro di tesi<sup>1</sup>, ovvero che la maggior parte delle citazioni e delle allusioni presenti all'interno delle testimonianze, sono riconducibili alla prima cantica, la quale presenta infatti 16 citazioni su 19 totali. Le rimanenti rivelano che anche le cantiche del Purgatorio e del Paradiso sono state citate dai testimoni, seppur in misura minore: il primo 2 volte, il secondo 1 volta.

Come illustra Elvira Mercatanti nel suo lavoro di tesi<sup>2</sup>, tra i testimoni che ricorrono maggiormente a citazioni dantesche, c'è Luigi Giuntini, che nel suo diario cita a memoria in maniera esplicita la *Commedia* per ben 12 volte. All'epoca dei fatti narrati Giuntini aveva solo la licenza elementare. Nonostante avesse interrotto gli studi, si era appassionato alla letteratura e aveva studiato da autodidatta dal 1934 fino al suo arruolamento. Tra i suoi autori prediletti c'era proprio Dante, dal quale Giuntini ha spesso preso in prestito le parole per riuscire a esprimere ciò che stava vivendo. Uno dei canti citati da Giuntini è, ad esempio, il Canto I dell'Inferno, di

---

<sup>1</sup>F. Siragusa, I luoghi danteschi della memoria dei sopravvissuti

<sup>2</sup>E. Mercatanti, Le voci dei sopravvissuti ai Lager: l'applicazione web per l'analisi, per l'interrogazione e per la fruizione delle testimonianze

cui Luigi, nel passo del diario riportato di seguito, cita il verso 21, e i versi 37-40 e 65-66:

Mi soffermo con grande interesse sul primo Canto dell'Inferno perché a me pare denso di allegorie e di similitudini. [...] Dirò che, prima di essere chiamato alle armi, avevo letto gran parte del divino Poema, che il mio babbo comprava a dispense e che era illustrato con indubbio talento artistico da Gustavo Doré. Questa era la seconda o la terza volta che mi avvicinavo ai primi canti dell'Inferno e potevo confrontare le interpretazioni del testo precedenti, confrontandole con quelle attuali. Allora penso di poter affermare che, sceverando tra i 136 versi di questo complesso proemio, almeno una decina di essi colpivano di nuovo la mia sensibile fantasia e, cioè, i seguenti: «*La notte ch'io passai con tanta pietà*», «*Temp'era dal principio del mattino e 'l sol montava in su con quelle stelle ch'eran con lui quando l'amor divino mosse di prima quelle cose belle*», «*“Miserere di me” gridai a lui “qual tu sii, od ombra od omo certo?”*» [...].

È sempre Giuntini a citare in forma esplicita anche alcuni versi del Purgatorio e del Paradiso. Per descrivere nel suo diario l'azzurro cielo estivo privo di nuvole, si serve del verso 12 del canto I della seconda cantica:

Stamani, guardando il cielo, potrei finalmente descriverlo come padre Dante: «*Dolce colore d'oriental zaffiro*» tanto è azzurro e senza una nuvola.

Del Paradiso, invece, riprende il verso 141 del canto XII:

Mentre ci auguriamo che ciò non accada, intorno al nostro tavolo, nel primo pomeriggio si aduna, si fa per dire, “la bella scola” di coloro che per ragioni diverse, sono interessati a discutere di letteratura, di filosofia e di storia e, cioè, lo studente pugliese e quello umbro, Riccardi, Nicolato, Breazzano, Basso, un calabrese, che sa molte cose su Gioacchino da Fiore «*di spirito profetico dotato*», De Marzi, un napoletano, innamoratissimo di Salvatore Di Giacomo.

Le citazioni implicite individuate nelle testimonianze del corpus provengono principalmente da Emanuele Artom, che nel suo diario si riferisce unicamente alla prima cantica. Interessante, per esempio, è il rimando a Inferno VI 10-12:

Almeno mi pareva di essere come Ciacco stasera, pochi minuti fa, quando tornavo in albergo dopo aver camminato per ore al buio, perdendomi e disperdendomi per la campagna, sotto una pioggia continua, affondando nel fango e nella fanghiglia [...].

Le 10 allusioni individuate provengono tutte dalla prima cantica della *Commedia*: l'*Inferno*. Un esempio di allusione è presente nell'intervista rilasciata da Goti Bauer a Marina Riccucci il 7 marzo 2020:

Era pieno il campo e noi siamo arrivati in una bolgia, dove tutti gli altri sapevano tutto e ci informavano. Era una cosa da infarto, venire a sapere una cosa così nel giro di un minuto.

Un'altra allusione si ritrova in un passo del diario di Nicola Ricci:

Se il divino poeta, appostato su una di quelle colline, ci avesse potuto vedere passare, ci avrebbe certamente scambiato per qualche stuolo di quelle anime perse, da lui collocate in qualche cerchio dell'inferno, che costantemente girano portando il fardello dei loro peccati. Stimolati da feroci mastini, visi pallidi, stracciati, emaciati, camminiamo.

Interessante è anche ciò che Romana Feld riferisce della sua esperienza concentrataria, non solo definendo come un inferno il luogo in cui si trova, ma anche testimoniando la difficoltà del sopravvissuto nel trovare le parole per dire il Lager:

È impossibile con le povere parole ridare quadro esatto dell'inferno vissuto.

Anche i 39 vocaboli danteschi rinvenuti nelle testimonianze rimandano alla prima cantica. Il termine che ricorre in assoluto più di tutti è proprio *inferno*, che si presenta per ben 21 volte. Gli altri vocaboli individuati sono:

- *dante*, che occorre 4 volte;

- *bolgia*, *virgilio* e *divino poeta* che occorrono 2 volte;
- *Infernale*, *fiere*, *divina commedia*, *anime dannate*, *purgatorio*, *anime*, *cerchio dell'inferno* e *peccati* che occorrono 1 volta.

Lo studio delle testimonianze all'interno del progetto Voci dall'Inferno mira ad analizzare il lessico e le allusioni dantesche presenti nel corpus, focalizzandosi più di altro sull'adozione del linguaggio dell'Inferno della *Commedia* da parte dei sopravvissuti per descrivere le esperienze nei Lager.

Data la vasta latitudine del progetto, è stato necessario coinvolgere metodi e tecniche informatiche e fare della ricerca anche un lavoro collettivo, che ha visto e che vede la collaborazione e il contributo di numerosi studenti dell'Università di Pisa.

Il progetto Voci dall'Inferno ha attraversato nella sua evoluzione tre fasi principali:

1. Sviluppo di una banca dati per la gestione dell'anagrafica delle testimonianze – l'archivio ha preso il nome di *memoria-archivio*;
2. Creazione, digitalizzazione e codifica del corpus delle testimonianze attraverso l'utilizzo dello standard XML-TEI;
3. Sviluppo, mediante la piattaforma eXist-db, di un'applicazione web per la fruizione e l'interrogazione dei dati conservati nell'archivio digitale: l'ambiente web permette di confrontare il lessico della testimonianza con il testo della *Commedia* dantesca da un lato mediante indicizzazione di forme testuali e lemmatizzazione delle stesse, dall'altro mediante confronti di stringhe implementati con politiche di *fuzzy matching* e similarità.

Con l'applicazione web *Dante Similarity Search* si intende raggiungere un ulteriore obiettivo: identificare in modo **automatico** la presenza di citazioni e/o allusioni al lessico di Dante all'interno del repertorio delle testimonianze.

## Capitolo 2

### Stato dell'arte

Il concetto che sta alla base dell'applicazione *Dante Similarity Search* è quello della *Sentence Similarity*, ovvero la tecnica utilizzata per misurare il grado di somiglianza (o similarità) tra due frasi. Per calcolare la similarità vengono utilizzati particolari algoritmi, chiamati modelli, progettati per valutare quanto due frasi sono semanticamente vicine. Questi modelli trasformano le frasi in vettori numerici chiamati *embedding*, che catturano le informazioni semantiche dei dati testuali. Una volta ottenuti gli embedding, il grado di similarità viene calcolato utilizzando particolari metriche, come ad esempio la distanza coseno o euclidea. Gli embedding, a seconda del modello utilizzato per ottenerli, possono essere statici o contestuali. Gli embedding statici utilizzano rappresentazioni vettoriali fisse per le parole, senza considerare il contesto in cui si trovano. La similarità viene dunque calcolata aggregando gli embedding delle parole, con risultati non soddisfacenti. Modelli che producono embedding statici sono Word2Vec (Mikolov et al., 2013)<sup>1</sup>, GloVe (Pennington et al., 2014)<sup>2</sup>, FastText (Joulin et al., 2016)<sup>3</sup>. Gli embedding contestuali, al contrario, considerano il contesto in cui le parole appaiono, generando per ogni parola un vettore diverso per ogni contesto. Modelli che producono embedding con-

---

<sup>1</sup>T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient Estimation of Word Representations in Vector. <https://arxiv.org/abs/1301.3781>

<sup>2</sup>C. Pennington, R. Socher, C. D. Manning, GloVe: Global Vectors for Word representation. <https://nlp.stanford.edu/pubs/glove.pdf>

<sup>3</sup>A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of Tricks for Efficient Text Classification. <https://arxiv.org/abs/1607.01759>



testuali sono BERT (Devlin et al., 2019)<sup>4</sup> e SBERT (Reimers, Gurevych, 2019)<sup>5</sup>. I modelli BERT (Bidirectional Encoder Representations from Transformers) sono modelli di apprendimento automatico basati sull'architettura Transformer<sup>6</sup> e progettati per l'elaborazione del linguaggio naturale. BERT ha rappresentato un punto di svolta rispetto ai modelli tradizionali che elaborano il testo in modo unidirezionale (da sinistra a destra o viceversa). BERT, infatti, utilizza un approccio bidirezionale per analizzare le relazioni tra le parole in una frase, considerando sia il contesto precedente sia quello successivo. Il modello BERT viene addestrato su una grande quantità di dati linguistici (circa 110 milioni di parametri e oltre 3 miliardi di token) mediante due attività non supervisionate<sup>7</sup>: il Masked Language Model (MLM), in cui il 15% delle parole in input vengono nascoste in modo casuale affinché il modello impari a prevederle; la Next Sentence Prediction (NSP), in cui il modello impara a capire se una frase è la continuazione logica della precedente o meno. Per farlo, analizza coppie di frasi (correlate o casuali) e cerca di prevedere la relazione tra di esse. Questo aiuta BERT a sviluppare una comprensione più profonda del contesto e delle relazioni tra le frasi. Successivamente BERT può essere ulteriormente affinato e migliorato mediante il processo di fine-tuning, che consiste nell'adattamento e ottimizzazione del modello pre-addestrato per eseguire compiti specifici. Il modello SBERT (SentenceTransformers) è una versione ottimizzata di BERT, orientata specificamente al confronto di frasi in modo più efficiente e al calcolo della similarità. Per calcolare la Sentence Similarity esistono anche altri metodi come le ontologie o i Large Language Models. Tuttavia, possono risultare computazionalmente costosi e richiedono una buona quantità di dati per essere perfezionati su domini specifici. Per il presente progetto di tesi, i SentenceTransformers si sono dimostrati efficaci nel conseguire l'obiettivo stabilito.

---

<sup>4</sup>J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. <https://arxiv.org/pdf/1810.04805>

<sup>5</sup>N. Reimers, I. Gurevych, Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. <https://arxiv.org/pdf/1908.10084>

<sup>6</sup>Il Transformer è una rete neurale avanzata che elabora il linguaggio in parallelo grazie al meccanismo di self-attention, permettendogli di cogliere meglio le relazioni tra le parole. Questo lo rende efficace per attività come traduzione automatica e generazione di testo ed è alla base di molti modelli di intelligenza artificiale moderni. Per approfondire: Vaswani et al., Attention is all you need (2017). <https://arxiv.org/pdf/1706.03762>

<sup>7</sup><https://web.stanford.edu/~jurafsky/slp3/11.pdf>

Numerosi sono i progetti che hanno come idea di base la Sentence Similarity e utilizzano le tecnologie di embedding e modelli BERT. Di seguito vengono illustrate iniziative analoghe all'applicazione *Dante Similarity Search*.

## 2.1 *Verse Similarity for Multi-lingual Representations of the Qur'an*

I ricercatori dell'Università della Georgia (USA) A. Basharat, D. Yazdansepas e K. Rasheed nel 2015 hanno presentato uno studio sul problema della similarità testuale nelle diverse rappresentazioni multilingue del Corano, concentrandosi su testi in arabo e inglese e valutando diverse misure di similarità<sup>8</sup>. Il Corano, con le sue numerose traduzioni, rappresenta un caso ideale per analizzare la similarità tra versi in lingue diverse. L'obiettivo principale dello studio è confrontare diverse misure di similarità applicate a differenti rappresentazioni del testo coranico. Il database utilizzato nello studio include quattro rappresentazioni del testo coranico: testo arabo con diacritici, testo arabo senza diacritici, radici delle parole arabe e una traduzione inglese. Dopo la tokenizzazione dei testi, i versi vengono rappresentati come vettori nel modello dello spazio vettoriale e vengono utilizzati tre metodi di pesatura delle parole: booleano (1 se la parola è presente, 0 altrimenti); conteggio delle frequenze delle parole; TF-IDF (Term Frequency–Inverse Document Frequency), che è la misura basata sulla frequenza relativa delle parole nei versi. Per confrontare i versi, lo studio utilizza quattro misure di similarità: distanza euclidea, similarità coseno, similarità Jaccard e coefficiente di correlazione di Pearson. Gli esperimenti hanno evidenziato che: il dataset del testo arabo senza diacritici è risultato il più accurato per identificare versetti identici; la misura Pearson con TF-IDF si è dimostrata particolarmente efficace, soprattutto per versetti con elevata similarità; le traduzioni inglesi spesso non catturano completamente le sfumature semantiche e morfologiche del testo arabo originale. Lo studio sottolinea l'importanza di combinare misure

---

<sup>8</sup>Comparative Study of Verse Similarity for Multi-lingual Representations of the Qur'an. <https://delaram.lmu.build/publication/basharat-2015-comparative/basharat-2015-comparative.pdf>

diverse per catturare meglio la similarità semantica, evidenziando la complessità del linguaggio e l'utilità di standardizzare queste analisi.

## 2.2 *Vulgate Verse Similarity Search*

Il ricercatore americano William Mattingly<sup>9</sup> nel 2022 ha messo a punto il progetto dal nome *Vulgata spaCy*<sup>10</sup>. Si tratta di una libreria costruita su spaCy per automatizzare l'identificazione e l'estrazione di potenziali citazioni bibliche nei testi latini medievali. La versione della *Vulgata* – ossia la traduzione in latino della Bibbia dall'antica versione greca ed ebraica, realizzata alla fine del IV secolo da Sofronio Eusebio Girolamo – utilizzata è la versione Clementina disponibile presso il *Clementine Text Project*<sup>11</sup>. Ai fini della realizzazione della libreria è stata ripulita da segni di interpunzione ed è stata strutturata in un file CSV. La pipeline principale sfrutta i Bloom embeddings, che sono stati addestrati sull'intera Patrologia Latina<sup>12</sup> (PL) dopo una sostanziale pulizia. Questi embeddings sono stati caricati in una pipeline spaCy come *floret embeddings*<sup>13</sup> (vettori più leggeri e robusti). Ciò consente alla pipeline di avere dimensioni ridotte, pur essendo in grado di catturare un significato semantico e sintattico profondo. La pipeline contiene due componenti: un *Entity-Ruler*, i cui modelli possono identificare citazioni dirette o parziali delle Scritture con o senza segni di punteggiatura; un modello di apprendimento automatico per il rilevamento delle citazioni, progettato per identificare le citazioni all'interno del contesto. Le virgolette e altri indicatori sono stati rimossi prima dell'addestramento, in modo che il modello possa apprendere le caratteristiche e il contesto delle citazioni latine. Entrambi questi componenti segnalano le potenziali o probabili citazioni scritturali. La seconda fase del processo consiste nel collegare la citazione a uno specifico versetto della Vulgata. Questo aspetto presenta però alcuni problemi. In primo luogo, i testi latini non hanno un'ortografia standard (anche nelle edizioni moderne); in secondo luogo, i testi latini moderni non concordano sulla punteggiatura.

---

<sup>9</sup><https://www.wjbmattlingly.com/>

<sup>10</sup><https://github.com/wjbmattlingly/vulgata-spacy>

<sup>11</sup><https://sourceforge.net/projects/vulsearch/>

<sup>12</sup><https://patristica.net/latina/>

<sup>13</sup><https://explosion.ai/blog/floret-vectors>

tura; in terzo luogo, nel Medioevo circolavano molte versioni della Bibbia che non erano conformi alla Vulgata standard, il che significa che le parole potevano apparire fuori ordine o che venivano usati dei sinonimi. Pertanto, per collegare correttamente i dati, è necessario mantenere il significato semantico e sintattico. Oltre a questi problemi, molte citazioni delle Scritture nei testi medievali sono riferimenti parziali. Per ovviare a questo problema, ogni riga della Scrittura è stata incorporata insieme ai suoi componenti di frase. Questi sono poi stati mappati alla riga specifica della Scrittura. Per ridurre i falsi positivi, una frase deve essere superiore a 3 parole per essere considerata idonea.

Nel 2024 Mattingly ha elaborato una nuova versione di questo progetto, chiamata *Weaviate Vulgate*. La tecnologia che sta alla base, come suggerisce il nome, è Weaviate, un database vettoriale open-source per combinare la ricerca vettoriale con il filtraggio strutturato. Weaviate viene utilizzato per indicizzare il testo della Vulgata, permettendo ricerche basate sul significato piuttosto che su semplici corrispondenze di parole. Questa nuova versione del progetto mira a creare un motore di ricerca semantico che consenta agli utenti di esplorare i testi biblici in latino in modo efficiente e intuitivo. Lo sviluppo dello strumento si articola in tre fasi: i versi biblici vengono convertiti in rappresentazioni vettoriali mediante il modello di SentenceTransformers LaBSE; i dati ottenuti vengono inseriti nel database vettoriale; infine, mediante l'utilizzo del framework Streamlit, viene creata un'interfaccia web interattiva che permette di cercare frammenti di testo e individuare i versi della Vulgata più simili.

Weaviate Vulgate è il software da cui prende le mosse lo sviluppo dell'applicazione *Dante Similarity Search*.

# Capitolo 3

## Strumenti

La realizzazione dell'applicazione *Dante Similarity Search* ha richiesto, nelle varie fasi, l'utilizzo di molteplici strumenti. Nello specifico: come linguaggio di programmazione è stato scelto Python, particolarmente adatto alla realizzazione di applicazioni di elaborazione automatica del testo e di sviluppo web; per la fase di scrittura del codice dello script sviluppato si è impiegato l'ambiente Google Colab; per la segmentazione del testo della *Commedia* è stata utilizzata la libreria per l'elaborazione del linguaggio naturale spaCy; per la rappresentazione dei dati in strutture vettoriali embeddings si è fatto ricorso a modelli di Sentence Transformers; per la memorizzazione e il recupero dei dati testuali è stata utilizzata la piattaforma Weaviate; per realizzare l'applicazione web è stato adottato il framework Streamlit; per la creazione di una demo dell'applicativo è stato adoperato il servizio di tunneling di Cloudflare; per la gestione e la conservazione di tutte le componenti del progetto si è sfruttata la piattaforma di hosting GitHub.

### 3.1 Linguaggio di programmazione Python

Python<sup>1</sup> è un linguaggio di programmazione di alto livello multipiattaforma, dinamico, orientato agli oggetti e utilizzabile per molti tipi di sviluppo software. Offre un forte supporto all'integrazione con altri linguaggi e programmi ed è fornito di una estesa libreria standard e di numerosi frameworks. Python è distribuito con licenza

---

<sup>1</sup><https://www.python.org/>

open-source e il suo utilizzo è gratuito e libero anche per prodotti commerciali. Tra le sue molteplici applicazioni c'è l'accesso ai database, la lettura e scrittura di file, la creazione di interfacce grafiche, l'accesso al sistema operativo per operazioni su file e cartelle.

## 3.2 Ambiente di sviluppo Google Colab

Google Colab<sup>2</sup> è una piattaforma basata sull'architettura tecnologica cloud che consente di scrivere e eseguire codice Python attraverso l'interfaccia web fornita dall'ambiente Google. È un ambiente di sviluppo integrato (IDE) che offre risorse computazionali gratuite insieme a strumenti per la scrittura, l'esecuzione e la condivisione di codice Python. Google Colab presenta diversi vantaggi: offre gratuitamente l'accesso a risorse di calcolo potenti che possono accelerare l'esecuzione di calcoli intensivi; fornisce un ambiente Python e numerose librerie di supporto, evitando così configurazioni in locale; permette di condividere facilmente i notebook con altri utenti, consentendo loro di visualizzare e modificare il codice in modo collaborativo.

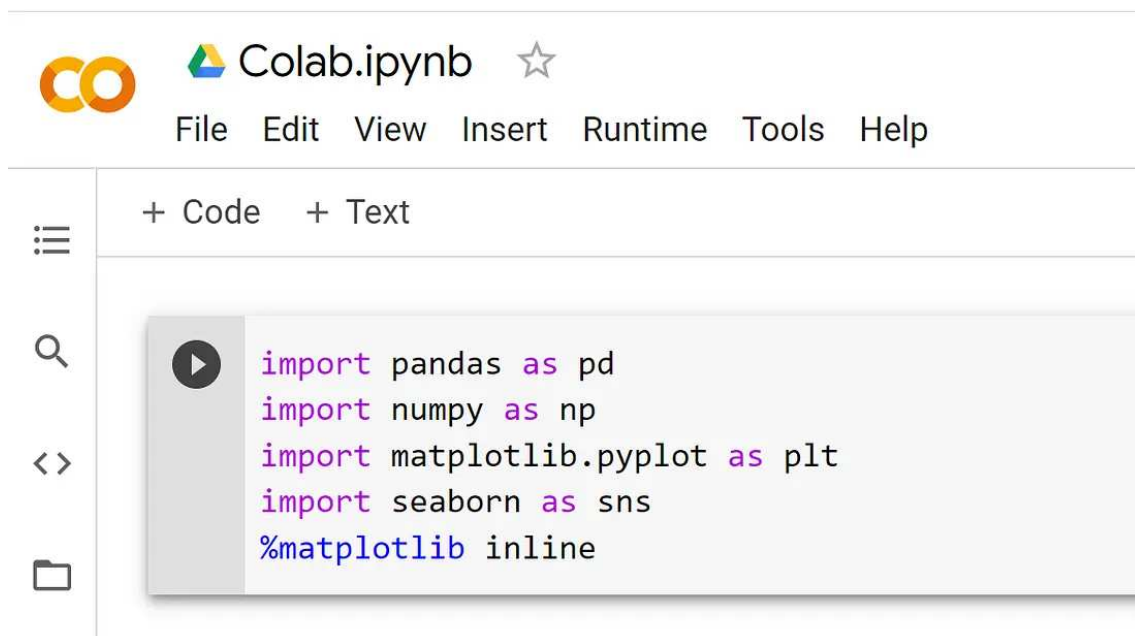


Figura 3.1: Interfaccia di Google Colab

---

<sup>2</sup><https://colab.research.google.com/>

### 3.3 Libreria spaCy

spaCy<sup>3</sup> è una libreria open-source di elaborazione del linguaggio naturale (NLP) in Python. È progettata per essere veloce, efficiente e facile da usare, ed è particolarmente adatta per applicazioni che richiedono analisi avanzate del testo, come chatbot, motori di ricerca e classificazione dei documenti. Può essere utilizzata per creare sistemi di estrazione di informazioni o di comprensione del linguaggio naturale, o per pre-elaborare testi per l'apprendimento profondo. Le caratteristiche principali di spaCy sono: Tokenizzazione avanzata, per dividere il testo in parole, frasi o sottounità; POS tagging, per l'assegnazione di categorie grammaticali (sostantivo, verbo, aggettivo, ecc.); Named Entity Recognition (NER), per riconoscere entità come nomi di persone, luoghi, aziende, date, ecc.; Dependency Parsing, per analizzare relazioni tra parole in una frase; Word Vectors, ossia supporta modelli con embedding semantici per calcolare la similarità tra parole e frasi; include modelli pre-addestrati per diverse lingue; è di facile integrazione con il Machine Learning.

### 3.4 Sentence Transformers

Sentence Transformers, noto anche come SBERT<sup>4</sup>, è il pacchetto Python di riferimento per l'accesso, l'utilizzo e l'addestramento di modelli di embedding di testo e immagini allo stato dell'arte. Questi modelli sono progettati per compiti di ricerca semantica, confronto tra testi e riconoscimento di parafrasi in ampie collezioni di documenti. L'obiettivo principale di Sentence Transformers è creare rappresentazioni semantiche di frasi, paragrafi o testi attraverso la generazione di vettori, detti embedding, che catturano il significato del contenuto testuale. Ciò permette di confrontare e misurare la somiglianza tra frasi o documenti in modo efficiente. Il pacchetto offre funzionalità avanzate, tra cui il calcolo della similarità tra testi, la ricerca semantica e la classificazione automatica del contenuto. Inoltre, garantisce il supporto multilingua grazie a modelli specifici per diverse lingue, inclusi quelli addestrati su dati multilingue, e assicura un'elevata efficienza, consentendo il calcolo

---

<sup>3</sup><https://spacy.io/>

<sup>4</sup><https://sbert.net/>

rapido degli embedding anche su grandi volumi di dati. Attualmente, sono disponibili oltre 12.000 modelli Sentence Transformers pre-addestrati su Hugging Face<sup>5</sup>, di cui più di 9.000 specifici per il task di Sentence Similarity. Il pacchetto consente inoltre di addestrare o perfezionare modelli personalizzati in modo semplice, adattandoli a esigenze specifiche. Nel contesto del progetto di tesi, la scelta del modello di embedding è cruciale per garantire risultati accurati e coerenti nell'analisi della similarità tra testi. Per questo motivo, sono stati testati tre diversi modelli di Sentence Transformers: *LaBSE*, *paraphrase-mpnet-base-v2* e *all-mpnet-base-v2*. Questi modelli sono stati selezionati per la loro capacità di gestire dati multilingue, catturare relazioni semantiche complesse e offrire un buon compromesso tra accuratezza ed efficienza computazionale.

### 3.4.1 LaBSE

LaBSE<sup>6</sup> (Language-Agnostic BERT Sentence Embedding) è un modello di sentence embedding multilingue sviluppato da Google AI. È progettato per generare rappresentazioni vettoriali di frasi indipendenti dalla lingua, mantenendo una struttura semantica coerente tra lingue diverse. Questo lo rende particolarmente utile per attività di retrieval cross-lingua, similarità semantica, traduzione automatica e clustering multilingue. LaBSE è un modello dual encoder<sup>7</sup>, basato su BERT-base (12 strati, 768 dimensioni). Ogni frase è convertita in un embedding indipendente dalla lingua, permettendo di confrontare frasi in lingue diverse in uno spazio vettoriale comune. Usa la funzione Additive Margin Softmax<sup>8</sup> per migliorare la separazione

---

<sup>5</sup><https://huggingface.co/>

<sup>6</sup><https://huggingface.co/sentence-transformers/LaBSE>

<sup>7</sup>Un dual encoder consiste in due reti neurali separate che codificano due diversi insiemi di dati nei rispettivi embeddings.

<sup>8</sup>La funzione Softmax è una funzione matematica che converte un vettore di numeri (chiamati logits), che rappresentano le "ipotesi grezze" provenienti dalla rete neurale, in probabilità. A ogni logit viene assegnato un valore compreso tra 0 e 1 e le probabilità ottenute sono distribuite in modo tale che la loro somma sia pari a 1. Softmax aiuta a trasformare i valori di output della rete neurale in un formato interpretabile come probabilità, rendendolo adatto per compiti di classificazione. La Additive Margin Softmax (AM-Softmax) è una variante della funzione Softmax standard: introduce un margine additivo tra le classi per migliorare la discriminazione tra di esse e rendere le decisioni della rete neurale più puntuali creando una separazione maggiore tra le diverse categorie. Questo margine forzato aiuta a ottenere embedding più distintivi e ben separati nello spazio, migliorando così la precisione del modello.



tra frasi simili e non e la Loss contrastiva<sup>9</sup> per ottimizzare il ranking delle frasi tradotte. Inoltre, il pretraining è stato fatto su oltre 6 miliardi di coppie di traduzioni per un migliore allineamento multilingue. Purtroppo LaBSE presenta anche alcune limitazioni: riesce a supportare 109 lingue ma non tutte con la stessa qualità; risulta molto pesante e richiede elevate risorse computazionali.

### 3.4.2 Paraphrase Mpnet Base V2

Paraphrase Mpnet Base V2<sup>10</sup> è un modello di sentence embeddings che converte frasi e paragrafi in vettori densi a 768 dimensioni sviluppato dal UKPLab<sup>11</sup> (Ubiquitous Knowledge Processing Lab). Questo processo facilita compiti come il clustering e la ricerca semantica, permettendo di confrontare facilmente il significato di diverse frasi. Paraphrase Mpnet Base V2 è basato su MPNet<sup>12</sup>, che consente di catturare in modo efficiente le relazioni semantiche tra le frasi. È adatto per attività come la text generation, la text classification, il sentence embeddings, la semantic textual similarity e l'identificazione di parafrasi. È ideale, dunque, per confrontare il significato di due frasi, trovare frasi simili in un ampio dataset e raggruppare frasi basate sul loro significato. Sebbene il modello sia molto efficace, presenta alcune limitazioni: ha una comprensione contestuale limitata, quindi potrebbe non cogliere completamente il contesto in cui una frase è stata scritta; potrebbe non comprendere implicazioni o conoscenze non esplicitamente presenti nei dati di addestramento; ha una conoscenza di dominio limitata, dunque se applicato a domini specifici non presenti nei dati di addestramento (da cui è dipendente), le prestazioni potrebbero diminuire.

---

<sup>9</sup>La loss contrastiva è una funzione di perdita usata per apprendere rappresentazioni di dati in uno spazio di embedding. È comunemente applicata per confrontare la similarità o dissimilarità tra vettori. Il suo scopo è quello di avvicinare istanze simili e di separare quelle dissimili all'interno dello spazio di embedding.

<sup>10</sup><https://huggingface.co/sentence-transformers/paraphrase-mpnet-base-v2>

<sup>11</sup><https://github.com/ukplab>

<sup>12</sup>MPNet adotta un metodo di pre-addestramento innovativo, chiamato masked and permuted language modeling, che combina i vantaggi di due tecniche di pre-addestramento: il masked language modeling, in cui alcuni token in una sequenza vengono nascosti (mascherati) e il modello impara a prevedere questi token utilizzando il contesto circostante; il permuted language modeling, che prevede i token in un ordine casuale, consentendo al modello di apprendere dipendenze bidirezionali (cioè il contesto precedente e successivo al token considerato) e migliorare la comprensione contestuale del linguaggio.

### 3.4.3 All Mpnet Base V2

All Mpnet Base V2<sup>13</sup> è un modello di sentence-transformers progettato per mappare frasi e paragrafi in uno spazio vettoriale denso a 768 dimensioni. Questo consente applicazioni come clustering, ricerca semantica e recupero di informazioni. Anche questo modello è basato su MPNet, dunque si presenta molto simile a `paraphrase-mpnet-base-v2`. È adatto per attività come recupero di informazioni, clustering e misurazione della similarità tra frasi. Il modello è stato addestrato su un vasto dataset di oltre 1 miliardo di coppie di frasi utilizzando un obiettivo di apprendimento contrastivo auto-supervisionato –tecnica di apprendimento automatico in cui un modello viene addestrato a distinguere tra coppie di rappresentazioni simili e dissimili senza la necessità di etichette manuali–, garantendo alta accuratezza ed efficienza. Anche All Mpnet Base V2 presenta alcune limitazioni: se il testo in input è più lungo di 384 token viene troncato, il che potrebbe portare a una perdita di informazioni semantiche in testi più lunghi; l'uso di una lunghezza di sequenza di 128 token limita la quantità di contesto considerato durante la codifica di una frase o paragrafo; infine, come qualsiasi modello di apprendimento automatico, potrebbe riflettere *bias* presenti nei dati di addestramento, restituendo rappresentazioni inaccurate di determinati gruppi o argomenti.

## 3.5 Database vettoriale Weaviate

Weaviate<sup>14</sup> è un vector database open source che consente di memorizzare e recuperare dati in base alle loro proprietà semantiche, indicizzandoli mediante una specifica rappresentazione vettorializzata. Offre numerose funzionalità tra cui la ricerca semantica, la gestione di compiti di estrazione di domande e risposte e la classificazione dei dati. Weaviate, inoltre, semplifica l'utilizzo di modelli di intelligenza artificiale allo stato dell'arte, offrendo al contempo la scalabilità, la facilità d'uso, la sicurezza e l'economicità di un database vettoriale appositamente costruito. In particolare

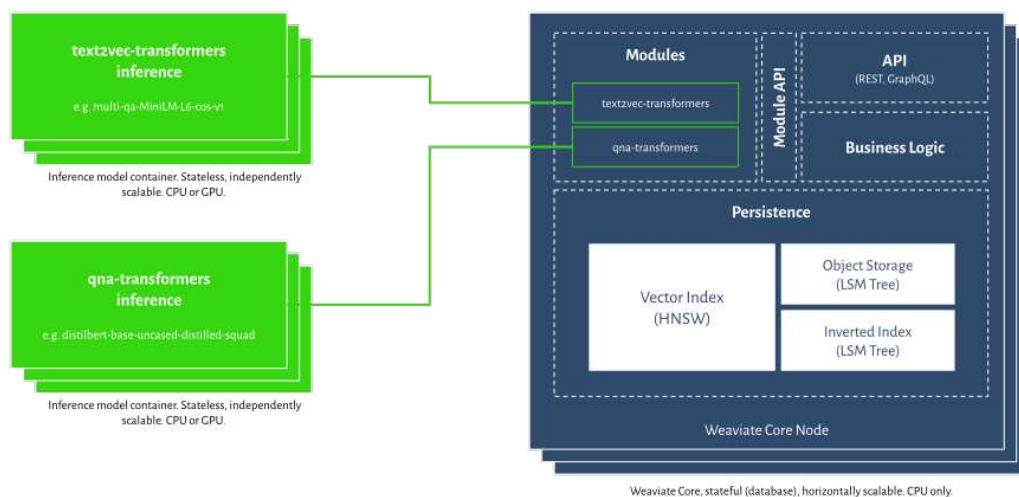
---

<sup>13</sup><https://huggingface.co/sentence-transformers/all-mpnet-base-v2>

<sup>14</sup><https://weaviate.io/>

Weaviate: esegue ricerche di nearest neighbor<sup>15</sup> (NN) su milioni di oggetti generalmente in un tempo notevolmente inferiore ai 100 ms; permette l'impiego di modelli AI allo stato dell'arte quali, per esempio, i Transformers, per accedere a dati di qualsiasi tipo (testo, immagini, ecc.) gestendone efficacemente il processo di vettorializzazione; consente di combinare in modo efficiente ricerche vettoriali e scalari e di effettuare ricerche nei dati anche se sono in fase di importazione o di aggiornamento. Un aspetto fondamentale di Weaviate è la sua architettura scalabile basata su cluster, che consente la distribuzione dei dati e delle query su più nodi. Grazie a questa struttura, è possibile gestire grandi volumi di informazioni garantendo alte prestazioni e affidabilità.

Weaviate System Level Overview (Example with two modules)



Two modules (text2vec-transformers, qna-transformers) shown as an example. Other modules include vectorization for other media types, entity recognition, spell checking and others.

Persistence in Weaviate Core shows one shard as an example. Users can create any number of indices, each index can contain any number of shards. Shards can be distributed and/or replicated across nodes in the cluster. A shard always contains object, inverted and vector storage. Vector storage is not affected by LSM segmentation.

Figura 3.2: Architettura di Weaviate

<sup>15</sup>L'algoritmo di nearest neighbor è un metodo utilizzato per classificare o trovare somiglianze nei dati, basandosi sulla distanza tra punti (vettori) in uno spazio multidimensionale. Dato un punto di interesse, l'algoritmo cerca i punti più vicini a esso in base a una misura di distanza (ad esempio la distanza euclidea o cosine).

## 3.6 Framework sviluppo web app Streamlit

Streamlit<sup>16</sup> è un framework gratuito e open-source per costruire e condividere rapidamente applicazioni web di machine learning e data science. Si tratta di una libreria basata sulle tecnologie dell’ecosistema Python, semplice nell’utilizzo ed efficace nella resa. Permette infatti di trasformare script Python in applicazioni web interattive in pochi passaggi, integrare widget interattivi come slider, pulsanti e caselle di controllo per migliorare l’esperienza utente, Visualizzare grafici, mappe e dati in tempo reale e distribuire facilmente le applicazioni sviluppate su piattaforme cloud. È compatibile con la maggior parte delle librerie Python e, tra le altre funzioni, consente l’interazione con i dati da parte di altri utenti, la rapida raccolta di feedback e l’istantaneo aggiornamento dell’applicazione non appena vengono apportate modifiche.

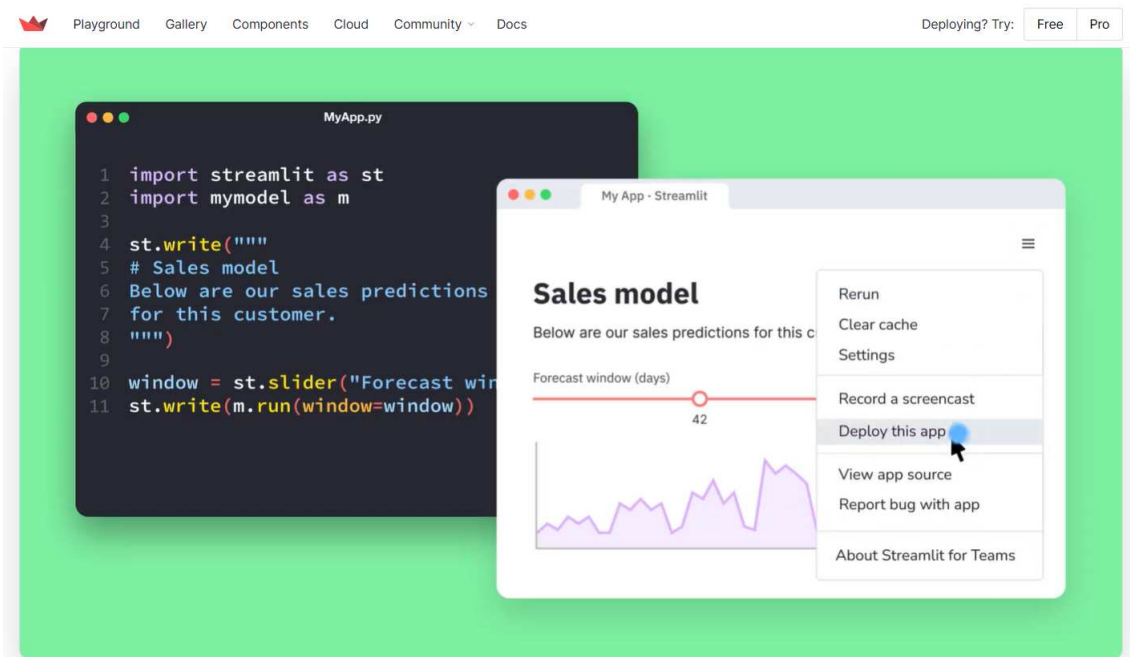


Figura 3.3: Funzionamento di Streamlit

<sup>16</sup><https://streamlit.io/>

## 3.7 Sistema di deploy Cloudflare Tunnel

Cloudflare Tunnel<sup>17</sup> è un servizio che consente di connettere in modo sicuro risorse e applicazioni da una infrastruttura alla rete globale di Cloudflare<sup>18</sup>, senza bisogno di un indirizzo IP pubblico. Funziona creando connessioni unidirezionali tramite un daemon leggero (programma in esecuzione in background) chiamato "cloudflared". Queste connessioni permettono di indirizzare il traffico verso Cloudflare, mantenendo al sicuro le origini da attacchi che bypassano i livelli di protezione di Cloudflare. Cloudflare Tunnel permette un accesso remoto sicuro a server e applicazioni locali durante le fasi di sviluppo, senza necessità di esporre l'infrastruttura a rischi esterni e, grazie al servizio TryCloudflare<sup>19</sup>, offre tunnel gratuiti e persistenti, ideali per testare e condividere applicazioni in modo semplice e protetto.

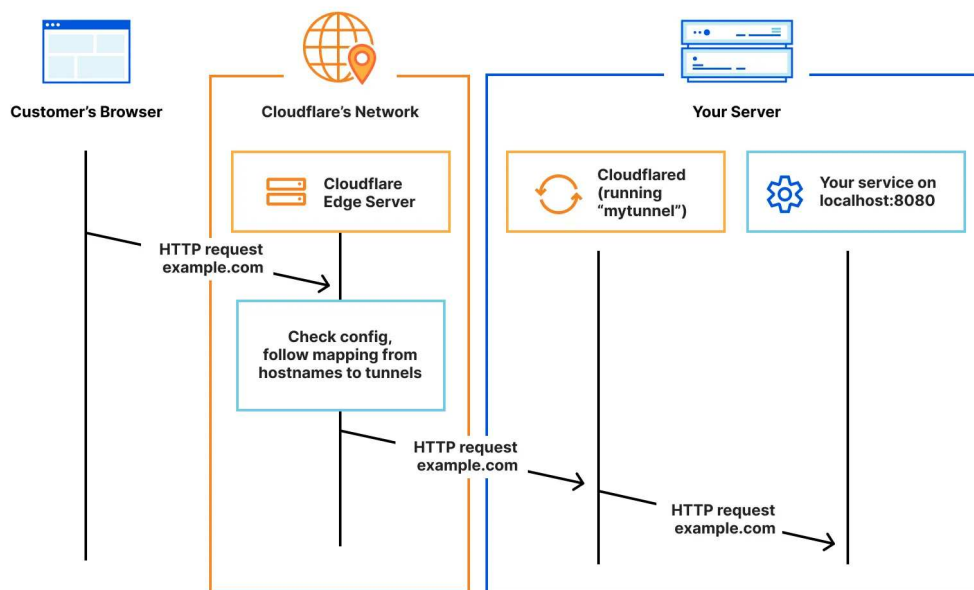


Figura 3.4: Architettura di Cloudflare tunnel

<sup>17</sup><https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/>

<sup>18</sup><https://www.cloudflare.com/it-it/>

<sup>19</sup><https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/do-more-with-tunnels/trycloudflare/>

## 3.8 Sistema di tracciamento GitHub

GitHub<sup>20</sup> è una piattaforma cloud-based in cui è possibile creare, memorizzare, condividere il proprio codice e collaborare con altri allo sviluppo di progetti software open-source. GitHub garantisce, inoltre, numerose funzionalità, come il controllo sugli accessi, il tracciamento degli errori, il monitoraggio delle modifiche al codice nel tempo, la gestione dei task e permette il lavoro collaborativo, reso possibile dal software open-source Git<sup>21</sup>, su cui GitHub è costruito.

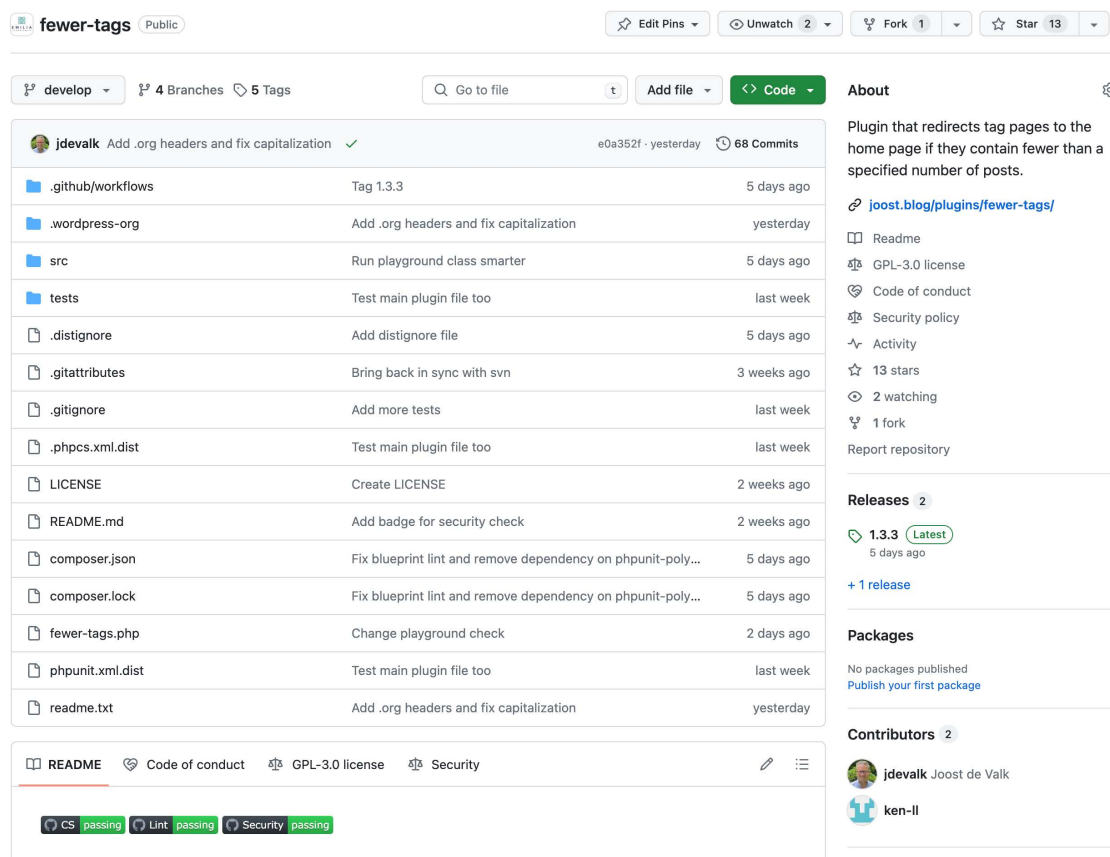


Figura 3.5: Interfaccia di GitHub

<sup>20</sup><https://github.com/>

<sup>21</sup><https://git-scm.com/>

## Capitolo 4

# Creazione dei dataset dalla Commedia

La *Commedia* di Dante Alighieri è suddivisa in tre cantiche, per un totale di cento canti. L’Inferno, la prima cantica, comprende 34 canti: il primo funge da proemio all’intera opera, mentre i restanti 33 sono dedicati alla descrizione dell’Inferno. Le altre due cantiche, Purgatorio e Paradiso, contano ciascuna 33 canti. L’intero poema è scritto in terzine incatenate di versi endecasillabi. Il numero di versi ammonta a 14.233, per un totale di 4.761 terzine e 101.698 parole.

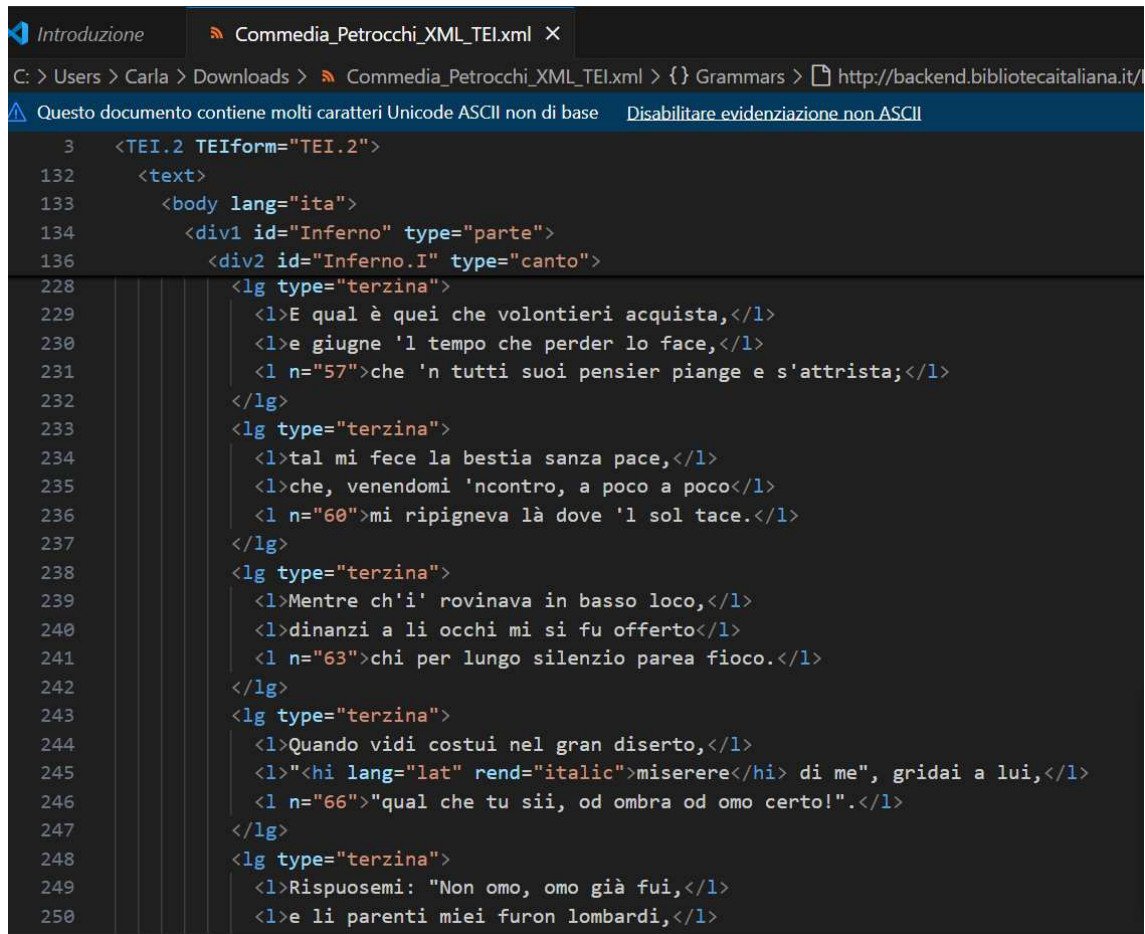
Per effettuare una comparazione accurata delle citazioni e allusioni presenti nelle testimonianze con i versi della *Commedia* si è optato per la creazione, mediante i DataFrame della libreria Pandas<sup>1</sup>, di tre diversi tipi di dataset in formato CSV, che suddividono il poema rispettivamente in terzine, versi e frasi. In questo modo è possibile individuare più precisamente il passo della *Commedia* a cui l’estratto della testimonianza fa riferimento. Per la realizzazione dei dataset sono stati implementati tre programmi Python, uno per ciascuna “metrica”. L’edizione della *Commedia* presa in esame per effettuare i confronti con gli enunciati dei sopravvisuti è quella curata da Giorgio Petrocchi del 1994<sup>2</sup>. Di questa edizione esiste una versione in formato digitale machine readable realizzata mediante lo standard XML-TEI ([Commedia-Petrocchi-XML-TEI](https://tinyurl.com/Commedia-Petrocchi)). Da essa sono stati estratti i dati testuali per

---

<sup>1</sup><https://pandas.pydata.org/>

<sup>2</sup><https://tinyurl.com/Commedia-Petrocchi>

la costruzione dei dataset. Per fare ciò si è adoperata libreria standard di Python `lxml`, specifica per l'elaborazione di file XML, e nello specifico il modulo `etree` per eseguire il parsing dei dati.



```
3 <TEI.2 TEIform="TEI.2">
132 <text>
133 <body lang="ita">
134 <div1 id="Inferno" type="parte">
136 <div2 id="Inferno.I" type="canto">
228 <lg type="terzina">
229 <l>E qual è quei che volontieri acquista,</l>
230 <l>e giugne 'l tempo che perder lo face,</l>
231 <l n="57">che 'n tutti suoi pensier piange e s'attrista;</l>
232 </lg>
233 <lg type="terzina">
234 <l>tal mi fece la bestia senza pace,</l>
235 <l>che, venendomi 'ncontro, a poco a poco</l>
236 <l n="60">mi ripigheva là dove 'l sol tace.</l>
237 </lg>
238 <lg type="terzina">
239 <l>Mentre ch'i' rovinava in basso loco,</l>
240 <l>dinanzi a li occhi mi si fu offerto</l>
241 <l n="63">chi per lungo silenzio parea fioco.</l>
242 </lg>
243 <lg type="terzina">
244 <l>Quando vidi costui nel gran diserto,</l>
245 <l><hi lang="lat" rend="italic">miserere</hi> di me", gridai a lui,</l>
246 <l n="66">"qual che tu sii, od ombra od omo certo!".</l>
247 </lg>
248 <lg type="terzina">
249 <l>Rispuosemi: "Non omo, omo già fui,</l>
250 <l>e li parenti miei furon lombardi,</l>
```

Figura 4.1: Codifica XML della Commedia



## 4.1 Segmentazione in terzine

Dopo aver individuato le cantiche (contenute negli elementi `div1`) e i canti (contenuti negli elementi `div2` con attributo `type="canto"`), viene estratto il titolo del canto (`h_canto`) e le terzine, ciascuna ottenuta dalla concatenazione e raggruppamento di tre versi contenuti nei tag `lg` come elementi `l` (righe 1-22). Viene poi calcolato il range di versi corrispondente a ogni terzina (ossia il numero del primo e dell'ultimo verso della terzina, es. vv. 1-3), con la distinzione dei versi che appaiono singolarmente (in genere l'ultimo verso di ogni canto)(righe 26-38).

```
1 # Estrae le cantiche dal file XML
2 cantiche = xml_data.xpath('//div1')
3 canti = {}
4
5 # Itera sulle cantiche estratte
6 for cantica_element in cantiche:
7     cantica = cantica_element.xpath('./@id')[0] # Recupera l'ID
           della cantica
8     tot_canti = cantica_element.xpath('./div2') # Estrae i canti
           contenuti nella cantica
9
10
11     for c in tot_canti:
12         h_canto = c.xpath('./head//text()') # Estrae il titolo del
           canto
13         terzine = []
14         # Estrae i versi e rimuove eventuali spazi indesiderati
15         terzina = ["".join(l.itertext()).strip() for l in c.xpath('
           ./l') if "".join(l.itertext()).strip()]
16
17         # Raggruppa i versi in terzine (gruppi di 3)
18         for i in range(0, len(terzina), 3):
19             terzine.append(terzina[i:i+3])
20
```

```
21 # Salva le terzine con la chiave composta da (cantica, titolo del
    canto)
22     canti[(cantica, h_canto[0])] = terzine
23
24 data = []
25 # Organizza i dati per la creazione del DataFrame
26 for (cantica, titolo), ter in canti.items():
27     i = 1 # Contatore dei versi
28     for t in ter:
29         if len(t) == 3: # Se e' una terzina completa
30             range_versi = f"{i}-{i+2}"
31             frase = " ".join(t) # Unisce i versi della terzina in
    una singola stringa
32             data.append([cantica, titolo, range_versi, frase])
33             i += 3
34         else: # Se rimangono versi isolati non appartenenti a una
    terzina completa
35             for verso in t:
36                 range_versi = f"{i}"
37                 data.append([cantica, titolo, range_versi, verso])
38                 i += 1
39
40 # Crea un DataFrame Pandas
41 df = pd.DataFrame(data, columns=['cantica', 'canto', 'range_versi',
    'terzina'])
42 # Aggiunge un carattere invisibile (zero-width space) al range dei
    versi per evitare problemi di formattazione
43 df['range_versi'] = "\u200B" + df['range_versi'].astype(str)
44 # Esporta i dati in un file CSV
45 df.to_csv("terzine.csv", index=False, sep=';', encoding='utf-8-sig',
    )
```

Listing 4.1: Dataset delle terzine

## 4.2 Segmentazione in versi

Partendo dalla prima parte della costruzione del dataset delle terzine (Listing 4.1, righe 1-22), i versi vengono estratti uno alla volta e il numero del verso (`n_verso`) segue il normale ordine crescente.

```
1 data = []
2
3 # Itera sui canti organizzati per (cantica, titolo) e sulle
   relative terzine
4
5 for (cantica, titolo), ter in canti.items():
6     i = 1 # Contatore per il numero di verso
7     for t in ter: # Itera sulle terzine
8         # Itera sui singoli versi all'interno di ogni terzina
9         for verso in t:
10            # Assegna il numero progressivo al verso
11            n_verso = f"{i}"
12            # Aggiunge il dato alla lista
13            data.append([cantica, titolo, n_verso, verso])
14            i += 1 # Incrementa il numero del verso
15
16 df = pd.DataFrame(data, columns=['cantica', 'canto', 'n_verso', '
   verso'])
17 df['n_verso'] = "\u200B" + df['n_verso'].astype(str)
18 df.to_csv("versi.csv", index=False, sep=';', encoding='utf-8-sig')
```

Listing 4.2: Dataset dei versi

### 4.3 Segmentazione in frasi

Per l'estrazione delle frasi – intese come proposizioni o periodi che finiscono in genere con un segno di interpunzione forte – è stata impiegata la libreria spaCy. Essa infatti fornisce una pipeline per l'elaborazione automatica del testo e l'analisi automatica della lingua anche per testi scritti in lingua italiana e permette il riconoscimento automatico e la suddivisione di un testo in *sentences*. Come nel caso delle terzine, anche qui è stata effettuata la concatenazione dei versi ma senza alcun raggruppamento (riga 16). La suddivisione in frasi è stata affidata, come detto sopra, a spaCy (riga 24), in particolare al modello `it_core_news_sm`, specifico per elaborare testi in lingua italiana (riga 4). Il calcolo del range di versi corrispondente approssimativamente alla posizione delle frasi è stato più complesso. Per determinare la posizione di ogni verso all'interno del testo intero del canto, il codice utilizza una variabile `cur_pos` che tiene traccia della posizione attuale nel testo. Per ogni verso, viene memorizzata la sua posizione iniziale e finale sotto forma di una coppia (`start`, `end`). Dopo aver aggiunto un verso, `cur_pos` viene aggiornato sommando la lunghezza del verso e aggiungendo 1 per simulare lo spazio tra i versi (righe 27-31). Ogni coppia (`start`, `end`) rappresenta quindi la posizione di un verso nel testo completo del canto. Una volta costruita questa mappatura dei versi, il codice esamina ogni frase individuata da spaCy per trovarne la posizione (`start_idx`, `end_idx`) nel testo intero del canto. Se la frase è troppo corta (meno di 2 caratteri), viene ignorata (righe 34-37). La funzione `testo_canto.find(frase_testo)` viene utilizzata per individuare l'inizio della frase nel testo, mentre la fine della frase viene calcolata aggiungendo la sua lunghezza a `start_idx` (righe 40-41). Successivamente, il codice verifica in quali versi compare la frase. Scorrendo tutti i versi con `enumerate(pos_versi)`, controlla se la posizione della frase si sovrappone con uno dei versi. Un verso viene considerato corrispondente se non termina prima che inizi la frase (`end < start_idx`) e non inizia dopo la fine della frase (`start > end_idx`). Poiché i versi della *Commedia* sono numerati a partire da 1, viene utilizzato `i + 1` per assegnare il numero corretto del verso (righe 44-46). Infine, il codice assegna un range di versi alla frase individuata. Se nessun verso corrisponde, viene stampato

un messaggio di errore e assegnato il valore “NA”. Se la frase si trova interamente in un solo verso, viene assegnato direttamente il numero di quel verso. Se invece la frase si estende su più versi, viene assegnato un intervallo costituito dal numero del primo verso in cui inizia la frase e il numero dell’ultimo verso in cui finisce la frase (es. vv. 13-18)(righe 48-55).

```
1 import spacy
2 # Caricamento del modello NLP di spaCy per la lingua italiana
3 nlp = spacy.load("it_core_news_sm")
4
5 canti = {}
6
7 # Estrazione delle informazioni sui canti dalla struttura XML
8 for cantica_element in cantiche:
9     cantica = cantica_element.xpath('./@id')[0] # Nome della
    cantica (Inferno, Purgatorio, Paradiso)
10     tot_canti = cantica_element.xpath('./div2') # Selezione dei
    canti all'interno della cantica
11
12     for c in tot_canti:
13         h_canto = c.xpath('./head//text()') # Titolo del canto
14         # Estrazione dei versi, eliminando spazi vuoti
15         versi = [" ".join(l.itertext()).strip() for l in c.xpath('./l') if " ".join(l.itertext()).strip()]
16         canti[(cantica, h_canto[0])] = versi # Salvataggio dei
    versi con chiave (cantica, titolo del canto)
17
18 data = []
19 # Elaborazione dei testi per suddividerli in frasi con spaCy
20 for (cantica, titolo), versi in canti.items():
21     testo_canto = " ".join(versi) # Unisce tutti i versi in un
    unico testo
22     doc = nlp(testo_canto) # Analizza il testo con il modello NLP
23
24     # Calcola la posizione dei versi nel testo unificato
25     pos_versi = []
26     cur_pos = 0
```

```
27     for v in versi:
28         pos_versi.append((cur_pos, cur_pos + len(v)))
29         cur_pos += len(v) + 1 # Aggiorna la posizione corrente
30
31 # Analizza il testo per estrarre le frasi suddivise da spaCy
32     for frase in doc.sents:
33         frase_testo = frase.text.strip()
34         if len(frase_testo) <= 2: # Scarta frasi troppo brevi
35             continue
36
37 # Trova la posizione della frase nel testo del canto
38         start_idx = testo_canto.find(frase_testo)
39         end_idx = start_idx + len(frase_testo)
40
41 # Identifica i versi che contengono la frase
42         versi_inclusi = [
43             i + 1 for i, (start, end) in enumerate(pos_versi) if
44             not (end < start_idx or start > end_idx)
45         ]
46
47         if not versi_inclusi:
48             print(f"Nessun verso trovato per la frase:\n{
49             frase_testo}")
50             range_versi = "NA"
51         else:
52             range_versi = f"{versi_inclusi[0]}-{versi_inclusi[-1]}"
53             if len(versi_inclusi) > 1 else str(versi_inclusi[0])
54
55 # Salva i dati della frase estratta
56         data.append([cantica, titolo, range_versi, frase_testo])
57
58 df = pd.DataFrame(data, columns=['cantica', 'canto', 'range_versi',
59                                'frase'])
60 df['range_versi'] = "\u200B" + df['range_versi'].astype(str)
61 df.to_csv("frasi.csv", index=False, sep=';', encoding='utf-8-sig')
```

Listing 4.3: Dataset delle frasi

## Capitolo 5

# *Dante Similarity Search:* workflow

In questo capitolo verrà illustrato il processo di sviluppo dell'applicazione *Dante Similarity Search*, evidenziando le principali fasi della sua implementazione. Si partirà dalla preparazione dei dati e creazione degli embeddings e dalla connessione con il database vettoriale Weaviate. Verranno analizzati l'uso dei dati vettorializzati, la configurazione di un tunnel Cloudflare per un accesso sicuro e l'esecuzione dell'applicazione Streamlit, con focus su implementazione tecnica, interfaccia grafica e interazione utente. Come ulteriore passaggio, i risultati ottenuti dall'applicazione sono stati integrati nelle codifiche XML delle testimonianze dei sopravvissuti (cfr. Capitolo 7).



Figura 5.1: Diagramma del workflow

## 5.1 Preparazione dei dati e creazione degli *embeddings*

Dopo aver importato i moduli necessari, viene definita una lista contenente i tre tipi di unità testuali – terzine, versi e frasi – di cui sono stati creati in precedenza i dataset in formato CSV, e per ogni modalità di testo, viene generato dinamicamente il percorso del file corrispondente (righe 2-5). I file CSV vengono dunque caricati in un dizionario di DataFrame (`df_dict`), in cui la chiave è il tipo di unità testuale e il valore è il corrispondente DataFrame (riga 7). Successivamente, viene caricato il modello di SentenceTransformer (riga 12) per generare rappresentazioni numeriche vettoriali (*embeddings*) del testo – in questo caso viene usato il modello *all-mpnet-base-v2*, che risulta quello più preciso e accurato (maggiori dettagli sul confronto tra i modelli sperimentati sono illustrati nel capitolo successivo). Per ogni modalità di testo si estrae la colonna corrispondente (terzina, verso, frase) e si genera l’embedding utilizzando la funzione della libreria `model.encode()` (righe 16-18). Gli embedding calcolati vengono salvati in un dizionario (`embeddings_dict`). Viene aggiunta una nuova colonna “embedding” a ciascun DataFrame, contenente gli embedding calcolati. Ogni embedding viene convertito in lista, poiché i DataFrame di Pandas non supportano direttamente array numpy<sup>1</sup>. I DataFrame aggiornati vengono infine salvati in formato `.parquet`<sup>2</sup> perché è più efficiente in termini di memoria e velocità rispetto al `.csv` (righe 21-32).

```
1 # Definisce le modalita' di analisi (terzine, versi, frasi)
2 modalita = ["terzine", "versi", "frasi"]
3
4 # Crea un dizionario con i percorsi dei file CSV per ogni modalita'
```

---

<sup>1</sup>Generalmente i DataFrame di Pandas supportano gli array di NumPy, ma in questo caso il formato degli embedding non è direttamente compatibile con la struttura di un DataFrame. I DataFrame possono gestire colonne con tipi eterogenei di dati e utilizzano internamente oggetti come serie Pandas, mentre gli array NumPy sono omogenei, e tutte le loro componenti devono avere lo stesso tipo. Convertendo gli embedding in liste, si garantisce una rappresentazione coerente che può essere facilmente memorizzata in una colonna del DataFrame, senza problemi di compatibilità.

<sup>2</sup>Apache Parquet è un formato open-source di file di dati a colonne, pensato per lo stoccaggio e il recupero efficiente dei dati. Offre una compressione efficiente dei dati e schemi di codifica con prestazioni avanzate per gestire dati complessi in massa.



```
5 path_dict = {m: f"../data/{m}.csv" for m in modalita}
6 # Carica i dataset dai file CSV in un dizionario di DataFrame
7 df_dict = {m: pd.read_csv(path_dict[m], delimiter=';') for m in
    modalita}
8
9 # Carica il modello di Sentence Transformer per generare gli
    embedding
10 #model = SentenceTransformer('sentence-transformers/LaBSE')
11 #model = SentenceTransformer('sentence-transformers/paraphrase-
    mpnet-base-v2')
12 model = SentenceTransformer('sentence-transformers/all-mpnet-base-
    v2')
13
14 # Genera gli embedding per ogni modalita' utilizzando il modello
    caricato
15 embeddings_terzine = model.encode(df_dict["terzine"]["terzina"].
    tolist(), show_progress_bar=True)
16 embeddings_versi = model.encode(df_dict["versi"]["verso"].tolist(),
    show_progress_bar=True)
17 embeddings_frase = model.encode(df_dict["frasi"]["frase"].tolist(),
    show_progress_bar=True)
18
19 # Memorizza gli embedding in un dizionario
20 embeddings_dict = {
21     "terzine": embeddings_terzine,
22     "versi": embeddings_versi,
23     "frasi": embeddings_frase
24 }
25 # Aggiunge la colonna "embedding" ai DataFrame corrispondenti
26 for m in modalita:
27     df_dict[m]["embedding"] = list(embeddings_dict[m])
28
29 # Salva i DataFrame con gli embedding in formato Parquet per un
    accesso piu' efficiente
30 for m in modalita:
31     df_dict[m].to_parquet(f'../data/{m}_vectors.parquet')
```

Listing 5.1: Preparazione dei dati e creazione degli embeddings

## 5.2 Connessione con Weaviate

Dopo aver creato un cluster Weaviate, viene attivata la connessione al database definendo l'URL, specificando la chiave API per autenticare l'accesso e impostando il nome della collezione (righe 2-10). Per ogni modalità (terzine, versi, frasi), se la collezione esiste, viene rimossa completamente per evitare duplicati (righe 12-14). Vengono create dunque tre collezioni Weaviate con nomi distinti e proprietà specifiche (righe 17-81). Ogni collezione memorizza informazioni pertinenti a ciascun tipo di unità testuale e possiede le seguenti proprietà:

- **cantica:** Nome della cantica (Inferno, Purgatorio, Paradiso)
- **canto:** Numero del canto.
- **range\_versi:** Intervallo dei versi (per le modalità *terzine* e *frasi*)
- **n\_verso:** Numero del verso (per la modalità *versi*)
- **terzina/verso/frase:** Testo della terzina, del verso e della frase in base alla rispettiva modalità.

Infine si effettua la verifica della connessione: se la connessione con Weaviate è attiva e pronta all'uso viene stampato **True** (riga 82).

```
1 # URL e chiave API per la connessione al database Weaviate
2 WEAVIATE_URL = "https://nv_b1lhkrccnn_vivjge3a.c0.europe-west3.gcp.
   weaviate.cloud"
3 WEAVIATE_API_KEY = "5S_bK8NcTV5ZjUmPJD41dySF_m6KbWj8zaYT"
4 collection_name = "Voci_dall_Inferno"
5
6 # Connessione a Weaviate
7 client = weaviate.connect_to_weaviate_cloud(
8     cluster_url=WEAVIATE_URL,
9     auth_credentials=Auth.api_key(WEAVIATE_API_KEY),
10 )
11 # Eliminazione delle collezioni esistenti
```

```
12 for m in modalita:
13     if client.collections.exists(f"{collection_name}_{m}"): # Se la
14         collezione esiste gia'
15
16         client.collections.delete(f"{collection_name}_{m}") # Cancella
17         la collezione e tutti i suoi dati
18
19 # Creazione della collezione per le terzine
20 voci_dall_inferno_terzine = client.collections.create(
21     name=f"{collection_name}_terzine",
22     properties=[
23         wvc.config.Property(
24             name="cantica",
25             data_type=wvc.config.DataType.TEXT
26         ),
27         wvc.config.Property(
28             name="canto",
29             data_type=wvc.config.DataType.TEXT
30         ),
31         wvc.config.Property(
32             name="range_versi",
33             data_type=wvc.config.DataType.TEXT
34         ),
35         wvc.config.Property(
36             name="terzina",
37             data_type=wvc.config.DataType.TEXT
38         ),
39     ]
40 )
41
42 # Creazione della collezione per i singoli versi
43 voci_dall_inferno_versi = client.collections.create(
44     name=f"{collection_name}_versi",
45     properties=[
46         wvc.config.Property(
47             name="cantica",
48             data_type=wvc.config.DataType.TEXT
49         ),
50         wvc.config.Property(
```

```
47         name="canto",
48         data_type=wvc.config.DataType.TEXT
49     ),
50     wvc.config.Property(
51         name="n_verso",
52         data_type=wvc.config.DataType.TEXT
53     ),
54     wvc.config.Property(
55         name="verso",
56         data_type=wvc.config.DataType.TEXT
57     ),
58 ]
59 )
60 # Creazione della collezione per le frasi
61 voci_dall_inferno_fras = client.collections.create(
62     name=f"{collection_name}_frasi",
63     properties=[
64         wvc.config.Property(
65             name="cantica",
66             data_type=wvc.config.DataType.TEXT
67         ),
68         wvc.config.Property(
69             name="canto",
70             data_type=wvc.config.DataType.TEXT
71         ),
72         wvc.config.Property(
73             name="range_versi",
74             data_type=wvc.config.DataType.TEXT
75         ),
76         wvc.config.Property(
77             name="frase",
78             data_type=wvc.config.DataType.TEXT
79         ),
80     ]
81 )
82 print(client.is_ready()) # Verifica dello stato del client Weaviate
```

Listing 5.2: Connessione con Weaviate

### 5.3 Utilizzo dei dati vettorializzati

In questa fase i dati vengono preparati per essere inseriti, ognuno con il corrispondente embedding, nel database Weaviate (righe 1-37). Per ogni tipo di dato (terzine, versi, frasi), il codice scorre il corrispondente DataFrame (`df_dict["terzine"]`, `df_dict["versi"]`, `df_dict["frasi"]`), creando una lista di oggetti pronti per l'inserimento. Per ogni riga dei DataFrame, viene creato un dizionario con:

- **properties**, ossia attributi testuali (cantica, canto, range versi, testo);
- **vector**, ossia l'embedding vettoriale associato alla relativa unità testuale.

Dopo aver preparato i dati per terzine, versi e frasi, il programma organizza questi dati e li collega alle rispettive collezioni prima di inserirli nel database vettoriale Weaviate (righe 39-49):

- `voci_dall_inferno_dict` contiene i riferimenti alle collezioni Weaviate per ogni tipo di dato;
- `data_rows_dict` raccoglie i dati strutturati pronti per essere inseriti nel database vettoriale.

Una volta che i dati sono pronti, viene eseguito il loro inserimento in *batch*, ossia blocchi di dati, mediante l'utilizzo del metodo `.batch.dynamic()` di Weaviate, il quale crea automaticamente blocchi di dimensione ottimale per caricare i dati in modo efficiente (righe 51-57). Ciò migliora la velocità rispetto all'inserimento di un oggetto alla volta. Quindi, per ogni tipo di dato (terzine, versi, frasi) viene aperta un'operazione batch in cui:

- vengono scorsi i dati già preparati (`data_rows_terzine`, `data_rows_versi`, `data_rows_frase`) e, mediante la funzione `tqdm()`<sup>3</sup>, viene mostrata una barra di avanzamento del loro inserimento;
- ogni oggetto viene aggiunto al batch con i suoi metadati testuali (**properties**) e l'embedding associato (**vector**)

---

<sup>3</sup><https://tqdm.github.io/>

Il batch viene infine caricato automaticamente in Weaviate una volta raggiunta una dimensione ottimale.

```
1 data_rows_terzine = []
2 # Itera sulle righe del DataFrame "terzine" per preparare i dati
3 for _, row in tqdm(df_dict["terzine"].iterrows(), total=len(df_dict
4     ["terzine"]), desc="Preparing data"):
5     data_rows_terzine.append({
6         "properties": { # Salva le proprieta' testuali della
7             "cantica": row['cantica'],
8             "canto": row['canto'],
9             "range_versi": row['range_versi'],
10            "terzina": row['terzina']
11        },
12        "vector": row['embedding'] # Salva l'embedding associato
13    })
14 data_rows_versi = []
15 # Itera sulle righe del DataFrame "versi" per preparare i dati
16 for _, row in tqdm(df_dict["versi"].iterrows(), total=len(df_dict["
17     versi"]), desc="Preparing data"):
18     data_rows_versi.append({
19         "properties": { # Salva le proprieta' testuali del verso
20             "cantica": row['cantica'],
21             "canto": row['canto'],
22             "n_verso": row['n_verso'],
23             "verso": row['verso']
24        },
25        "vector": row['embedding'] # Salva l'embedding associato
26    })
27 data_rows_frase = []
28 # Itera sulle righe del DataFrame "frasi" per preparare i dati
29 for _, row in tqdm(df_dict["frasi"].iterrows(), total=len(df_dict["
30     frasi"]), desc="Preparing data"):
31     data_rows_frase.append({
32         "properties": { # Salva le proprieta' testuali della frase
```

```
31         "cantica": row['cantica'],
32         "canto": row['canto'],
33         "range_versi": row['range_versi'],
34         "frase": row['frase']
35     },
36     "vector": row['embedding'] # Salva l'embedding associato
37 })
38 # Dizionario che associa le modalita' ai rispettivi database
39     Weaviate
40 voci_dall_inferno_dict = {
41     "terzine": voci_dall_inferno_terzine,
42     "versi": voci_dall_inferno_versi,
43     "frasi": voci_dall_inferno_frase
44 }
45 # Dizionario che contiene i dati processati per ogni modalita'
46 data_rows_dict = {
47     "terzine": data_rows_terzine,
48     "versi": data_rows_versi,
49     "frasi": data_rows_frase
50 }
51 # Inserimento dei dati in batch nei database Weaviate
52 for m in modalita:
53     with voci_dall_inferno_dict[m].batch.dynamic() as batch:
54         for data_row in tqdm(data_rows_dict[m], desc="Inserting data"):
55             batch.add_object(
56                 properties=data_row['properties'], # Aggiunge le
57                 proprieta' testuali
58                 vector=data_row['vector'] # Aggiunge l'embedding
59                 vettoriale
60             )
```

Listing 5.3: Inserimento dei vettori

## 5.4 Creazione di un tunnel Cloudflare

Il codice qui di seguito è stato prodotto per configurare e avviare un server Streamlit nell'ambiente Google Colab con Cloudflare Tunnel per esporre localmente l'applicazione web su internet, rendendola accessibile pubblicamente, tramite un tunnel sicuro. Dopo aver scaricato Cloudflare (riga 1), il programma viene reso eseguibile in modo che possa essere avviato come applicazione e disponibile sotto forma di comando di sistema (righe 2-3). Viene poi costruito il file di configurazione `config.toml`<sup>4</sup> (righe 7-11) per Streamlit con le seguenti impostazioni:

- `headless = true`, che avvia Streamlit senza interfaccia grafica, utile per l'esecuzione in un server
- `port = 8501`, che assegna la porta 8501 all'applicazione Streamlit
- `enableCORS = false`, che disabilita le restrizioni CORS, permettendo l'accesso da qualsiasi origine

Infine, viene creato il file `secrets.toml` con le credenziali per Weaviate, contenente l'URL del server Weaviate (`WEAVIATE_URL`), la chiave API di autenticazione (`WEAVIATE_API_KEY`) e i nomi delle collezioni Weaviate per terzine, versi e frasi (righe 13-18). L'ultimo comando elenca i file nella cartella `.streamlit` per verificare che i file `config.toml` e `secrets.toml` siano stati creati correttamente.

```
1 !wget https://github.com/cloudflare/cloudflared/releases/latest/  
  download/cloudflared-linux-amd64  
2 !chmod +x cloudflared-linux-amd64  
3 !mv cloudflared-linux-amd64 /usr/local/bin/cloudflared  
4  
5 !mkdir -p ~/.streamlit  
6  
7 %%writefile ~/.streamlit/config.toml
```

---

<sup>4</sup>TOML (Tom's Obvious Minimal Language) è un formato di file di configurazione minimo che utilizza l'estensione `.toml`. E esso è di facile lettura, è rappresentabile senza ambiguità come una struttura di dati basata su coppie chiave-valore ed è semplice da analizzare e convertire in strutture dati in una vasta gamma di linguaggi.



```
8 [server]
9 headless = true
10 port = 8501
11 enableCORS = false
12
13 %%writefile ~/.streamlit/secrets.toml
14 WEAVIATE_URL = "https://nv_b1lhkrccn_vivjge3a.c0.europe-west3.gcp.
    weaviate.cloud"
15 WEAVIATE_API_KEY = "5S_bK8NcTV5ZjUmPJD41dySF_m6KbWj8zaYT"
16 COLLECTION_NAME_TERZINE = "Voci_dall_Inferno_terzine"
17 COLLECTION_NAME_VERSI = "Voci_dall_Inferno_versi"
18 COLLECTION_NAME_FRASI = "Voci_dall_Inferno_frase"
19
20 !ls -la ~/.streamlit
```

Listing 5.4: Creazione di un tunnel Cloudflare

## 5.5 Esecuzione dell'applicazione Streamlit

Al termine della configurazione del tunnel, viene avviata l'esecuzione dello script dell'applicazione Streamlit (`streamlit_app.py`) e del tunnel Cloudflare, che consente di esporre l'istanza locale di Streamlit (all'indirizzo `http://localhost:8501`) mediante un URL pubblico accessibile via internet. Cloudflare assegna un URL temporaneo attraverso il quale l'applicazione Streamlit in esecuzione può essere raggiunta da qualsiasi utente. L'utilizzo del tunnel elimina la necessità di configurare una porta pubblica o di ricorrere a un server esterno.

```
1 !streamlit run /content/drive/MyDrive/Colab_Notebooks/streamlit_app
    .py &>/dev/null &
2
3 !cloudflared tunnel --url http://localhost:8501
```

Listing 5.5: Lancio dell'applicazione Streamlit

## 5.6 Streamlit web application

L'applicazione Streamlit è sviluppata come uno script Python autonomo. Attraverso il tunnel Cloudflare, essa viene avviata nel browser mediante un URL temporaneo, il quale varia a ogni nuova esecuzione del codice. Dopo aver effettuato il caricamento del modello per convertire i dati testuali in rappresentazioni vettoriali (righe 2-4), viene definita la funzione `find_similar()`, che opererà il confronto tra le citazioni/allusioni presenti nelle testimonianze e i versi della *Commedia* (righe 7-60). Di seguito il suo funzionamento:

- converte il testo inserito in un embedding vettoriale;
- offre la possibilità di applicare filtri opzionali su `cantica` e `canto` per restringere la ricerca;
- esegue una ricerca per similarità nel database Weaviate, mediante la funzione `near_vector()`, per ottenere i versi più vicini nel modello vettoriale;
- restituisce un elenco con i risultati più simili, ordinati per distanza, includendo informazioni distinte in base all'unità testuale selezionata (terzine, versi, frasi).

Vengono dunque definite le categorie di ricerca mediante la creazione di dizionari (righe 64-66): `tot_cantiche` mappa il nome delle cantiche; `tot_canti` associa ogni canto al numero romano corrispondente; `modalità` specifica se la ricerca viene fatta sul dataset delle terzine, su quello dei versi o su quello delle frasi. Dopo aver configurato la connessione con il database Weaviate usando le credenziali segrete `WEAVIATE_URL` e `WEAVIATE_API_KEY` e aver recuperato le tre collezioni di dati (righe 75-81), viene definita l'interfaccia utente (righe 84-95):

- una casella di input permette l'inserimento del testo da analizzare;
- due menù a tendina permettono la scelta multipla opzionale di cantiche e canti;
- un terzo menù a tendina consente la scelta unica e obbligatoria del dataset da utilizzare;

- due slider offrono la possibilità 1) di alzare o abbassare il valore di soglia minimo della similarità dei risultati, 2) di aumentare o diminuire il numero di risultati da mostrare.

Attraverso il pulsante **Search** viene avviata la ricerca: se la query è vuota, viene mostrato un messaggio di errore; se invece la query è presente viene invocata la funzione `find_similar()` che effettua la ricerca. I risultati ottenuti vengono mostrati con la terzina o il verso o la frase individuata, seguiti dal riferimento alla cantica, al canto e all'intervallo di versi in cui si trovano nella *Commedia* e dal punteggio di similarità (righe 98-120).

```
1 @st.cache_resource
2 def load_model(): # Caricamento del modello
3     model = SentenceTransformer('sentence-transformers/all-mpnet-
4         base-v2')
5     return model
6
7 # Funzione per trovare i testi piu' simili alla query
8 def find_similar(query, model, limit=10, cantiche=[], canti=[],
9     evaluation="terzine"):
10     results = []
11     query_vector = model.encode([query])[0] # Converta la query in
12     un vettore numerico
13
14     # Creazione dei filtri per restringere la ricerca
15     filters = None
16     if cantiche and canti:
17         filters = (Filter.by_property("cantica").contains_any(
18             cantiche)
19             & Filter.by_property("canto").contains_any(canti)
20             )
21     elif cantiche:
22         filters = Filter.by_property("cantica").contains_any(
23             cantiche)
24     elif canti:
25         filters = Filter.by_property("canto").contains_any(canti)
```

```
21
22 # Query al database vettoriale Weaviate
23     response = voci_dall_inferno_dict[evaluation].query.near_vector
24     (
25         near_vector=query_vector,
26         limit=limit,
27         return_metadata=MetadataQuery(distance=True),
28         filters=filters
29     )
30 # Estrazione dei risultati in base alla modalita' scelta (terzine,
31     versi, frasi)
32     if evaluation == "terzine":
33         for o in response.objects:
34             results.append({
35                 "cantica": o.properties["cantica"],
36                 "canto": o.properties["canto"],
37                 "range_versi": o.properties["range_versi"],
38                 "terzina": o.properties["terzina"],
39                 "distance": o.metadata.distance
40             })
41     elif evaluation == "versi":
42         for o in response.objects:
43             results.append({
44                 "cantica": o.properties["cantica"],
45                 "canto": o.properties["canto"],
46                 "n_verso": o.properties["n_verso"],
47                 "verso": o.properties["verso"],
48                 "distance": o.metadata.distance
49             })
50     elif evaluation == "frasi":
51         for o in response.objects:
52             results.append({
53                 "cantica": o.properties["cantica"],
54                 "canto": o.properties["canto"],
55                 "range_versi": o.properties["range_versi"],
56                 "frase": o.properties["frase"],
57                 "distance": o.metadata.distance
```

```
56         })
57     else:
58         raise ValueError(f"Unexpected value: {evaluation}")
59
60     return results
61
62
63 # Dizionari per la selezione delle cantiche, dei canti e delle
64   unita' testuali
65 tot_cantiche = {"Inferno": "Inferno", "Purgatorio": "Purgatorio", "
    Paradiso": "Paradiso"}
66 tot_canti = {"Canto I": "I", "Canto II": "II", "Canto III": "III",
    "Canto IV": "IV", "Canto V": "V", "Canto VI": "VI", "Canto VII":
    "VII", "Canto VIII": "VIII", "Canto IX": "IX", "Canto X": "X",
    "Canto XI": "XI", "Canto XII": "XII", "Canto XIII": "XIII", "
    Canto XIV": "XIV", "Canto XV": "XV", "Canto XVI": "XVI", "Canto
    XVII": "XVII", "Canto XVIII": "XVIII", "Canto XIX": "XIX", "
    Canto XX": "XX", "Canto XXI": "XXI", "Canto XXII": "XXII", "
    Canto XXIII": "XXIII", "Canto XXIV": "XXIV", "Canto XXV": "XXV",
    "Canto XXVI": "XXVI", "Canto XXVII": "XXVII", "Canto XXVIII": "
    XXVIII", "Canto XXIX": "XXIX", "Canto XXX": "XXX", "Canto XXXI":
    "XXXI", "Canto XXXII": "XXXII", "Canto XXXIII": "XXXIII", "
    Canto XXXIV": "XXXIV"}
67 modalita = {"terzine": "terzine", "versi": "versi", "frasi": "frasi
    "}
68
69 # Interfaccia Streamlit
70 st.title("Dante Similarity Search")
71
72 # Caricamento del modello
73 model = load_model()
74
75 # Connessione al database Weaviate
76 client = weaviate.connect_to_weaviate_cloud(
77     cluster_url=st.secrets["WEAVIATE_URL"],
78     auth_credentials=Auth.api_key(st.secrets["WEAVIATE_API_KEY"]),
79 )
```

```

79
80 # Creazione del dizionario che collega le modalita' di ricerca alle
    collezioni Weaviate
81 voci_dall_inferno_dict = {m: client.collections.get(st.secrets[f"
    COLLECTION_NAME_{m.upper()}"])} for m in modalita.values()}
82
83 # Input per la ricerca
84 query = st.text_input("Enter your search query:")
85 cantiche = st.multiselect("Select cantica", tot_cantiche.keys())
86 select_cantiche = [tot_cantiche[cantica] for cantica in cantiche]
87 canti = st.multiselect("Select canto", tot_canti.keys())
88 select_canti = [tot_canti[canto] for canto in canti]
89 evaluation = st.multiselect("Select option", modalita.keys(),
    max_selections=1, default=["terzine"])
90 select_evaluation = modalita[evaluation[0]] if evaluation else None
91
92 # Slider per regolare soglia di similarita' e numero di risultati
93 col1, col2 = st.columns(2)
94 threshold = col1.slider("Similarity threshold:", 0.0, 1.0, value
    =0.5, step=0.01)
95 limit = col2.slider("Number of results:", 1, 10, value=5, step=1)
96
97 # Avvio della ricerca quando viene premuto il pulsante
98 if st.button("Search"):
99     if query.strip() == "" or query.strip() == None:
100         st.warning("Error. Insert search query.")
101     else:
102         results = find_similar(query, model, limit, select_cantiche
            , select_canti, select_evaluation)
103
104         if results:
105             st.subheader(f"Found {len(results)} similar verses:")
106             for i, result in enumerate(results, 1):
107
108 # Mostra i risultati trovati in base alla modalita' scelta
109             if select_evaluation == "terzine":
110                 with st.expander(f"{i}. {result ['cantica']},

```

```
111 Canto {result ['canto']], vv. {result['range_versi']} (
Similarity: {1 - result['distance']:.2f})", expanded=True):
112     st.markdown(f"<div class='verse'>{result['
113     terzina']}</div>", unsafe_allow_html=True)
114     elif select_evaluation == "versi":
115         with st.expander(f"{i}. {result ['cantica']},
116         Canto {result ['canto']], v. {result['n_verso']} (Similarity: {1
- result['distance']:.2f})", expanded=True):
117             st.markdown(f"<div class='verse'>{result['
118             verso']}</div>", unsafe_allow_html=True)
119             elif select_evaluation == "frasi":
120                 with st.expander(f"{i}. {result ['cantica']},
121                 Canto {result ['canto']], vv. {result['range_versi']} (
122                 Similarity: {1 - result['distance']:.2f})", expanded=True):
123                     st.markdown(f"<div class='verse'>{result['
124                     frase']}</div>", unsafe_allow_html=True)
125             else:
126                 st.warning("No results found. Try adjusting the
127                 similarity threshold or search query.")
```

Listing 5.6: Script dell'applicazione Streamlit

## 5.7 Interfaccia grafica e interazione utente

L'applicazione web *Dante Similarity Search* si presenta con un'interfaccia grafica molto semplice (fig.5.2). Una box con area testuale permette l'inserimento del frammento di testimonianza da valutare. Al di sotto, tre menù a tendina permettono di filtrare la ricerca: il primo consente la selezione multipla o singola della cantica (fig.5.3); il secondo la selezione multipla o singola dei canti (fig.5.4); il terzo consente la selezione singola e obbligatoria della “metrica” da esaminare (fig.5.5). L'interfaccia mette a disposizione anche due slider: quello a sinistra consente la regolazione del valore minimo della soglia di similarità (che va da 0.00 a 1.00) da prendere in considerazione per la selezione dei risultati; quello a destra permette la gestione del numero di risultati più simili al testo immesso da mostrare, da 1 a 10. Il pulsante *Search* avvia la ricerca.

The screenshot shows the 'Dante Similarity Search' web application interface. At the top, the title 'Dante Similarity Search' is displayed in a bold, dark font. Below the title, there is a search query input field with the placeholder text 'Enter your search query:'. Underneath the input field, there are three dropdown menus for filtering the search: 'Select cantica' with the placeholder 'Choose an option', 'Select canto' with the placeholder 'Choose an option', and 'Select option' with the placeholder 'Choose an option'. The 'Select option' dropdown is currently open, showing a red button labeled 'terzine' with a close icon (x) and a dropdown arrow (v). Below the dropdown menus, there are two sliders. The first slider is labeled 'Similarity threshold:' and ranges from 0.00 to 1.00, with a red dot indicating the current value at 0.50. The second slider is labeled 'Number of results:' and ranges from 1 to 10, with a red dot indicating the current value at 5. At the bottom of the interface, there is a 'Search' button.

Figura 5.2: Interfaccia dell'applicazione Dante Similarity Search



## Dante Similarity Search

Enter your search query:

Select cantica

Choose an option

- Inferno
- Purgatorio
- Paradiso

terzine x

Similarity threshold: 0.50 Number of results: 5

0.00 1.00 1 10

Search

Figura 5.3: Menù di selezione della cantica

Canto I

Canto II

Canto III

Canto IV

Canto V

Canto VI

Canto VII

Canto VIII

Choose an option

Select option

terzine x

Similarity threshold: 0.50 Number of results: 5

0.00 1.00 1 10

Search

Figura 5.4: Menù di selezione del canto

## Dante Similarity Search

Enter your search query:

Select cantica

Choose an option

Select canto

Choose an option

Select option

Choose an option

terzine

versi

frasi

Search

Figura 5.5: Menù di selezione della metrica

Nella figura 5.6 si presenta una simulazione di ricerca di un'espressione esatta del vocabolario di Dante. Viene inserito il frammento di testo “vuolsi così colà”, espressione emblematica del lessico dantesco. Avviando la ricerca senza modificare i valori di default, né aggiungendo filtri di cantica e canto, il sistema trova come primi due risultati le terzine esatte contenenti il frammento testuale immesso, ovvero i versi 22-24 del canto V dell’Inferno (con un punteggio di similarità di 0.63) e i versi 94-96 del canto III della medesima cantica (con un punteggio di similarità di 0.62).

## Dante Similarity Search

Enter your search query:

vuolsi così colà

Select cantica

Choose an option

Select canto

Choose an option

Select option

terzine x

Similarity threshold: 0.50

Number of results: 10

0.00 1.00 1 10

Search

### Found 10 similar verses:

Inferno, Canto V, vv. 22-24 (Similarity: 0.63)

*Non impedir lo suo fatale andare: vuolsi così colà dove si puote ciò che si vuole, e più non dimandare".*

Inferno, Canto III, vv. 94-96 (Similarity: 0.62)

*E 'l duca lui: "Caron, non ti crucciare: vuolsi così colà dove si puote ciò che si vuole, e più non dimandare".*

Purgatorio, Canto VI, vv. 109-111 (Similarity: 0.61)

*Vien, crudel, vieni, e vedi la pressura d'i tuoi gentili, e cura lor magagne; e vedrai Santafior com'è oscura!*

Figura 5.6: Esempio di ricerca di un'espressione esatta

Nella figura 5.7 si presenta, invece, una simulazione di ricerca di un'espressione non esatta. Viene inserito il frammento di testo “amore come vedi non mi abbandona”, espressione che richiama i versi 103-105 del Canto V dell’Inferno. Anche in questo caso, avviando la ricerca senza senza filtraggio, il sistema riesce a trovare la terzina corrispondente restituendola come quinto risultato, con un punteggio di similarità di 0.70.

## Dante Similarity Search

Enter your search query:  

amore come vedi non mi abbandona

Select cantica  

Choose an option

Select canto  

Choose an option

Select option  

terzine x

Similarity threshold: 0.50 Number of results: 10  

0.00 1.00 1 10

Search

providenza eterna;

Paradiso, Canto XXVI, vv. 16-18 (Similarity: 0.70)  

Lo ben che fa contenta questa corte, Alfa e O è di quanta scrittura mi legge Amore o lievemente o forte".

→

Inferno, Canto V, vv. 103-105 (Similarity: 0.70)  

Amor, ch'a nullo amato amar perdona, mi prese del costui piacer sì forte, che, come vedi, ancor non m'abbandona.

Inferno, Canto XVII, vv. 91-93 (Similarity: 0.70)  

I' m'assettai in su quelle spallacce; sì volli dir, ma la voce non venne com'io credetti: "Fa che tu m'abbracce".

Figura 5.7: Esempio di ricerca di un'espressione non esatta

## Capitolo 6

# Valutazione e discussione dell'approccio

In questo capitolo sono presentati i risultati derivanti dall'applicazione di *Dante Similarity Search*, che ha analizzato i frammenti testuali tratti dalle testimonianze dei sopravvissuti. I 17 frammenti esaminati provengono dai seguenti testi: *Diario di Emanuele Artom (novembre 1943 - febbraio 1944)*, *I lunghi giorni della pena di Luigi Giuntini*, *Diario di Nicola Ricci* e *Testimonianza di Liliana Segre (2007)*.

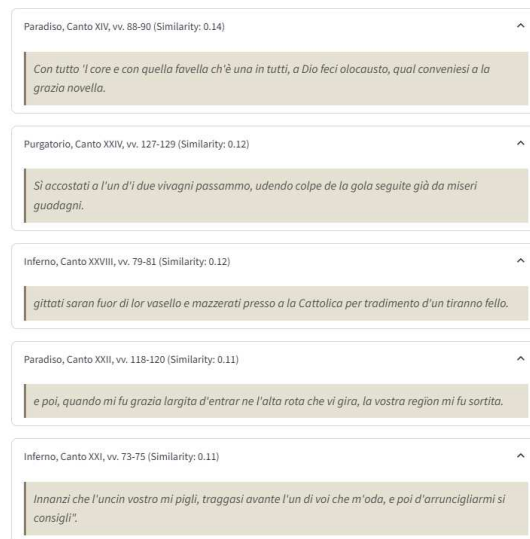
La rappresentazione in embedding è stata effettuata utilizzando il modello Sentence-Transformers All Mpnet Base V2, che ha dimostrato grande precisione e accuratezza nel confronto con i versi della *Commedia* rispetto agli altri due modelli. LaBSE, utilizzato da Mattingly nella sua applicazione *Weaviate Vulgate*, non ha colto nessuna somiglianza tra i frammenti danteschi inseriti per la ricerca e i versi della *Commedia*. Paraphrase Mpnet Base V2 ha dimostrato di riconoscere versi di Dante ma in modo meno puntuale rispetto al modello All Mpnet Base V2. Di seguito vengono riportati due screenshot d'esempio (figure 6.1 e 6.2), uno per ciascun modello, dei risultati che l'applicazione Dante Similarity Search ha restituito relativamente al frammento testuale "vuolsi così colà".

L'analisi degli enunciati è stata condotta prevalentemente senza l'applicazione dei filtri **cantica** e **canto**, con alcune eccezioni. I risultati sono riportati in tabelle: nella prima colonna sono elencati i passi danteschi individuati per ogni "metrica" testuale, preceduti dal numero di apparizione nell'elenco fornito dall'applicazione;

## CAPITOLO 6. VALUTAZIONE E DISCUSSIONE DELL'APPROCCIO

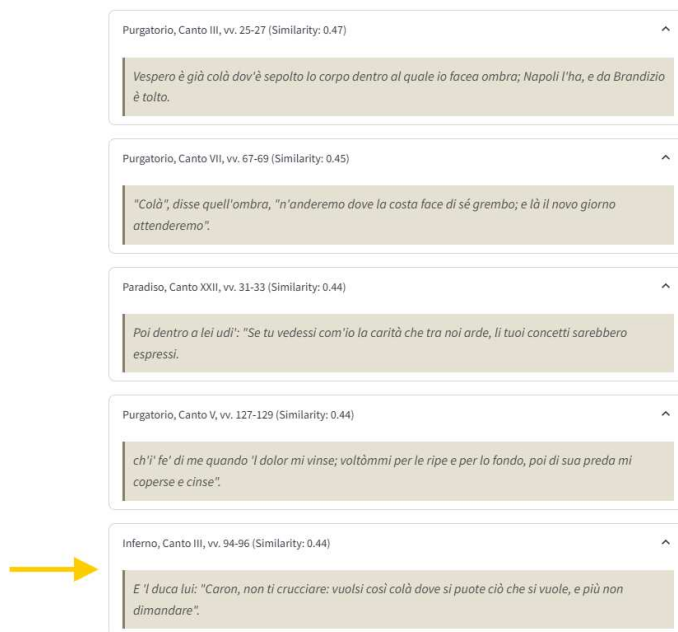
nella seconda colonna è indicato il rispettivo punteggio di similarità. Nei casi in cui è stato applicato un filtro, ciò è specificato nella prima colonna. Sopra ogni tabella è riportata la citazione estratta dalle testimonianze, seguita dalle iniziali dell'autore (EA, LG, NR, LS).

Il capitolo si apre con i risultati relativi alle citazioni esplicite, seguiti da quelli relativi alle citazioni implicite.



Paradiso, Canto XIV, vv. 88-90 (Similarity: 0.14)	Con tutto 'l core e con quella favella ch'è una in tutti, a Dio feci olocausto, qual conveniesi a la grazia novella.
Purgatorio, Canto XXIV, vv. 127-129 (Similarity: 0.12)	S' accostati a l'un d'i due vivagni passammo, udendo colpe de la gola seguite già da miseri guadagni.
Inferno, Canto XXVIII, vv. 79-81 (Similarity: 0.12)	gittati saran fuor di lor vasello e mazzerati presso a la Cattolica per tradimento d'un tiranno fello.
Paradiso, Canto XXII, vv. 118-120 (Similarity: 0.11)	e poi, quando mi fu grazia largita d'entrar ne l'alta rota che vi gira, la vostra region mi fu sortita.
Inferno, Canto XXI, vv. 73-75 (Similarity: 0.11)	Innanzi che l'uncin vostro mi pigli, traggasi avanti l'un di voi che m'oda, e poi d'arrunciarmi si consigli".

Figura 6.1: Risultati ottenuti utilizzando LaBSE



Purgatorio, Canto III, vv. 25-27 (Similarity: 0.47)	Vespera è già colà dov'è sepolto lo corpo dentro al quale io facea ombra; Napoli l'ha, e da Brandizio è tolto.
Purgatorio, Canto VII, vv. 67-69 (Similarity: 0.45)	"Colà", disse quell'ombra, "n'anderemo dove la costa face di sé grembo; e là il novo giorno attenderemo".
Paradiso, Canto XXII, vv. 31-33 (Similarity: 0.44)	Poi dentro a lei udi': "Se tu vedessi com'io la carità che tra noi arde, li tuoi concetti sarebbero espressi.
Purgatorio, Canto V, vv. 127-129 (Similarity: 0.44)	ch'i' fe' di me quando 'l dolor mi vinse; voltòmmi per le ripe e per lo fondo, poi di sua preda mi coperse e cinse".
Inferno, Canto III, vv. 94-96 (Similarity: 0.44)	E 'l duca lui: "Caron, non ti crucciare: vuolsi così colà dove si puote ciò che si vuole, e più non dimandare".

Figura 6.2: Risultati ottenuti utilizzando Paraphrase MpNet Base V2: il riferimento viene colto nel quinto risultato

## 6.1 Citazioni esplicite

Le citazioni esplicite, nell'applicazione Dante Similarity Search, risultano molto agevoli da collegare ai versi danteschi originali. Questo perché, quasi tutte, riportano fedelmente porzioni di testo della *Commedia*, riducendo le ambiguità che possono emergere nell'elaborazione del linguaggio naturale. Grazie a questa corrispondenza diretta, il sistema è in grado di restituire risultati precisi e affidabili, identificando con accuratezza i riferimenti al testo dantesco.

Tabella 6.1: «Grandine grossa, acqua tinta e neve per l'aer tenebroso si riversa;» EA

	Risultati	Similarità
<b>Terzine</b>	1. Inferno, Canto VI, vv. 10-12: <i>Grandine grossa, acqua tinta e neve per l'aere tenebroso si riversa; pute la terra che questo riceve.</i>	0.91
<b>Versi</b>	1. Inferno, Canto VI, v. 10: <i>Grandine grossa, acqua tinta e neve</i>	0.84
	2. Inferno, Canto VI, v. 11: <i>per l'aere tenebroso si riversa;</i>	0.83
<b>Frafi</b>	1. Inferno, Canto VI, vv. 10-11: <i>Grandine grossa, acqua tinta e neve per l'aere tenebroso si riversa;</i>	0.99

Tabella 6.2: «Le ripe eran grommate d'una muffa per l'alito di giù che vi s'appasta che con gli occhi e col naso facea zuffa» LG

	Risultati	Similarità
<b>Terzine</b>	1. Inferno, Canto XVIII, vv. 106-108 : <i>Le ripe eran grommate d'una muffa, per l'alito di giù che vi s'appasta, che con li occhi e col naso facea zuffa.</i>	0.97
<b>Versi</b>	1. Inferno, Canto XVIII, v. 108: <i>che con li occhi e col naso facea zuffa.</i>	0.72
	2. Inferno, Canto XVIII, v. 107: <i>per l'alito di giù che vi s'appasta,</i>	0.71
<b>Frase</b>	1. Inferno, Canto XVIII, vv. 106-108: <i>Le ripe eran grommate d'una muffa, per l'alito di giù che vi s'appasta, che con li occhi e col naso facea zuffa.</i>	0.97

Tabella 6.3: «contraddizion che nol consente» LG

	Risultati	Similarità
<b>Terzine</b>	2. Inferno, Canto XXVII, vv. 118-120: <i>ch'assolver non si può chi non si pente, né pentere e volere insieme puossi per la contradizion che nol consente</i> ".	0.70
<b>Versi</b>	1. Inferno, Canto XXVII, v. 120: <i>per la contradizion che nol consente</i> ".	0.87
<b>Frase</b>	2. Inferno, Canto XXVII, vv. 118-120: <i>ch'assolver non si può chi non si pente, né pentere e volere insieme puossi per la contradizion che nol consente</i> ".	0.70

Tabella 6.4: «La notte ch'io passai con tanta pièta» LG

	Risultati	Similarità
<b>Terzine</b>	5. Inferno, Canto I, vv. 19-21: <i>Allor fu la paura un poco queta, che nel lago del cor m'era durata la notte ch'i' passai con tanta pieta.</i>  (ottenuto con il filtro <b>Inferno</b> )	0.68
<b>Versi</b>	1. Inferno, Canto I, v. 21: <i>la notte ch'i' passai con tanta pieta.</i>	0.94
<b>Frase</b>	9. Inferno, Canto I, vv. 19-21: <i>Allor fu la paura un poco queta, che nel lago del cor m'era durata la notte ch'i' passai con tanta pieta.</i>  (ottenuto con il filtro <b>Inferno</b> )	0.68

Tabella 6.5: «Temp'era dal principio del mattino e 'l sol che montava 'n sù con quelle stelle ch'eran con lui quando l'amor divino mosse di prima quelle cose belle;» LG

	Risultati	Similarità
<b>Terzine</b>	1. Inferno, Canto I, vv. 37-39: <i>Temp'era dal principio del mattino, e 'l sol montava 'n sù con quelle stelle ch'eran con lui quando l'amor divino</i>	0.96
<b>Versi</b>	1. Inferno, Canto I, v. 37: <i>Temp'era dal principio del mattino,</i> 2. Inferno, Canto I, v. 40: <i>mosse di prima quelle cose belle;</i>	0.80 0.74
<b>Frase</b>	1. Inferno, Canto I, vv. 37-40: <i>Temp'era dal principio del mattino, e 'l sol montava 'n sù con quelle stelle ch'eran con lui quando l'amor divino mosse di prima quelle cose belle;</i>	1.00



Tabella 6.6: «"Miserere di me" gridai a lui "qual tu sii od ombra od omo certo?"» LG

	Risultati	Similarità
<b>Terzine</b>	1. Inferno, Canto I, vv. 64-66: <i>Quando vidi costui nel gran deserto, "miserere di me", gridai a lui, "qual che tu sii, od ombra od omo certo!"</i> .	0.93
<b>Versi</b>	1. Inferno, Canto I, v. 65: <i>"miserere di me", gridai a lui,</i>	0.80
	2. Inferno, Canto I, v. 66: <i>"qual che tu sii, od ombra od omo certo!"</i> .	0.77
<b>Frase</b>	1. Inferno, Canto I, vv. 64-66: <i>Quando vidi costui nel gran deserto, "miserere di me", gridai a lui, "qual che tu sii, od ombra od omo certo!"</i> .	0.93

Tabella 6.7: «Lo giorno se n'andava, e l'aere bruno toglieva gli animai che sono in terra dalle fatiche loro» LG

	Risultati	Similarità
<b>Terzine</b>	1. Inferno, Canto II, vv. 1-3: <i>Lo giorno se n'andava, e l'aere bruno toglieva li animai che sono in terra da le fatiche loro; e io sol uno</i>	0.96
<b>Versi</b>	1. Inferno, Canto II, v. 1: <i>Lo giorno se n'andava, e l'aere bruno</i>	0.88
<b>Frase</b>	1. Inferno, Canto II, vv. 1-3: <i>Lo giorno se n'andava, e l'aere bruno toglieva li animai che sono in terra da le fatiche loro;</i>	0.97

Tabella 6.8: «la bella scola» LG

	Risultati	Similarità
<b>Terzine</b>	2. Inferno, Canto IV, vv. 94-96: <i>Così vid'i' adunar la bella scola di quel signor de l'altissimo canto che sovra li altri com'aquila vola.</i>	0.56
<b>Versi</b>	5. Inferno, Canto IV, v. 94: <i>Così vid'i' adunar la bella scola</i>	0.67
<b>Frase</b>	9. Inferno, Canto IV, vv. 94-96 : <i>Così vid'i' adunar la bella scola di quel signor de l'altissimo canto che sovra li altri com'aquila vola.</i>	0.56

Tabella 6.9: «dentro di sé con la sua rabbia» LG

	Risultati	Similarità
<b>Terzine</b>	1. Inferno, Canto VII, vv. 7-9: <i>Poi si rivolse a quella 'nfiata labbia, e disse: "Taci, maladetto lupo! consuma dentro te con la tua rabbia.</i>	0.72
<b>Versi</b>	2. Inferno, Canto VII, v. 9: <i>consuma dentro te con la tua rabbia.</i>	0.74
<b>Frase</b>	1. Inferno, Canto VII, vv. 8-9: <i>"Taci, maladetto lupo! consuma dentro te con la tua rabbia.</i>	0.73

Tabella 6.10: «E caddi come corpo morto cade.» NR

	Risultati	Similarità
<b>Terzine</b>	1. Inferno, Canto V, v. 142: <i>E caddi come corpo morto cade.</i>	1.00
<b>Versi</b>	1. Inferno, Canto V, v. 142: <i>E caddi come corpo morto cade.</i>	1.00
<b>Frase</b>	1. Inferno, Canto V, v. 142: <i>E caddi come corpo morto cade.</i>	1.00

Tabella 6.11: «male altrui» LS

	Risultati	Similarità
<b>Terzine</b>	1. Inferno, Canto II, vv. 88-90: <i>Temer si dee di sole quelle cose c'hanno potenza di fare altrui male; de l'altre no, ché non son paurose.</i>	0.44
<b>Versi</b>	1. Inferno, Canto II, v. 89: <i>c'hanno potenza di fare altrui male;</i>	0.60
	2. Purgatorio, Canto XVII, v. 123: <i>e tal convien che 'l male altrui impronti.</i>	0.55
<b>Frase</b>	1. Inferno, Canto II, vv. 88-89: <i>Temer si dee di sole quelle cose c'hanno potenza di fare altrui male;</i>	0.50

Tabella 6.12: «Dolce colore d'oriental zaffiro» LG

	Risultati	Similarità
<b>Terzine</b>	1. Purgatorio, Canto I, vv. 13-15: <i>Dolce color d'oriental zaffiro, che s'accoglieva nel sereno aspetto del mezzo, puro infino al primo giro,</i>	0.83
<b>Versi</b>	1. Purgatorio, Canto I, v. 13: <i>Dolce color d'oriental zaffiro,</i>	0.96
<b>Frase</b>	1. Purgatorio, Canto I, vv. 13-18: <i>Dolce color d'oriental zaffiro, che s'accoglieva nel sereno aspetto del mezzo, puro infino al primo giro, a li occhi miei ricominciò diletto, tosto ch'io uscì' fuor de l'aura morta che m'avea contristati li occhi e 'l petto.</i>	0.78

Tabella 6.13: «fuor delle braccia del suo dolce amico» LG

	Risultati	Similarità
<b>Terzine</b>	3. Purgatorio, Canto IX, vv. 1-3: <i>La concubina di Titone antico già s'imbiancava al balco d'oriente, fuor de le braccia del suo dolce amico;</i>	0.69
<b>Versi</b>	1. Purgatorio, Canto IX, v. 3: <i>fuor de le braccia del suo dolce amico;</i>	0.90
<b>Frase</b>	4. Purgatorio, Canto IX, vv. 1-3: <i>La concubina di Titone antico già s'imbiancava al balco d'oriente, fuor de le braccia del suo dolce amico;</i>	0.69

Tabella 6.14: «di spirito profetico dotato» LG

	Risultati	Similarità
Terzine	1. Paradiso, Canto XII, vv. 139-141: <i>Rabano è qui, e lucemi dallato il calavrese abate Giovacchino di spirito profetico dotato.</i>	0.73
Versi	1. Paradiso, Canto XII, v. 141: <i>di spirito profetico dotato.</i>	0.98
Fraasi	1. Paradiso, Canto XII, vv. 139-141: <i>Rabano è qui, e lucemi dallato il calavrese abate Giovacchino di spirito profetico dotato.</i>	0.73

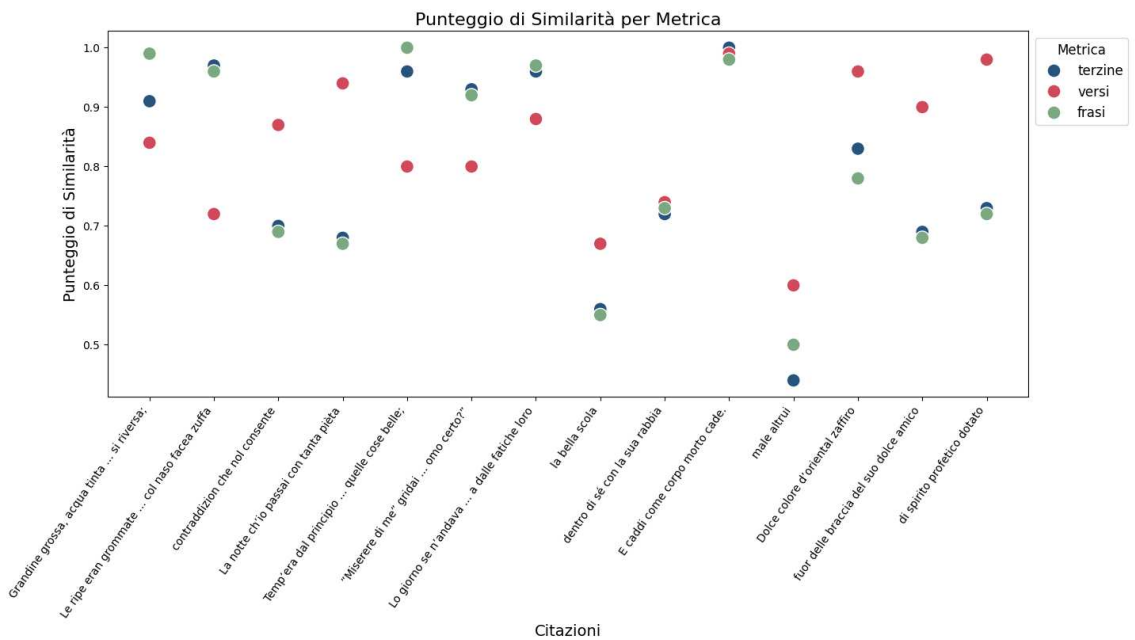


Figura 6.3: Grafico Scatterplot che mostra i punteggi di similarità per ogni metrica

## 6.2 Citazioni implicite

Le citazioni implicite, per l'applicazione Dante Similarity Search, presentano una maggiore difficoltà nell'essere collegate al passo dantesco esatto. Questo accade perché il linguaggio naturale, grazie alla sua imprevedibilità, può parafrasare i versi della *Commedia* in maniera originale, creando sfumature che i modelli informativi non riescono sempre a catturare pienamente. Nonostante ciò, Dante Similarity Search riesce a fornire risultati di buona qualità, coerenti con il frammento testuale analizzato.

Tabella 6.15: «Almeno mi pareva d'esser come Ciacco stasera, pochi minuti fa, quando tornavo in albergo dopo aver camminato per ore al buio, perdendomi e disperdendomi per la campagna, sotto una pioggia continua, affondando nel fango e nella fanghiglia» EA

	Risultati	Similarità
<b>Terzine</b>	5. Inferno, Canto VI, vv. 52-54: <i>Voi cittadini mi chiamaste Ciacco: per la dannosa colpa de la gola, come tu vedi, a la pioggia mi fiacco.</i>  (ottenuto con i filtri Inferno VI)	0.60
<b>Versi</b>	Nessun risultato utile	
<b>Frase</b>	10. Inferno, Canto VI, vv. 52: <i>Voi cittadini mi chiamaste Ciacco:</i>  (ottenuto con i filtri Inferno VI)	0.58

La citazione implicita «stare sempre agitato come selvaggina che può essere colta di sorpresa.» fa riferimento ai versi 109-114 del canto XIII dell'*Inferno* (*Noi eravamo ancora al tronco attesi, / credendo ch'altro ne volesse dire, / quando noi fummo d'un romor sorpresi, / similmente a colui che venire / sente 'l porco e la caccia a la sua posta, / ch'ode le bestie, e le frasche stormire*). L'applicazione non riesce a cogliere l'allusione a queste terzine e trova un'associazione con un passo caratterizzato dalla presenza delle parole *selvagge* e *cólti*.

Tabella 6.16: «stare sempre agitato come selvaggina che può essere colta di sorpresa.» EA

	Risultati	Similarità
<b>Terzine</b>	1. <i>Inferno</i> , Canto XIII, vv. 7-9: <i>Non han sì aspri sterpi né sì folli quelle fiere selvagge che 'n odio hanno tra Cecina e Corneto i luoghi cólti.</i>	0.69
<b>Versi</b>	5. <i>Inferno</i> , Canto XIII, v. 8: <i>quelle fiere selvagge che 'n odio hanno</i> (ottenuto con i filtri <b>Inferno XIII</b> )	0.60
<b>Frase</b>	1. <i>Inferno</i> , Canto XIII, vv. 7-9: <i>Non han sì aspri sterpi né sì folli quelle fiere selvagge che 'n odio hanno tra Cecina e Corneto i luoghi cólti.</i> (ottenuto con il filtro <b>Inferno</b> )	0.69

Anche nel caso illustrato nella tabella 6.17 l'applicazione non coglie uno dei più famosi versi della *Commedia*: *ed elli avea del cul fatto trombetta*. (Inferno, Canto XXI, v. 139). Tuttavia, riesce a riconoscere il nome *Barbariccia* e restituisce i passi in cui compare.

Tabella 6.17: «l'altro sonava di continuo la tromba di Barbariccia.» EA

	Risultati	Similarità
<b>Terzine</b>	1. Inferno, Canto XXII, vv. 145-147: <i>Barbariccia, con li altri suoi dolente, quattro ne fé volar da l'altra costa con tutt'i raffi, e assai prestamente</i>	0.76
	2. Inferno, Canto XXII, vv. 28-30: <i>sì stavan d'ogne parte i peccatori; ma come s'appressava Barbariccia, così si ritraén sotto i bollori.</i>	0.74
<b>Versi</b>	1. Inferno, Canto XXII, v. 145: <i>Barbariccia, con li altri suoi dolente,</i>	0.85
	2. Inferno, Canto XXI, v. 120: <i>e Barbariccia guidi la decina.</i>	0.78
	3. Inferno, Canto XXII, v. 59: <i>ma Barbariccia il chiuse con le braccia</i>	0.78
<b>Frase</b>	1. Inferno, Canto XXII, vv. 145-148: <i>Barbariccia, con li altri suoi dolente, quattro ne fé volar da l'altra costa con tutt'i raffi, e assai prestamente di qua, di là discesero a la posta;</i>	0.78
	2. Inferno, Canto XXII, vv. 59-60: <i>ma Barbariccia il chiuse con le braccia e disse:</i>	0.78
	3. Inferno, Canto XXI, vv. 120: <i>e Barbariccia guidi la decina.</i>	0.78
	4. Inferno, Canto XXII, vv. 29-30: <i>ma come s'appressava Barbariccia, così si ritraén sotto i bollori.</i>	0.77



Per quanto riguarda la citazione dal diario di Nicola Ricci «Si va verso la fame, si va verso il freddo, si va verso l'inferno», l'applicazione Dante Similarity Search non è riuscita a trovare una giusta corrispondenza. Mentre per un umano, con una anche minima conoscenza della *Commedia* di Dante, la terzina balena subito nella mente: *Per me si va ne la città dolente, / per me si va ne l'eterno dolore, / per me si va tra la perduta gente.* (Inferno, Canto III, vv. 1-3). Questo risultato mette in luce i limiti del modello informatico nell'interpretare i riferimenti impliciti e la profondità delle allusioni letterarie, che richiedono una sensibilità contestuale e culturale che supera le sue possibilità. Un possibile miglioramento potrebbe consistere nel fine-tuning del modello su un corpus specifico della *Commedia*, comprendente anche commenti critici al testo. L'addestramento su un dataset arricchito con esempi di citazioni, parafrasi e riferimenti intertestuali potrebbe migliorare la capacità del modello di riconoscere connessioni non esplicite tra testi diversi e di distinguere tra similitudini superficiali e relazioni semantiche più profonde.

L'impiego di un LLM come ChatGPT potrebbe rappresentare un'opzione per migliorare l'analisi delle somiglianze concettuali e stilistiche, grazie alla sua capacità di elaborare riferimenti testuali con maggiore profondità. Nella figura 6.4 si mostra un esempio di applicazione di ChatGPT per identificare la citazione di Nicola Ricci menzionata sopra. Tuttavia, un modello di questo tipo, pur sfruttando un'architettura avanzata di transformer, comporta alcune criticità. L'elevata complessità computazionale, i costi di addestramento, nonché la necessità di una gestione attenta delle risorse, rendono il suo utilizzo meno immediato rispetto a soluzioni basate su embeddings vettoriali. Inoltre, un LLM, per quanto potente, non è immune da errori interpretativi e può generare connessioni arbitrarie o fuori contesto. Di conseguenza, il suo impiego andrebbe valutato attentamente, bilanciando i potenziali benefici con le difficoltà tecniche e operative che comporta.

ChatGPT ▾

Immagina di essere un esperto di letteratura italiana medievale e romanza.  
Prendi in considerazione questa frase: "Si va verso la fame, si va verso il freddo, si va verso l'inferno".  
A quale poeta ti fa pensare?

Questa frase richiama molto da vicino lo stile e le tematiche di **Dante Alighieri**, in particolare la *Divina Commedia*. Il ritmo cadenzato e iterativo ("Si va verso...") ricorda la solennità e l'inesorabilità del viaggio ultraterreno di Dante attraverso Inferno, Purgatorio e Paradiso.

L'immagine della *fame* potrebbe evocare il conte Ugolino (Inferno, XXXIII), condannato alla fame eterna, mentre il *freddo* richiama gli ultimi cerchi dell'Inferno, in cui Lucifero è immerso nel ghiaccio (Inferno, XXXIV). Infine, l'idea dell'*Inferno* è una chiara allusione al cammino di dannazione che Dante descrive nel suo viaggio.

ChatGPT ▾

Sì, il poeta di riferimento è Dante. Riesci a indicarmi a quale *terzina* della *Commedia* allude?

La frase che hai citato sembra alludere a un passo del **Canto III dell'Inferno**, in particolare alla *terzina* che si trova poco dopo l'ingresso nella selva oscura, quando Dante e Virgilio giungono alla porta dell'Inferno e leggono la celebre iscrizione:

"Per me si va ne la città dolente,  
per me si va ne l'eterno dolore,  
per me si va tra la perduta gente." (*Inferno*, III, 1-3)

La struttura ripetitiva della tua frase ("Si va verso...") richiama il "Per me si va..." dantesco, sottolineando il carattere ineluttabile e terribile del cammino infernale.

Figura 6.4: Esempio di ricerca con ChatGPT

## Capitolo 7

# Integrazione dell'applicazione nel processo di codifica dei dati

Un'ulteriore passaggio che si è scelto di fare per il progetto Voci dall'Inferno è quello di aggiungere i risultati ottenuti dall'applicazione Dante Similarity Search alle codifiche XML delle testimonianze dei sopravvissuti. Per eseguire quest'operazione è stato costruito un nuovo codice Python, riportato di seguito.

Dopo aver scansionato la collezione dei documenti XML delle codifiche e aver caricato il modello SentenceTransformer mediante la funzione `load_model` importata dal codice dell'applicazione Streamlit (righe 1-7), si procede con l'elaborazione di ciascun file XML. Per ogni file viene caricato e analizzato l'albero XML (righe 9-11) e si esegue una ricerca di tutti gli elementi `<cit>`, che rappresentano le citazioni contenute nel testo (riga 13). Poiché le citazioni di Dante sono state codificate con l'attributo `@type`, a indicare la loro natura esplicita o implicita, se l'elemento `<cit>` che viene trovato non presenta questo attributo, viene ignorato. Al contrario, se l'attributo è presente, si naviga l'elemento `<cit>` alla ricerca dell'elemento `<quote>`, che è quello che contiene il testo della citazione (righe 14-16). Il testo viene dunque estratto e ripulito da punteggiatura e caratteri speciali (righe 19-21). Per ogni citazione trovata vengono eseguite tre ricerche semantiche (una per le terzine, una per i versi e una per le frasi) utilizzando la funzione `find_similar()` (anch'essa importata dal codice dell'applicazione Streamlit). Per ciascuna categoria vengono estratti i primi tre risultati più simili ottenuti dalla funzione (righe 23-34).

Per ogni risultato:

- viene creato un nuovo elemento `<quote>`, con l'attributo `type` che indica la categoria e un id progressivo (righe 36-38);
- il testo della citazione trovata viene inserito nell'elemento `<quote>` (riga 39);
- viene aggiunto un elemento `<bibl>` contenente il riferimento bibliografico composto da cantica, canto e range di versi (o numero del verso) (righe 40-50).

Il file XML viene infine salvato con un nuovo nome in modo da mantenere anche il file originale (riga 52).

```
1 cartella_xml = "../codifiche"
2 file_xml = glob.glob(f"{cartella_xml}/*.xml") # Trova tutti i file
      XML nella cartella specificata
3
4 print(f"Trovati {len(file_xml)} file XML")
5
6 # Caricamento del modello di Sentence Transformer
7 model = SentenceTransformer('sentence-transformers/all-mpnet-base-
      v2')
8 load_model() # Funzione esterna per il caricamento del modello
9
10 # Itera su tutti i file XML trovati
11 for file_path in file_xml:
12     tree = ET.parse(file_path)
13     root = tree.getroot()
14
15 # Cerca tutti gli elementi <cit> nel file XML
16 for cit in root.findall(f"//cit"):
17     if "type" not in cit.attrib:
18         continue # Salta gli elementi <cit> senza attributo "type"
19     quote_element = cit.find(f"//quote") # Trova il testo citato
20     cit_text = ""
21
```

```
22 # Estrazione del testo dalla citazione
23     for e in quote_element.itertext():
24         cit_text += e.strip() + " "
25     cit_text = cit_text.translate(str.maketrans(' ', '', string.
26         punctuation + '\"><>')) # Rimozione della punteggiatura e
27         caratteri speciali
28
29 # Confronto della citazione con tre tipi di unita' testuali
30     for select_evaluation in ["terzine", "versi", "frasi"]:
31         results = find_similar(cit_text, model, limit=3, cantiche=[],
32             canti=[], evaluation=select_evaluation)
33
34 # Itera sui primi 3 risultati trovati
35     for i, res in enumerate(results[:3]):
36         if select_evaluation == "terzine":
37             testo = res.get("terzina", "")
38         elif select_evaluation == "versi":
39             testo = res.get("verso", "")
40         elif select_evaluation == "frasi":
41             testo = res.get("frase", "")
42         else:
43             testo = ""
44
45 # Crea un nuovo elemento <quote> con il risultato trovato
46     quote = ET.SubElement(cit, "quote")
47     quote.set("type", select_evaluation)
48     quote.set("id", f"{select_evaluation}_{i+1}")
49     quote.text = testo
50     bibl = ET.SubElement(quote, "bibl") # Crea l'elemento
51     bibliografico
52
53 # Recupera riferimenti testuali
54     cantica = res.get("cantica", "")
55     canto = res.get("canto", "")
56     range_versi = res.get("range_versi", "")
57     n_verso = res.get("n_verso", "")
```

## CAPITOLO 7. INTEGRAZIONE DELL'APPLICAZIONE NEL PROCESSO DI CODIFICA DEI DATI

---

```
55     # Formatta la citazione bibliografica
56     if select_evaluation == "versi":
57         bibl.text = f"{cantica}, Canto {canto}, v. {n_verso}"
58     else:
59         bibl.text = f"{cantica}, Canto {canto}, vv. {range_versi}"
60     "
61 # Salva il file XML con le citazioni aggiornate
62 tree.write(file_path+".cit.xml", encoding="utf-8",
            xml_declaration=True)
```

Listing 7.1: Trasposizione dei risultati nelle codifiche delle testimonianze

# Conclusioni e sviluppi futuri

Il lavoro di tesi qui illustrato rappresenta un ulteriore contributo per il progetto di ricerca *Voci dall'Inferno* dell'Università di Pisa. L'applicazione web *Dante Similarity Search* consente il riconoscimento automatico di citazioni e allusioni al lessico dantesco. Lo studio è stato effettuato sul corpus di *Voci dall'Inferno*, costituito da 23 testimonianze (per un totale di 500.000 token) dei sopravvissuti al Lager. Per la realizzazione del software sono stati impiegati molteplici strumenti: i modelli di SentenceTransformers per la creazione degli embeddings contestuali; il database vettoriale allo stato dell'arte Weaviate per la conservazione e l'interrogazione dei dati; il framework Streamlit per lo sviluppo dell'applicazione web. *Dante Similarity Search* consente la ricerca di frasi e il loro confronto con i versi della *Commedia* di Dante (il cui numero di parole ammonta a 101.698), e, calcolando il punteggio di similarità, mostra i risultati che si avvicinano maggiormente alle espressioni dantesche. Infine questi risultati sono stati integrati nelle codifiche XML delle testimonianze dei sopravvissuti. L'applicazione *Dante Similarity Search* ha offerto risultati soddisfacenti. Le citazioni esplicite si associano facilmente ai versi originali della *Commedia*, poiché riportano fedelmente porzioni di testo e riducono le ambiguità nell'elaborazione del linguaggio naturale, consentendo così un'identificazione precisa dei riferimenti danteschi. Le citazioni implicite risultano più difficili da collegare a un passo specifico, dato che il linguaggio parafrasato introduce sfumature che il modello non sempre coglie appieno, evidenziando i limiti nell'interpretare allusioni letterarie profonde che richiederebbero una sensibilità contestuale e culturale maggiore. Tuttavia l'applicazione è riuscita a produrre risultati di elevata qualità, perfettamente in linea con il frammento testuale analizzato.

Il progetto qui discusso rappresenta un punto di partenza per ulteriori avanzamenti

e miglioramenti. Come primo passo, si prevede l'integrazione del software *Dante Similarity Search* con l'applicazione web *Voci dall'Inferno*, progettata nel 2024 da Elvira Mercatanti. Questa applicazione raccoglie l'intero corpus di testimonianze, consentendo la consultazione e l'analisi delle stesse, insieme al repertorio di tessere dantesche identificate nelle testimonianze.

Un'operazione di fine-tuning del modello destinato alla creazione degli embedding potrebbe rappresentare un importante passo in avanti per il miglioramento dell'applicazione. Questo processo di adattamento consentirebbe al modello di acquisire una maggiore sensibilità verso le particolarità linguistiche e stilistiche dei testi su cui viene addestrato. In particolare, se l'addestramento fosse condotto su testi in italiano volgare o, ancora meglio, su opere dantesche, il modello potrebbe catturare con maggiore precisione le sfumature che il testo può assumere. Di conseguenza, l'applicazione potrebbe offrire risultati più accurati e pertinenti, migliorando l'analisi semantica e la capacità di identificare somiglianze concettuali e stilistiche.

Inoltre, un adattamento del codice per la trasposizione dei risultati nelle codifiche XML potrebbe rendere possibile il riconoscimento automatico delle citazioni all'interno di testi non codificati, facilitandone così la successiva codifica.

Attualmente è in corso la ricerca per individuare un servizio di hosting affidabile al fine di ospitare l'applicazione in modo da renderla accessibile online e fruibile da tutti.



# Bibliografia

- Artom, E. (2008). *Diari di un partigiano ebreo, gennaio 1940-febbraio 1944*. (G. Schwarz, Cur.). Bollati Boringhieri.
- Baiolo, C. (2024). *La codifica del Diario partigiano di Emanuele Artom: sulle tracce di una biblioteca da studiare* [Tesi Triennale]. Università di Pisa.
- Basharat, A., Yazdansepas, D., & Rasheed, K. (2015). Comparative study of verse similarity for multi-lingual representations of the qur'an.
- Burnard, L. (2014). *What is the text encoding initiative? : How to add intelligent markup to digital resources*. OpenEdition Press. Recuperato dicembre 8, 2024, da <https://books.openedition.org/oep/426>
- Calderini, S. (2015). *L'ineffabilità della nefandezza* [Tesi di dottorato, Università di Pisa].
- Calderini, S., & Riccucci, M. (2020). L'ineffabilità della nefandezza : Dante 'per dire' il Lager : un sondaggio preliminare nelle testimonianze non letterarie. *Italianistica*, (1). <https://doi.org/10.19272/202001301011>
- Del Grosso, A. M., Riccucci, M., & Mercatanti, E. (2024a, ottobre 13). *Voci dall'Inferno: The voices of the Lager survivors*. <https://doi.org/10.5281/zenodo.13925918>
- Del Grosso, A. M., Riccucci, M., & Mercatanti, E. (2024b). Voci dall'Inferno: Dante per dire il Lager - digitalizzare e studiare le testimonianze.
- Del Grosso, A. M., Riccucci, M., & Mercatanti, E. (2024c, maggio). The Impact of Digital Editing on the Study of Holocaust Survivors' Testimonies in the context of Voci dall'Inferno Project. In I. Anuradha, M. Wynne, F. Frontini & A. Plum (Cur.), *Proceedings of the First Workshop on Holocaust Testimonies as Language Resources (HTRes) @ LREC-COLING 2024* (pp. 1–9). ELRA;

- ICCL. Recuperato dicembre 8, 2024, da <https://aclanthology.org/2024.htres-1.1>
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, maggio 24). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. <https://doi.org/10.48550/arXiv.1810.04805>
- Feng, F., Yang, Y., Cer, D., Arivazhagan, N., & Wang, W. (2022, marzo 8). Language-agnostic BERT Sentence Embedding. <https://doi.org/10.48550/arXiv.2007.01852>
- Giuntini, L. (2021). *I lunghi giorni della pena: diario di prigionia (8 settembre 1943-15 aprile 1945)* (G. F. Gabrielli, Cur.; Vol. III). Regione Toscana, Consiglio regionale.
- Mercatanti, E. (2024). *Le voci dei sopravvissuti ai Lager: l'applicazione web per l'analisi, per l'interrogazione e per la fruizione delle testimonianze* [Tesi Magistrale]. Università di Pisa.
- Mercatanti, E., Congiu, C., Del Grosso, A. M., & Riccucci, M. (2024). Voci dall'inferno: A web application to study and analyze the lager testimonies.
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, settembre 7). Efficient Estimation of Word Representations in Vector Space. <https://doi.org/10.48550/arXiv.1301.3781>
- Pennington, J., Socher, R., & Manning, C. (2014, ottobre). GloVe: Global Vectors for Word Representation. In A. Moschitti, B. Pang & W. Daelemans (Cur.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Association for Computational Linguistics. <https://doi.org/10.3115/v1/D14-1162>
- Petrilli, I. (2022). *Dante fra gli Internati Militari Italiani: il caso 'Luigi Giuntini'. Codifica preliminare del suo diario I lunghi giorni della pena* [Tesi Triennale]. Università di Pisa.
- Reimers, N., & Gurevych, I. (2019, agosto 27). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. <https://doi.org/10.48550/arXiv.1908.10084>

- Ricci, P. (2022). *Mio nonno deportato ad Armensterin, il suo diario inedito* [Tesi Triennale]. Università di Pisa.
- Riccucci, M., Del Grosso, A. M., Valecchi, F., & Causarano, G. (2021). Testimoniare il Lager: l'informatica al servizio della memoria [Medium: text Publisher: Alma Mater Studiorum - Università di Bologna]. *AIUCD 2021 - Book of Extended Abstracts*. <https://doi.org/10.6092/UNIBO/AMSACTA/6712>
- Riccucci, M., & Ricotti, L. (2021). *Il dovere della parola. La Shoah nelle testimonianze di Liliana Segre e di Goti Herskovitz Bauer*. Pacini Editore.
- Siragusa, F. (2024). *I luoghi danteschi della memoria dei sopravvissuti* [Tesi Magistrale]. Università di Pisa.
- Song, K., Tan, X., Qin, T., Lu, J., & Liu, T.-Y. (2020, novembre 2). MPNet: Masked and permuted pre-training for language understanding. <https://doi.org/10.48550/arXiv.2004.09297>
- streamlit/streamlit* [original-date: 2019-08-24T00:14:52Z]. (2025, marzo 28). Streamlit. Recuperato marzo 28, 2025, da <https://github.com/streamlit/streamlit>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023, agosto 2). Attention Is All You Need. <https://doi.org/10.48550/arXiv.1706.03762>

# Sitografia

- About GitHub and git* [GitHub docs]. (2008). Recuperato marzo 28, 2025, da <https://docs-internal.github.com/en/get-started/start-your-journey/about-github-and-git>
- The AI-native database developers love — weaviate*. (2019). Recuperato gennaio 20, 2025, da <https://weaviate.io/>
- All mpnet base v2 · models · dataloop*. (2024). Recuperato marzo 28, 2025, da [https://dataloader.ai/library/model/sentence-transformers\\_all-mpnet-base-v2/](https://dataloader.ai/library/model/sentence-transformers_all-mpnet-base-v2/)
- Bag of Tricks for Efficient Text Classification*. (2016). Recuperato marzo 28, 2025, da <https://arxiv.org/abs/1607.01759>
- Build software better, together* [GitHub]. (2025). Recuperato marzo 28, 2025, da <https://github.com>
- Che cos'è Apache Parquet?* [Databricks]. (2018, ottobre 30). Recuperato marzo 28, 2025, da <https://www.databricks.com/it/glossary/what-is-parquet>
- Cloudflare tunnel · cloudflare zero trust docs* [Cloudflare docs]. (2024, agosto 27). Recuperato marzo 28, 2025, da <https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/>
- Colab.google* [Colab.google]. (2017). Recuperato marzo 28, 2025, da <http://0.0.0.0:8080/>
- Cos'è Google Colab* [Aulab]. (2017). Recuperato marzo 28, 2025, da <https://aulab.it/guide-avanzate/cose-google-colab>
- Da Costa-Luis, C. e. a. (2024, novembre 27). *tqdm: A fast, Extensible Progress Bar for Python and CLI* (Ver. v4.67.1). Zenodo. <https://doi.org/10.5281/ZENODO.595120>

- Danteonline versione beta 2020* [danteonlineita]. (2020). Recuperato marzo 28, 2025, da <https://danteonline.it/index.html>
- floret*. (2022, agosto 23). Recuperato marzo 28, 2025, da <https://explosion.ai/blog/explosion.ai>
- Google — *LaBSE* — *kaggle*. (2021). Recuperato marzo 28, 2025, da <https://www.kaggle.com/models/google/labse>
- Google Colab. (2017). Recuperato marzo 28, 2025, da <https://research.google.com/colaboratory/intl/it/faq.html>
- Govindaraj, P. (2024, agosto 26). *Building sentence embeddings with dual-encoder* [Medium]. Recuperato marzo 28, 2025, da <https://medium.com/@govindarajpriyanthan/building-sentence-embeddings-with-dual-encoder-a-comprehensive-guide-3dd3069b54ff>
- Hugging Face – The AI community building the future*. (2025, gennaio 18). Recuperato gennaio 20, 2025, da <https://huggingface.co/>
- Introduction* — *weaviate*. (2019). Recuperato marzo 28, 2025, da <https://weaviate.io/developers/weaviate/introduction>
- Le parole di Dante per raccontare l'orrore dei Lager - Gli Stati Generali*. (2019). Recuperato marzo 28, 2025, da <https://www.glistatigenerali.com/cultura/letteratura/le-parole-di-dante-per-raccontare-lorrore-dei-lager/>
- Linguaggio di programmazione Python*. (2021). Recuperato marzo 28, 2025, da <https://www.python.it/>
- lxml - Processing XML and HTML with Python*. (2024). Recuperato marzo 28, 2025, da <https://lxml.de/>
- Mattingly, W. (2024a, ottobre 24). *wjbmattingly/vulgata-spacy* [original-date: 2022-12-27T20:23:52Z]. Recuperato gennaio 22, 2025, da <https://github.com/wjbmattingly/vulgata-spacy>
- Mattingly, W. (2024b, novembre 15). *wjbmattingly/weaviate-vulgate* [original-date: 2024-08-04T23:25:14Z]. Recuperato marzo 28, 2025, da <https://github.com/wjbmattingly/weaviate-vulgate>
- microsoft/mpnet-base · Hugging Face*. (2021). Recuperato gennaio 20, 2025, da <https://huggingface.co/microsoft/mpnet-base>

- MPNet*. (2021). Recuperato marzo 28, 2025, da [https://huggingface.co/docs/transformers/model\\_doc/mpnet](https://huggingface.co/docs/transformers/model_doc/mpnet)
- Opere*. (2020). Recuperato marzo 28, 2025, da <https://www.danteonline.it/opere/index.php?opera=Commedia%20-%20ed.%20Petrocchi>
- Paraphrase mpnet base v2 · models · dataloop*. (2022). Recuperato marzo 28, 2025, da [https://dataloader.ai/library/model/sentence-transformers\\_paraphrase-mpnet-base-v2/](https://dataloader.ai/library/model/sentence-transformers_paraphrase-mpnet-base-v2/)
- Pretrained Models — Sentence Transformers documentation*. (2025). Recuperato marzo 28, 2025, da [https://www.sbert.net/docs/sentence\\_transformer/pretrained\\_models.html](https://www.sbert.net/docs/sentence_transformer/pretrained_models.html)
- Preview local projects with cloudflare tunnel · cloudflare pages docs* [Cloudflare docs]. (2024). Recuperato marzo 28, 2025, da <https://developers.cloudflare.com/pages/how-to/preview-with-cloudflare-tunnel/>
- Python tutorial*. (2024). Recuperato marzo 28, 2025, da [https://www.datacamp.com/tutorial/streamlit?dc\\_referrer=https%3A%2F%2Fwww.google.com%2F&utm\\_source=google&utm\\_medium=paid\\_search&utm\\_campaign=230119\\_1-sea%7Edsa%7Etofu\\_2-b2c\\_3-row-p2\\_4-prc\\_5-na\\_6-na\\_7-le\\_8-pdsh-go\\_9-nb-e\\_10-na\\_11-na-jan25](https://www.datacamp.com/tutorial/streamlit?dc_referrer=https%3A%2F%2Fwww.google.com%2F&utm_source=google&utm_medium=paid_search&utm_campaign=230119_1-sea%7Edsa%7Etofu_2-b2c_3-row-p2_4-prc_5-na_6-na_7-le_8-pdsh-go_9-nb-e_10-na_11-na-jan25)
- Quick tunnels · cloudflare zero trust docs* [Cloudflare docs]. (2024, agosto 27). Recuperato marzo 28, 2025, da <https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/do-more-with-tunnels/trycloudflare/sentence-transformers/all-mpnet-base-v2>
- sentence-transformers/all-mpnet-base-v2 · Hugging Face*. (2024, gennaio 5). Recuperato marzo 28, 2025, da <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>
- sentence-transformers/LaBSE · Hugging Face*. (2021). Recuperato marzo 28, 2025, da <https://huggingface.co/sentence-transformers/LaBSE>
- sentence-transformers/paraphrase-mpnet-base-v2 · Hugging Face*. (2022). Recuperato marzo 28, 2025, da <https://huggingface.co/sentence-transformers/paraphrase-mpnet-base-v2>
- SentenceTransformers Documentation — Sentence Transformers documentation*. (2025). Recuperato marzo 28, 2025, da <https://sbert.net/>

- Smaller LaBSE · models · dataloop*. (2021). Recuperato marzo 28, 2025, da [https://dataloader.ai/library/model/setu4993\\_smaller-labse/](https://dataloader.ai/library/model/setu4993_smaller-labse/)
- spaCy · industrial-strength natural language processing in python*. (2025). Recuperato marzo 28, 2025, da <https://spacy.io/>
- Speech and Language Processing*. (2025). Recuperato marzo 28, 2025, da <https://web.stanford.edu/~jurafsky/slp3/>
- Streamlit • a faster way to build and share data apps*. (2021, gennaio 14). Recuperato gennaio 20, 2025, da <https://streamlit.io/>
- Text Encoding Initiative*. (1994). Recuperato gennaio 20, 2025, da <https://tei-c.org/>
- TOML: Tom's Obvious Minimal Language*. (2013). Recuperato marzo 28, 2025, da <https://toml.io/en/>
- W, I. (2021, giugno 17). *Google colab-getting started* [Clarusway]. Recuperato marzo 28, 2025, da <https://medium.com/clarusway/google-colab-d73da882360b>
- Welcome to python.org* [Python.org]. (2025, marzo 14). Recuperato marzo 28, 2025, da <https://www.python.org/>
- Yu, M. (2024, febbraio 8). *What is self-supervised contrastive learning?* [Medium]. Recuperato marzo 28, 2025, da <https://medium.com/@c.michael.yu/what-is-self-supervised-contrastive-learning-df3044d51950>