

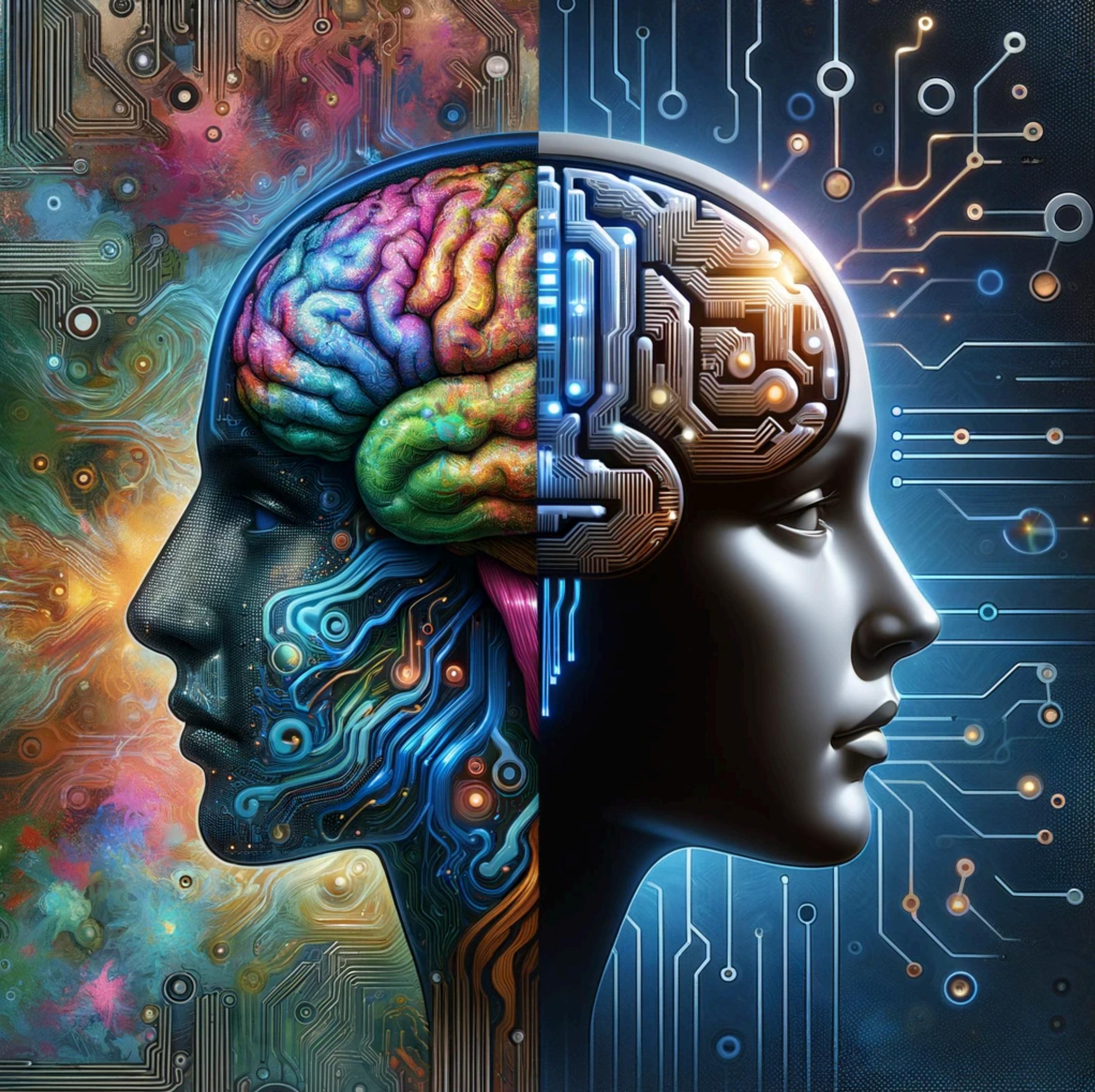


AI VS HUMAN

By Rosy Giudice, Anna Magaldi e Antonia Guerra

INDEX:

- Introduction and Project Objectives
- Data cleaning
- Firma Stilometrica
- EDA
- Modeling
- Reti neurali
- User interface del sistema AI detection
- AI detector database
- Docker container
- Conclusioni



INTRODUCTION

"UMANO O ALGORITMO?

PER QUESTO PROGETTO ABBIAMO MESSO SOTTO LA LENTE D'INGRANDIMENTO UN DATASET DI TESTI AI-GENERATED E HUMAN-WRITTEN.

L'OBIETTIVO?

CHI HA SCRITTO QUESTO TESTO? FACCIALE LE PULCI A STILE E GRAMMATICA PER SMASCHERARE I BOT E RITROVARE L'AUTENTICITÀ UMANA.



Data Cleaning

PREPROCESSING, GESTIONE DEGLI OUTLIER E NEUTRALIZZAZIONE DEI BIAS SEMANTICI

INTEGRITÀ DEL DATO: ZERO DATA-LEAKAGE

Neutralizzazione del lessico AI

- Rimozione di ogni riferimento esplicito ai modelli (GPT, LLM) per forzare il sistema ad analizzare lo stile e non le parole chiave

Eliminazione dei Pattern Boilerplate

- Eliminazione delle frasi standardizzate che l'AI usa come 'stampini' strutturali, riducendo il rischio che il modello impari scorciatoie statistiche

Validazione Statistica (Feature Sentinel)

- Implementazione di un sistema di monitoraggio automatico che identifica feature con correlazione sospetta ($|r| > 0.7$) rispetto al target

Data Stabilization (Clipping degli Outlier)

- Utilizzo della tecnica di Clipping basata sul range interquartile (IQR) per stabilizzare le metriche numeriche (lunghezza, sentiment, punteggiatura)

```
print("Leakage Semantico: keyword-label dependency")

suspicious_tokens = [
    'ai', 'language model', 'generated', 'human',
    'chatgpt', 'gpt', 'llm'
]

def contains_suspicious_terms(text):
    text = text.lower()
    return any(tok in text for tok in suspicious_tokens)

# assegnazione a colonna del DataFrame
df_clean['leakage_flag'] = df['text_cleaned'].apply(contains_suspicious_terms)

def clean_text_for_nlp(text):
    # 1. Rimozione marker di fonte interna e rumore tipico dell'AI
    text = re.sub(r'\|\|', '', text)

    # 2. Rimozione boilerplate (frasi standard che l'AI usa per iniziare)
    # Questi sono pattern comuni che confondono i modelli di tokenizzazione
    boilerplate_patterns = [
        r'\banalysis indicates that\b',
        r'\bthe following summary on\b',
        r'\bthis article discusses\b',
        r'\bas someone who follows\b',
        r'\bi recently experienced\b',
        r'\bin my experience,\?\b',
        r'\bbased on the data,\?\b',
        r'\bcommunity feedback\b',
        r'\bresearch suggests\b',
        r'\baccording to the data\b'
    ]

    # Feature Engineering: AI
    ai_patterns_detect = [
        'community response', 'research suggests', 'according to the data',
        'analysis indicates', 'this article discusses', 'the following summary'
    ]

    def count_ai_patterns(text):
        text_lower = text.lower()
        return sum([1 for pattern in ai_patterns_detect if pattern in text_lower])

    df['ai_pattern_count'] = df['text'].apply(count_ai_patterns)
    print(f" - Feature 'ai_pattern_count' creata (AI avg: {df[df['label']=='ai']['ai_pattern_count'].mean():.2f})")

    print("## 5. Gestione Outlier (clipping IQR)")
    # Definiamo le colonne numeriche critiche (quelle che useremo nel modello)
    col_num_critical = ['length_words', 'quality_score', 'sentiment', 'plagiarism_score']

    for col in col_num_critical:
        # Calcolo dei quantili e dell'IQR
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR_val = Q3 - Q1

        # Limiti di clipping (1.5 * IQR)
        lower_bound = Q1 - 1.5 * IQR_val
        upper_bound = Q3 + 1.5 * IQR_val

        # Applichiamo il clipping: limitiamo i valori anomali senza eliminarli
        df[col] = df[col].clip(lower_bound, upper_bound)
        print(f" - Colonna '{col}' clippata tra {lower_bound:.3f} e {upper_bound:.3f}")
```

ARCHITETTURA DELLE FEATURE

Analisi Strutturale: Oltre la Sintassi Lineare

Ritmo e respirazione

- Gli umani hanno un ritmo "respiratorio" irregolare, mentre l'AI tende a una distribuzione meccanica e costante dei segni di interpunkzione

Gerarchia sintattica

- l'AI produce spesso strutture lineari, l'uomo architetture nidificate

Melodia grammaticale

- Mappatura delle sequenze di parti del discorso (es. Verbo-Avverbio) come impronta stilistica unica e non semantica

Entropia informativa

- misura il grado di "sorpresa" nelle scelte linguistiche

```
def extract_deep_stylistic_features(text):  
    # 0. Analisi preliminare (Usiamo il testo originale per la punteggiatura)  
    doc = nlp(text) # Rimosso .lower() qui per non perdere info sulle maiuscole subito  
  
    # Liste di supporto  
    words = [token.text.lower() for token in doc if token.is_alpha]  
    sentences = [sent.text.strip() for sent in doc.sents if len(sent.text.strip()) > 0]  
    pos_tags = [token.pos_ for token in doc]  
  
    features = {}  
  
    # ---  
    # 1. RITMO E VARIANZA (Punteggiatura e Respirazione)  
    # ---  
    # Varianza intervalli punteggiatura (Novità)  
    # Gli umani hanno pause irregolari, l'AI è costante.  
    punc_indices = [i for i, token in enumerate(doc) if token.is_punct]  
    if len(punc_indices) > 2:  
        intervals = np.diff(punc_indices)  
        features['punc_interval_variance'] = np.var(intervals)  
    else:  
        features['punc_interval_variance'] = 0.0  
  
    if len(sentences) > 1:  
        sent_lengths = [len(sent.split()) for sent in sentences]  
        features['sentence_length_cv'] = np.std(sent_lengths) / (np.mean(sent_lengths) + 1e-6)  
    else:  
        features['sentence_length_cv'] = 0.0  
  
    # ---  
    # 2. SINTASSI PURA (POS N-Grams Diversity)  
    # ---  
    # Diversità dei bigrammi POS (Novità)  
    # Misura quanto è varia la "melodia" grammaticale  
    if len(pos_tags) > 1:  
        pos_bigrams = [f'{pos_tags[i]_{pos_tags[i+1]}' for i in range(len(pos_tags)-1)]  
        features['pos_ngram_diversity'] = len(set(pos_bigrams)) / (len(pos_bigrams) + 1e-6)  
    else:  
        features['pos_ngram_diversity'] = 0.0  
  
    # Entropia POS (Esistente)  
    pos_counts = Counter(pos_tags)  
    pos_probs = np.array([count / len(pos_tags) for count in pos_counts.values()])  
    features['pos_entropy'] = entropy(pos_probs)  
  
    # ---  
    # 3. COMPLESSITÀ ALBERO SINTATTICO  
    # ---  
    tree_depths = []  
    for sent in doc.sents:  
        max_depth = 0  
        for token in sent:  
            depth = 1  
            curr = token  
            while curr.head != curr:  
                depth += 1  
                curr = curr.head  
            max_depth = max(max_depth, depth)  
        tree_depths.append(max_depth)  
  
    features['avg_tree_depth'] = np.mean(tree_depths) if tree_depths else 0.0  
    features['tree_depth_std'] = np.std(tree_depths) if len(tree_depths) > 1 else 0.0
```

Dinamica Lessicale e Performance

Marcatori antropometrici e ottimizzazione su larga scala

Evoluzione del lessico

Monitoraggio della ricchezza del vocabolario dinamico (finestre mobili): identifica la naturale variazione del registro umano contro la staticità dell'AI

Marcatori di formalità e stile

Analisi della densità di sostantivi e particelle grammaticali "invisibili": lo scheletro strutturale che differenzia la macchina dall'uomo

Autenticità naturale

Rilevamento di segnali di linguaggio informale e contrazioni: indicatori chiave di una genesi testuale non sintetica

Lemmatizzazione conservativa

Riduzione delle parole alla loro radice (es. "correndo" -> "correre") mantenendo però la struttura sintattica

```
# -----
# 4. RIPETIZIONE (Windowed TTR)
# -----
window_size = 50
ttr_values = []
for i in range(0, len(words) - window_size, window_size):
    window = words[i:i+window_size]
    ttr_values.append(len(set(window)) / len(window))
features['ttr_variance'] = np.var(ttr_values) if len(ttr_values) > 1 else 0.0

# -----
# 5. FUNCTION WORDS & FORMALITY
# -----
function_words = {'the', 'a', 'an', 'of', 'to', 'in', 'for', 'on', 'at', 'by', 'with', 'from'}
features['function_word_ratio'] = sum(1 for w in words if w in function_words) / (len(words) + 1e-6)

nominalizations = ['tion', 'ment', 'ness', 'ity', 'ance', 'ence']
features['nominalization_rate'] = sum(1 for w in words if any(w.endswith(s) for s in nominalizations)) / (len(words) + 1e-6)

# -----
# 6. MARKERS UMANI (Contrazioni ed Ellissi)
# -----
# Le contrazioni sono forti indicatori umani
contractions = ["n't", "'m", "'re", "'ve", "'ll", "'d", "'s"]
features['contraction_rate'] = sum(text.lower().count(c) for c in contractions) / (len(words) + 1e-6)
features['ellipsis_count'] = text.count('...')

return pd.Series(features)
```

Infrastruttura: uso della GPU per processare migliaia di righe in pochi secondi (scalabilità)

Stilometria: A differenza dei modelli base, la tua pipeline non distrugge le "stop-words" (articoli, preposizioni), perché sai che nell'AI detection sono proprio quelle a tradire la macchina

ARCHITETTURA DELLE FEATURES

Analisi Strutturale: Oltre la Sintassi Lineare

Varianza degli intervalli di punteggiatura

- Misura la regolarità delle pause. Gli umani hanno un ritmo "respiratorio" irregolare, mentre l'AI tende a una distribuzione meccanica e costante dei segni di interpunkzione

Coefficiente di Variazione della Lunghezza delle Frasi

- Analizza l'alternanza tra periodi brevi e complessi. Un alto CV indica una firma umana autentica, contrapposta alla monotonia strutturale tipica dei modelli linguistici

Profondità Media dell'Albero Sintattico

- Utilizza il parsing di dipendenza per misurare la complessità gerarchica della frase. Identifica la "linearità" sintattica degli output generati

Diversità dei Bigrammi POS

- Mappa la "melodia grammaticale" calcolando la varietà delle sequenze di parti del discorso (es. Verbo-Avverbio). È un'impronta stilistica profonda difficile da emulare

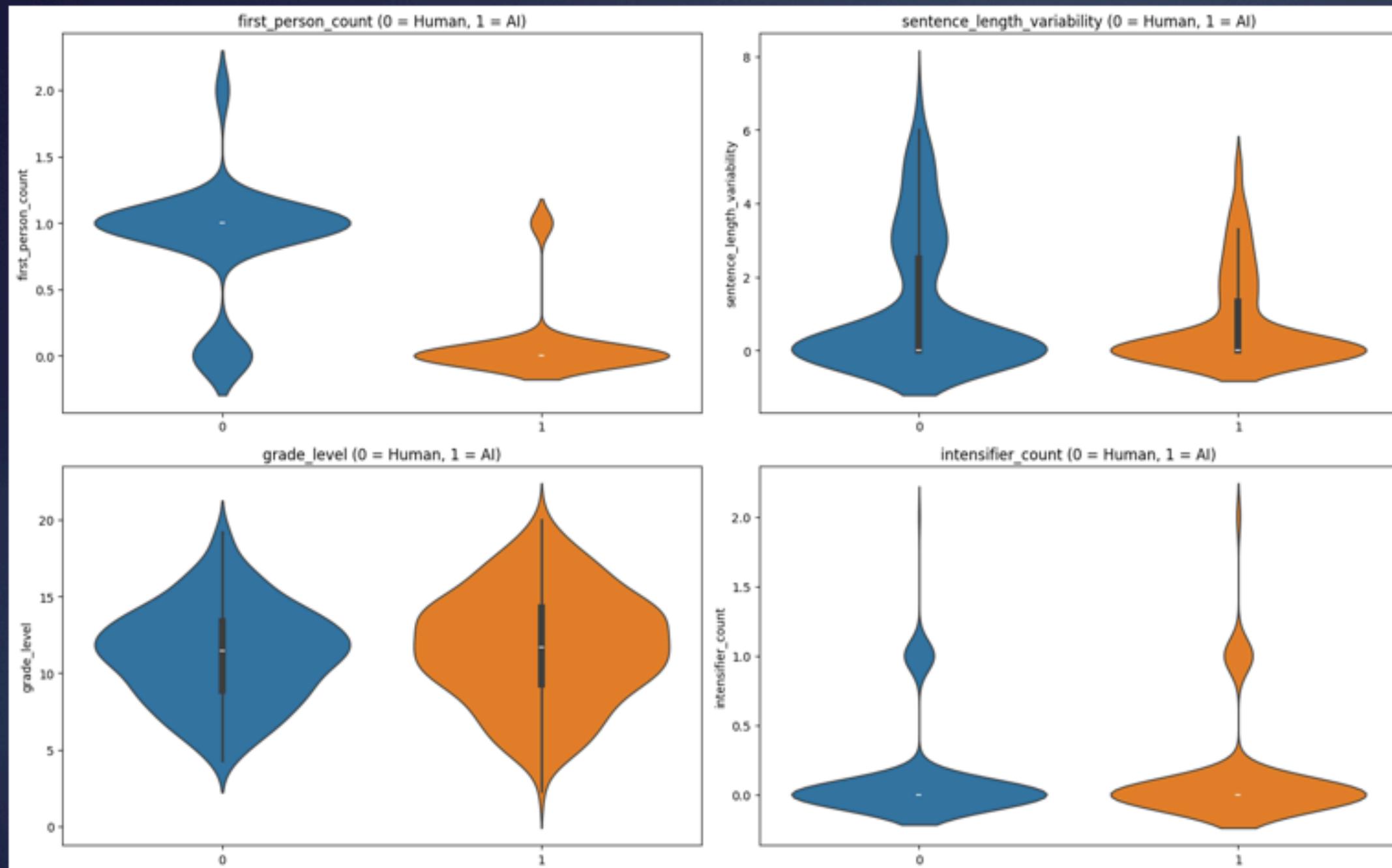
Entropia delle Parti del Discorso

- Applica la teoria dell'informazione per misurare la prevedibilità delle scelte grammaticali dell'autore

Firma Stilometrica

Analisi delle caratteristiche stilistiche per il riconoscimento di testi umani e generati da AI

Distribuzione delle Feature Stilometriche (AI vs Human)



Le feature sono state scelte perché **class-agnostic** e indipendenti dal contenuto semantico

Distribuzione delle Feature Stilometriche (AI vs Human)

C — Cognitive Load Distribution

```
def cognitive_features(doc):
    """Firma cognitiva: complessità lettura"""
    readability = []
    for sent in doc.sents:
        try:
            score = flesch_reading_ease(sent.text)
            readability.append(score)
        except:
            pass

    sub_clauses = sum(1 for tok in doc if tok.dep_ in {"advcl", "ccomp", "xcomp",
sentences = len(list(doc.sents))

    return {
        "readability_oscillation": np.std(readability) if readability else 0.0,
        "clause_density": sub_clauses / sentences if sentences > 0 else 0.0
    }
```

Python

R — Rhythmic Control

```
def rhythmic_features(doc):
    """Firma del ritmo: variabilità lunghezza frasi"""
    lengths = sentence_lengths(doc)
    if len(lengths) == 0:
        return {"sentence_length_cv": 0.0, "burstiness_index": 0.0}

    return {
        "sentence_length_cv": coefficient_of_variation(lengths),
        "burstiness_index": burstiness_index(lengths)
    }
```

Separazione linguistica: stile umano vs stile AI

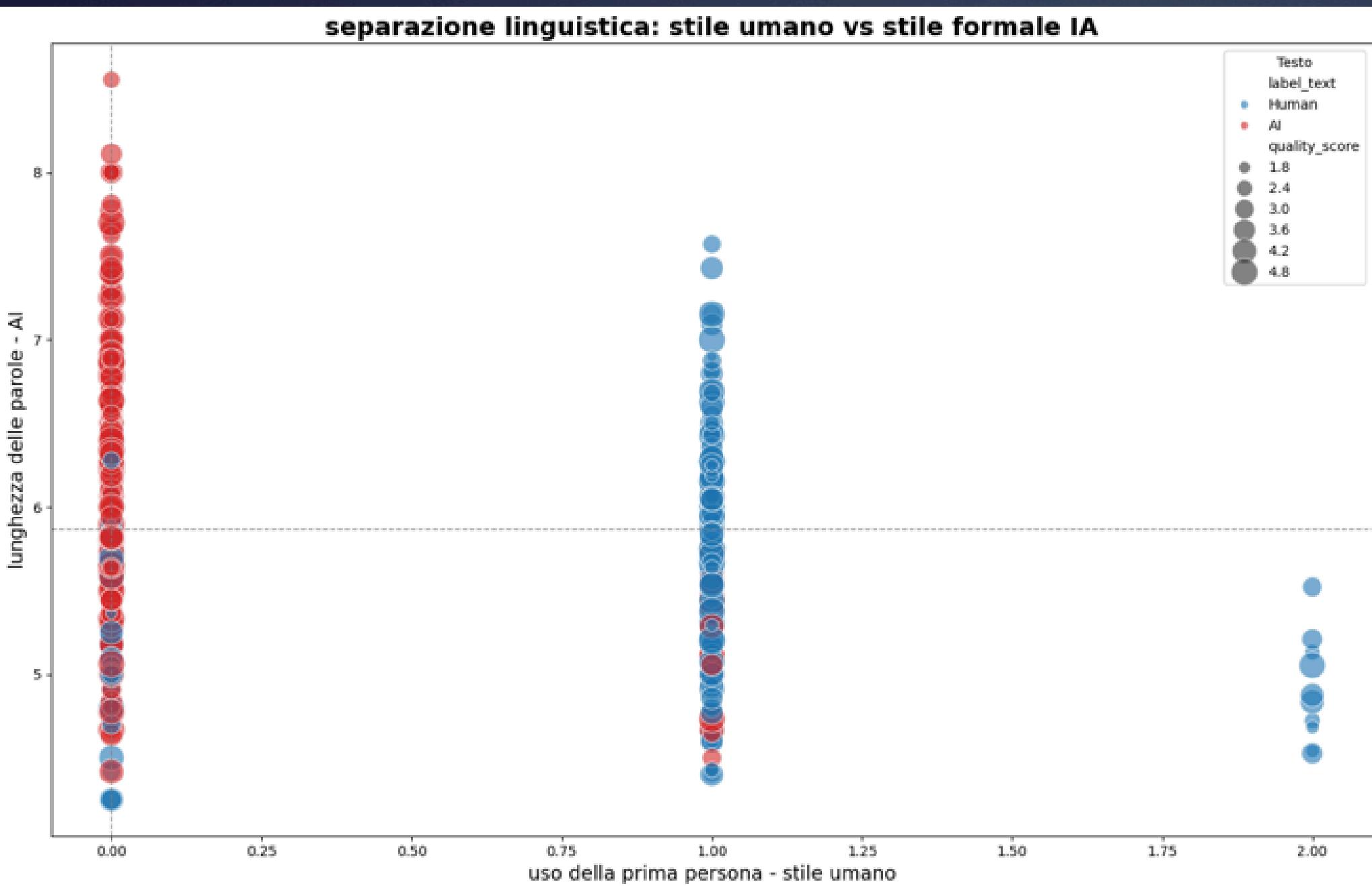


Grafico
• Ogni punto è un testo
• Le coordinate rappresentano scelte stilistiche, non il contenuto

Asse X – Uso della prima persona
• Valori alti → stile soggettivo (umano)
• Valori bassi → stile impersonale (AI)

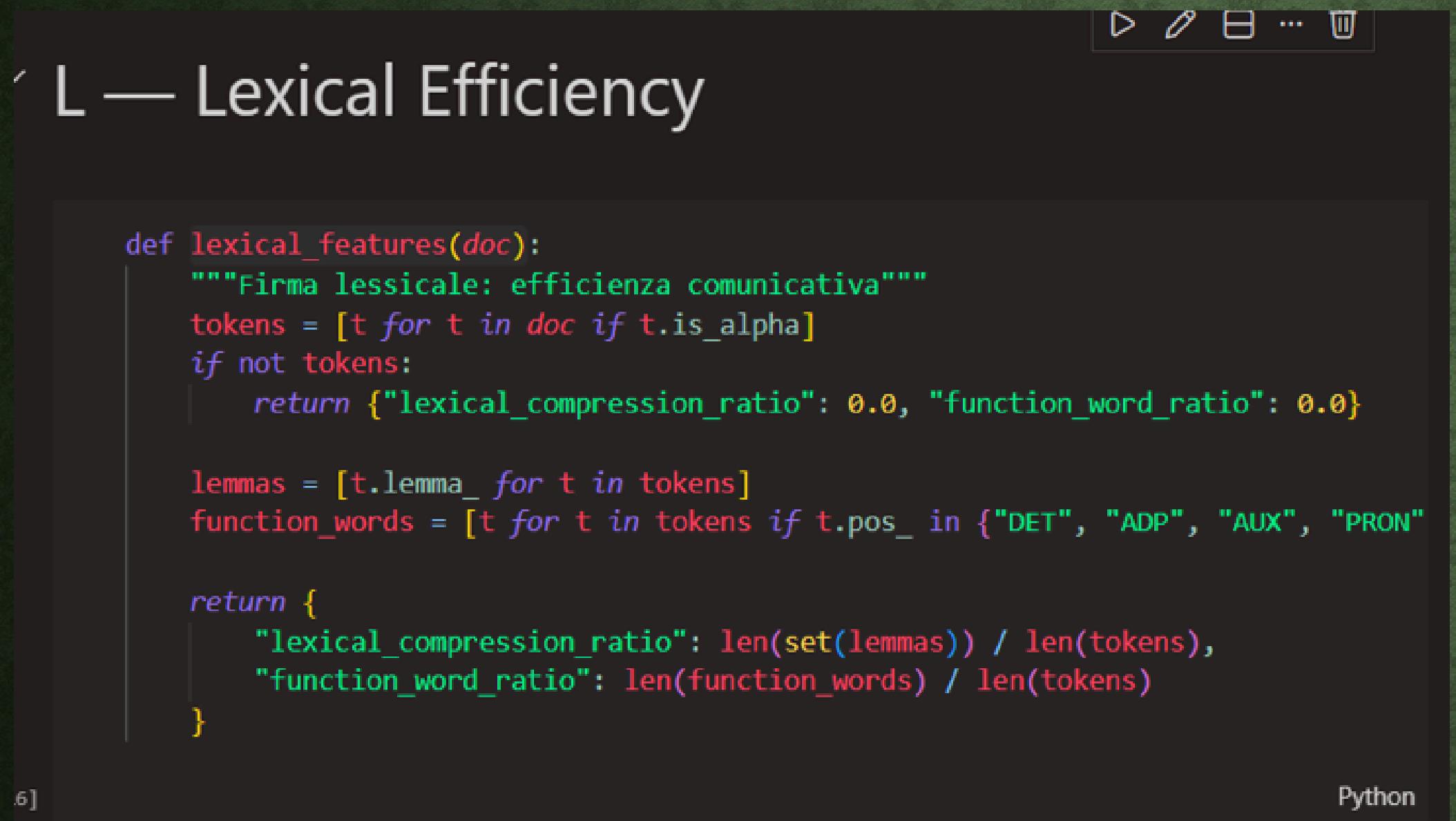
Asse Y – Lunghezza media delle parole
• Valori bassi → linguaggio semplice
• Valori alti → linguaggio formale e tecnico (AI)

Pattern osservati
• Umano: più prima persona, parole più brevi
• AI: meno soggettività, parole più lunghe
• Sovrapposizione limitata

Quality Score
• I testi AI più formali mostrano qualità tecnica più alta

Questo grafico mostra che l'AI non scrive "male":
scrive in un'altra regione del linguaggio.

Separazione linguistica: stile umano vs stile AI



The image shows a screenshot of a dark-themed code editor. At the top, there is a toolbar with icons for file operations like Open, Save, and Print. Below the toolbar, the title bar displays "L — Lexical Efficiency". The main area of the editor contains the following Python code:

```
def lexical_features(doc):
    """Firma lessicale: efficienza comunicativa"""
    tokens = [t for t in doc if t.is_alpha]
    if not tokens:
        return {"lexical_compression_ratio": 0.0, "function_word_ratio": 0.0}

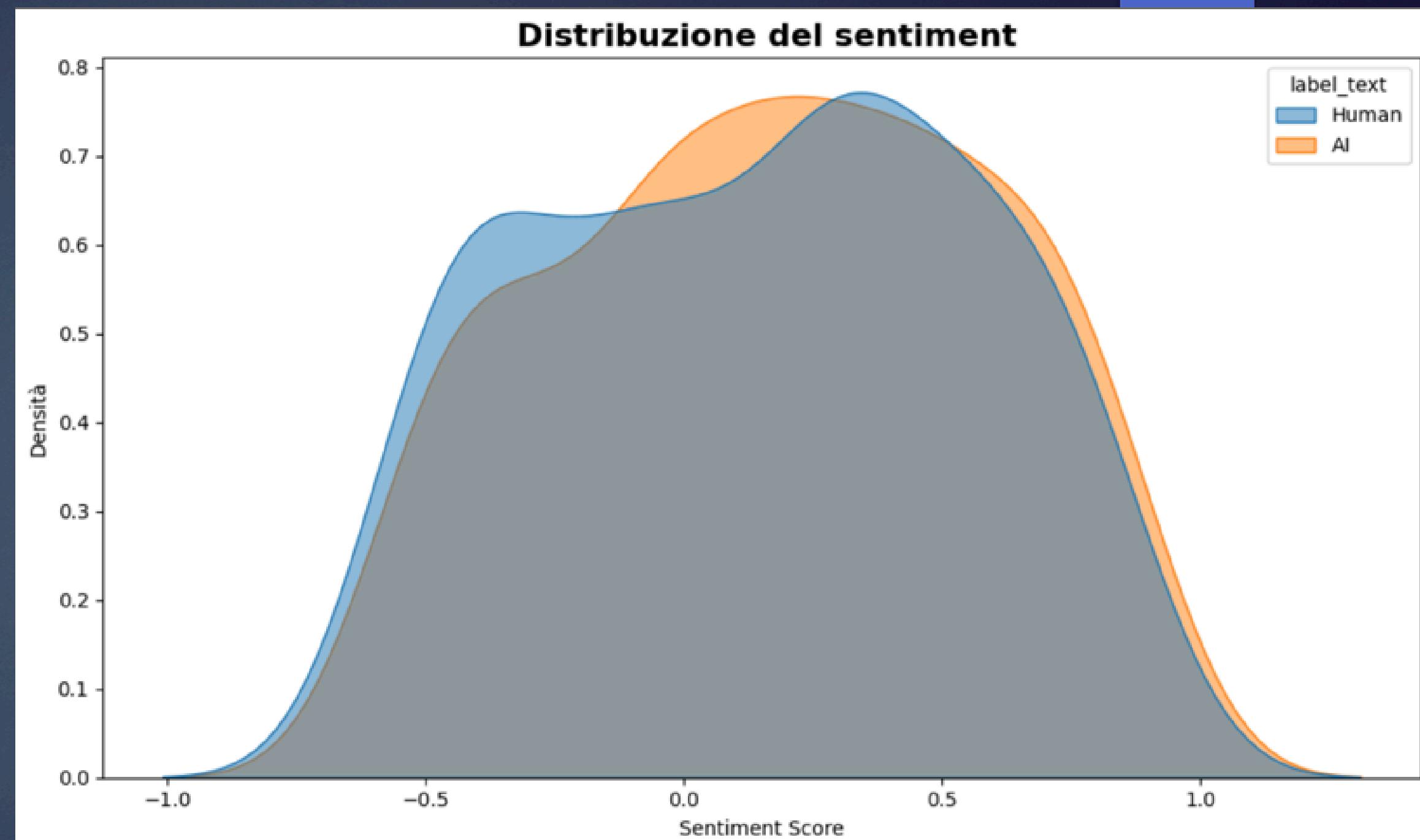
    lemmas = [t.lemma_ for t in tokens]
    function_words = [t for t in tokens if t.pos_ in {"DET", "ADP", "AUX", "PRON"}]

    return {
        "lexical_compression_ratio": len(set(lemmas)) / len(tokens),
        "function_word_ratio": len(function_words) / len(tokens)
    }
```

In the bottom right corner of the code editor, the word "Python" is displayed, indicating the language of the code.

Distribuzione del Sentiment: Umano vs AI

- il sentiment da solo non basta a distinguere AI e umano.
- Però rivela un bias: l'AI tende a rimanere neutra o positiva, mentre l'umano è più emotivamente instabile
- Il sentiment è un segnale debole
- Utile solo come feature di supporto, non decisionale
- Questo conferma che il valore reale del modello non è semantico, ma stilometrico



E — Emotional Variance

```
def fast_sentiment(text):
    """Sentiment rapido con TextBlob"""
    try:
        return TextBlob(text).sentiment.polarity
    except:
        return 0.0

def emotional_features(doc):
    """Firma emotiva: varianza polarità"""
    sentiments = []
    for sent in doc.sents:
        score = fast_sentiment(sent.text)
        sentiments.append(score)

    return {"sentiment_variance": np.std(sentiments) if sentiments else 0.0}
```

1. **Analizza ogni frase del testo**
2. Calcola per ogni frase un punteggio emotivo
 1. negativo → emozioni negative
 2. positivo → emozioni positive
3. Osserva quanto varia il sentiment all'interno del testo
Non guarda *che emozione*, ma quanto cambia nel tempo

Distribuzione del Sentiment: Umano vs AI

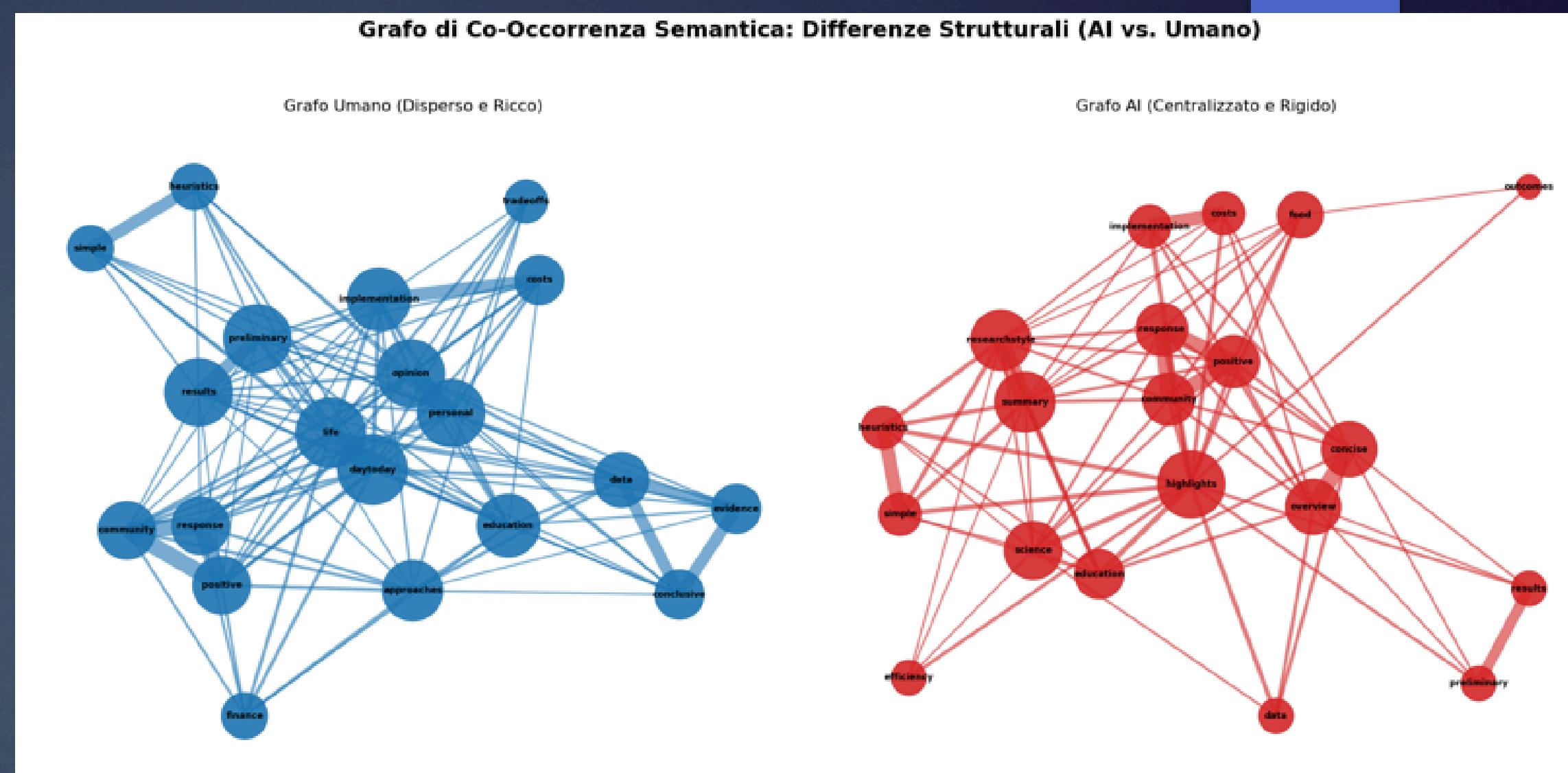
Grafo umano (sx)

- le parole sono collegate in modo più disperso e irregolare
- Questo riflette il modo umano di scrivere, che è più narrativo, personale e meno pianificato

Grafo AI (dx)

- connessioni sono più centralizzate attorno a pochi nodi chiave
- L'AI tende a organizzare il discorso in modo più ordinato e ripetibile, usando strutture simili

Grafo di Co-Occorrenza Semantica: Differenze Strutturali (AI vs. Umano)



Anche quando il contenuto sembra “umano”, la forma del linguaggio tradisce l’origine artificiale

Melodia Grammaticale e coesione

S — Syntactic Entropy

```
def pos_bigram_entropy(doc):
    """Entropia delle transizioni POS (Part-of-Speech)"""
    pos_tags = [token.pos_ for token in doc]
    if len(pos_tags) < 2:
        return 0.0

    bigrams = list(pairwise(pos_tags))
    counts = Counter(bigrams)
    return safe_entropy(counts)

def dependency_depth_mean(doc):
    """Profondità media albero sintattico"""
    depths = []
    for sent in doc.sents:
        for token in sent:
            depth, current = 0, token
            while current.head != current:
                depth += 1
                current = current.head
            depths.append(depth)
    return np.mean(depths) if depths else 0.0

def syntactic_features(doc):
    """Firma sintattica: complessità strutturale"""
    return {
        "pos_bigram_entropy": pos_bigram_entropy(doc),
        "dependency_depth_mean": dependency_depth_mean(doc)
    }
```

D — Discourse Regularization

```
def sentence_similarity_drift(doc):
    """Drift semantico tra frasi consecutive"""
    sentences = [sent.text for sent in doc.sents]
    if len(sentences) < 2:
        return 0.0

    vectors = np.array([nlp(sent).vector for sent in sentences])
    sims = [cosine_similarity([vectors[i]], [vectors[i + 1]])[0][0]
            for i in range(len(vectors) - 1)]
    return float(np.mean(sims))

def structural_redundancy(doc):
    """Ridondanza pattern sintattici"""
    patterns = []
    for sent in doc.sents:
        pattern = tuple(tok.dep_ for tok in sent)
        patterns.append(pattern)

    if not patterns:
        return 0.0

    counts = Counter(patterns)
    repeated = sum(c for c in counts.values() if c > 1)
    return repeated / len(patterns)

def discourse_features(doc):
    """Firma del discorso: coesione e ridondanza"""
    return {
```

HAPAX & TEMPLATE BIAS (IL DNA E LA GABBIA)

Queste sono due feature chiave.

- La densità di hapax misura quante parole compaiono una sola volta nel testo. È una vera impronta dell'autore (DNA).
- Il template bias, invece, misura quanto il testo segue strutture rigide, come liste, connettivi iperstrutturati e conclusioni eccessivamente standardizzata (gabbia invisibile della scrittura artificiale).

From Features to Geometry

```
def extract_stylometric_signature(text):
    """
    Estrae la firma stilometrica completa (13 feature).
    Ottimizzata per velocità e robustezza.
    """
    try:
        # Limita Lunghezza per performance
        text = str(text)[:10000]

        # Parsing spaCy
        doc = nlp(text)

        # Estrai tutte le feature
        features = {}
        features.update(rhythmic_features(doc))
        features.update(syntactic_features(doc))
        features.update(lexical_features(doc))
        features.update(discourse_features(doc))
        features.update(emotional_features(doc))
        features.update(cognitive_features(doc))

        # Feature avanzate (non richiedono doc spaCy)
        features['hapax_density'] = get_hapax_density(text)
        features['template_bias_score'] = get_template_bias(text)

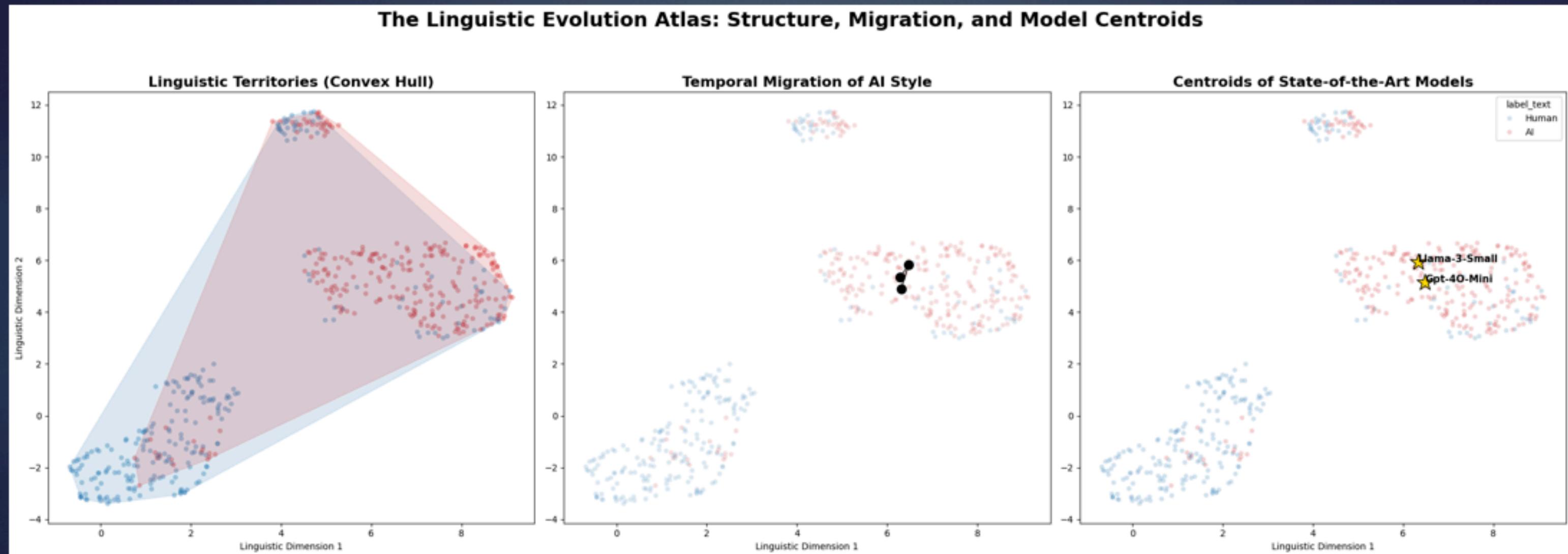
    return pd.Series(features)
```

- Ogni testo viene rappresentato come un vettore stilometrico
- Le feature catturano come il testo è scritto, non cosa dice
- Testi con stili simili risultano vicini nello spazio
- La geometria rende visibile la separazione tra umano e IA

Linguistic Space Mapping (UMAP)

Mettendo insieme le singole feature: ritmo, sintassi, lessico ed emozione, ogni testo diventa un punto in uno spazio multidimensionale:

- la distanza rappresenta la somiglianza stilistica.
- questo ci permette di osservare se i testi umani e quelli dell'IA occupano regioni diverse dello spazio.



Linguistic Space Mapping (UMAP)

```
# Applica con progress bar
tqdm.pandas(desc="Processing texts")
signature_df = df_clean['text_cleaned'].progress_apply(extract_stylometric_signature)

print(f"\n Signature estratte: {signature_df.shape[1]} feature")
print(f" Feature list: {', '.join(signature_df.columns)}\n")

# Merge con dataset originale
print("[Merge con dataset originale...]")
df_final = pd.concat([df_clean, signature_df], axis=1)
print(f" Dataset finale: {df_final.shape[0]} righe, {df_final.shape[1]} colonne\n")
```

Testo

extract_stylometric_signature

vettore di 13 feature

UMAP

spazio 2D



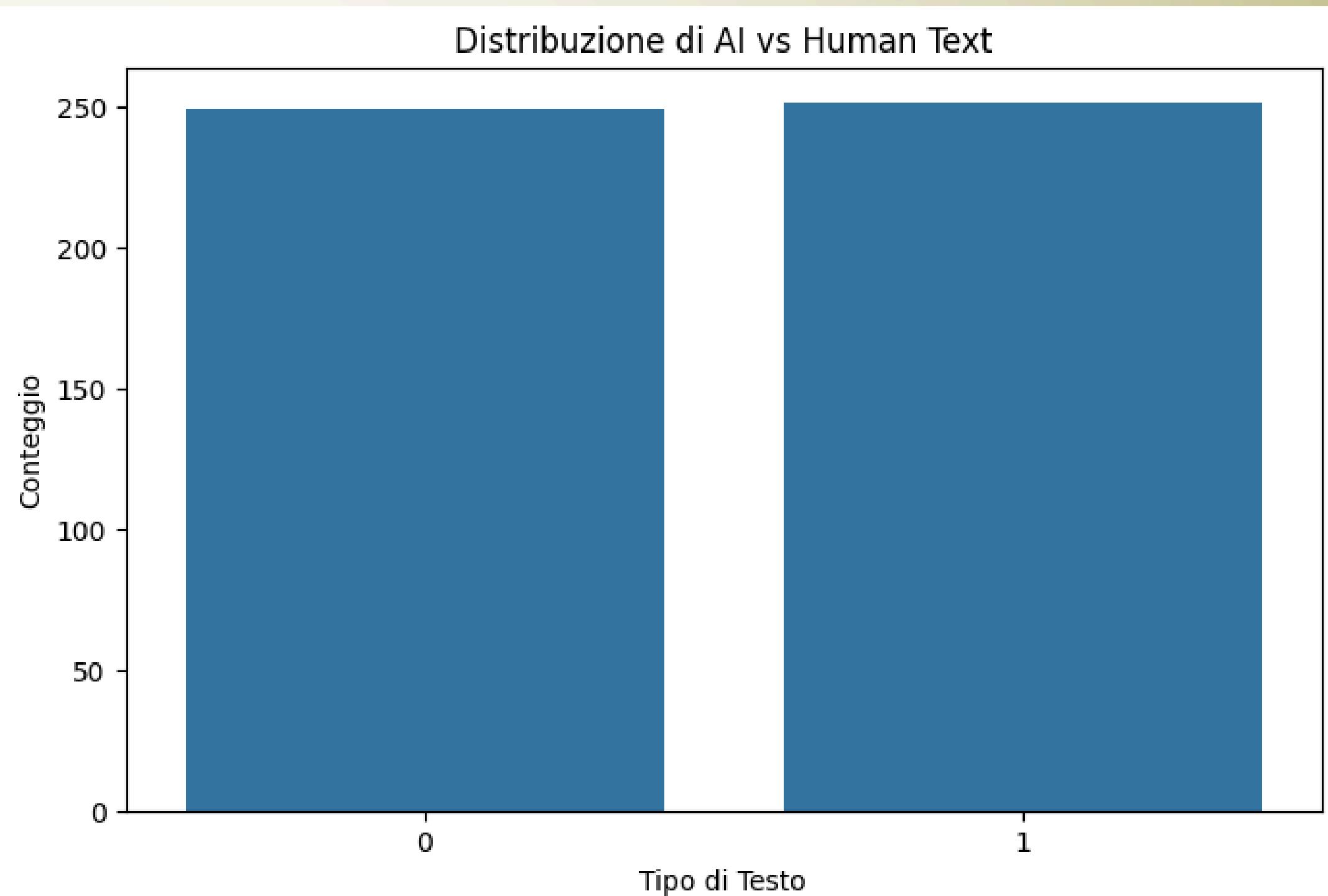
EDA

ANALISI ESPLORATIVA DEI DATI

Grafico a barre

Questo grafico conta quanti testi sono stati scritti da umani e quanti sono stati generati da un'intelligenza artificiale. Serve a verificare se il dataset è bilanciato, ovvero se abbiamo un numero simile di esempi per entrambe le categorie.

```
# Visualizza la distribuzione della colonna 'label'  
plt.figure(figsize=(8, 5))  
sns.countplot(x='label_encoded', data=df)  
plt.title('Distribuzione di AI vs Human Text')  
plt.xlabel('Tipo di Testo')  
plt.ylabel('Conteggio')  
plt.show()
```

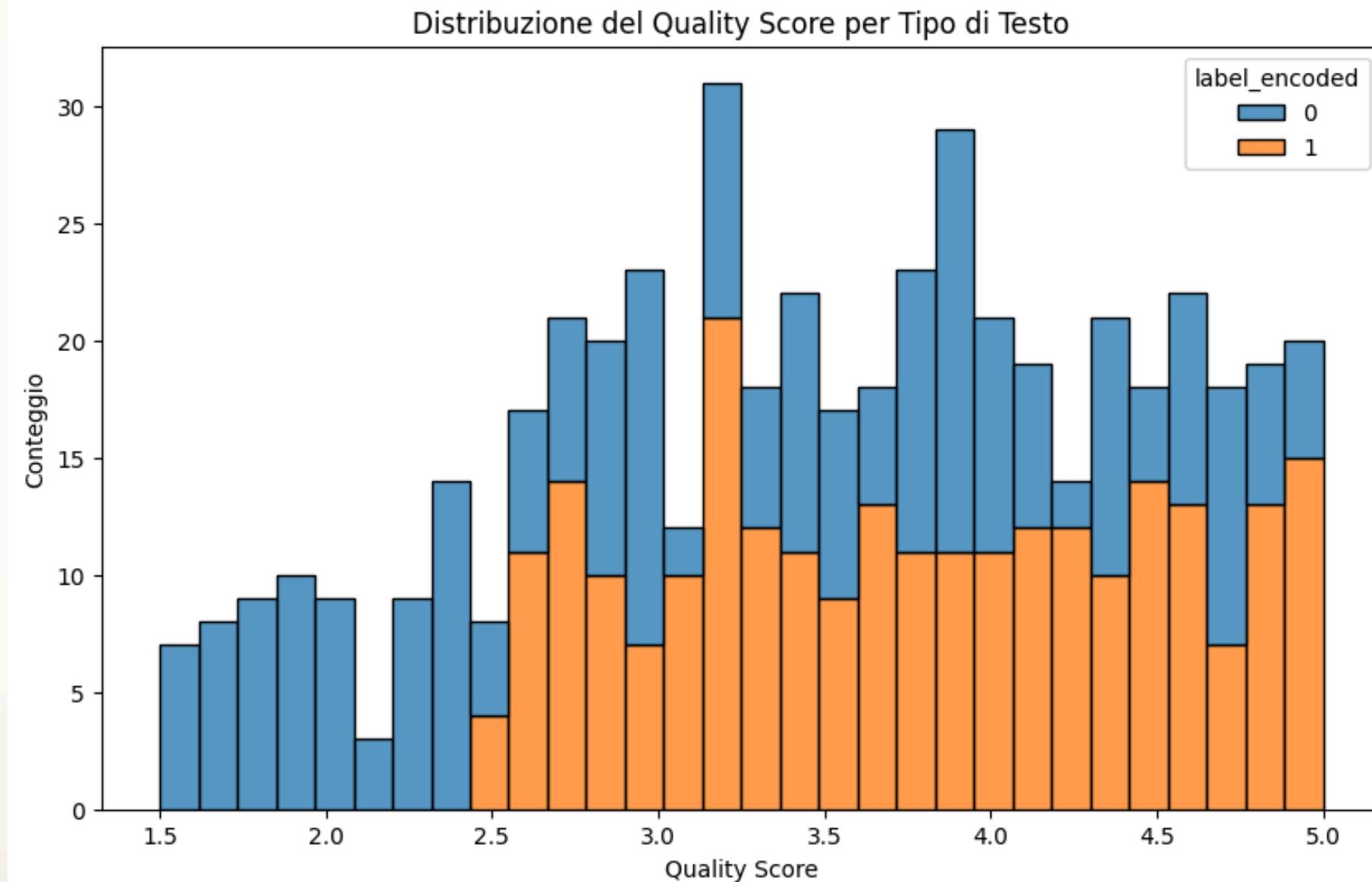


Histplot

Confronta la distribuzione dei punteggi di qualità (quality_score) per i testi umani e quelli generati dall'IA.

Aiuta a capire se ci sono differenze sistematiche nella qualità percepita tra i due tipi di testo. Per esempio, potremmo scoprire che i testi umani tendono ad avere punteggi di qualità più alti o più bassi.

```
# Visualizza la distribuzione del quality_score per ogni label  
plt.figure(figsize=(10, 6))  
sns.histplot(data=df, x='quality_score', hue='label_encoded', multiple='stack', bins=30)  
plt.title('Distribuzione del Quality Score per Tipo di Testo')  
plt.xlabel('Quality Score')  
plt.ylabel('Conteggio')  
plt.show()
```

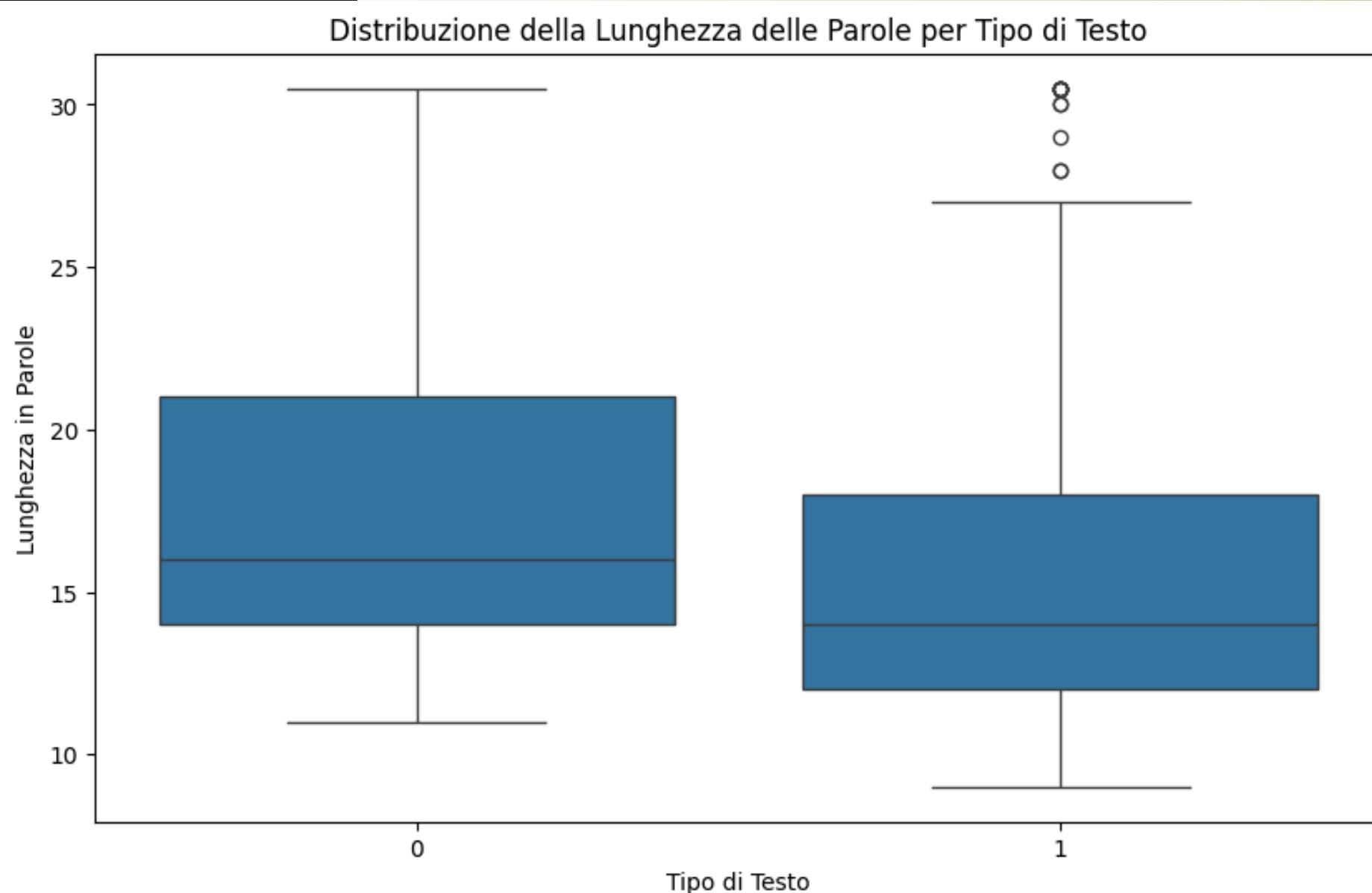


BoxPlot

Confronta la distribuzione della lunghezza dei testi, basato sul numero di parole (length_words),, tra le due categorie.

Il boxplot evidenzia la mediana, i quartili e gli eventuali outlier (valori anomali), permettendo di capire se, in media, i testi generati dall'IA sono più lunghi o più corti di quelli umani.

```
# Boxplot per 'length_words' vs 'label'  
plt.figure(figsize=(10, 6))  
sns.boxplot(x='label_encoded', y='length_words', data=df)  
plt.title('Distribuzione della Lunghezza delle Parole per Tipo di Testo')  
plt.xlabel('Tipo di Testo')  
plt.ylabel('Lunghezza in Parole')  
plt.show()
```



Heatmap: Matrice di Correlazione delle Features Numeriche

Aggiungi corpo del testo più freddi (es. blu) indicano una correlazione negativa forte.

Aiuta a identificare quali variabili sono legate tra loro. Ad esempio, una forte correlazione tra length_words e length_chars è prevedibile.

Relazioni meno ovvie possono essere spunti per analisi più approfondite.

```
# Correlazione tra le features numeriche
numerical_features = df.select_dtypes(include=np.number)
columns = ['length_chars', 'length_words', 'quality_score', 'sentiment', 'plagiarism_score', 'label_encoded']
correlation_matrix = numerical_features[columns].corr()

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Matrice di Correlazione delle Features Numeriche')
plt.show()
```

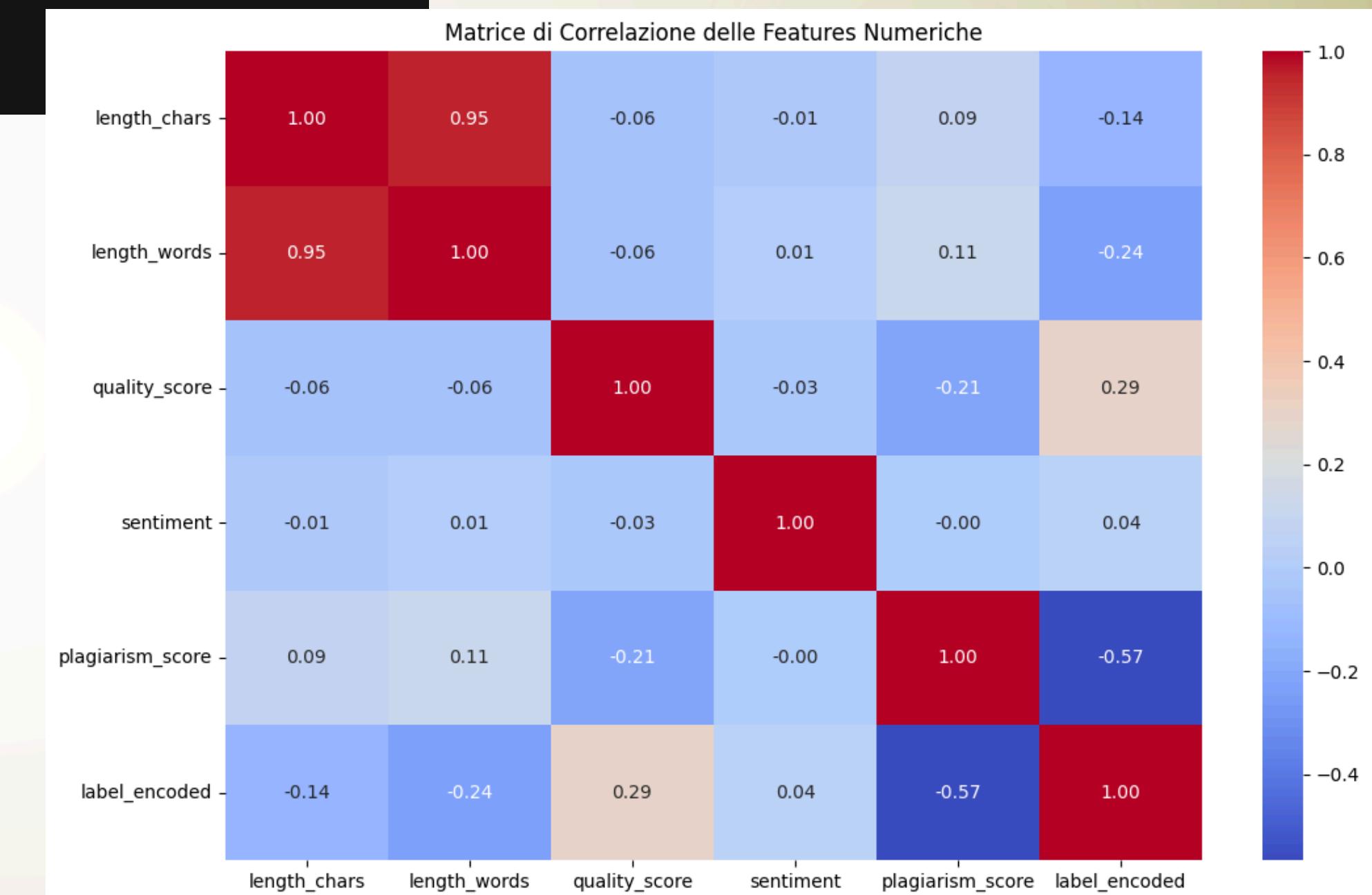
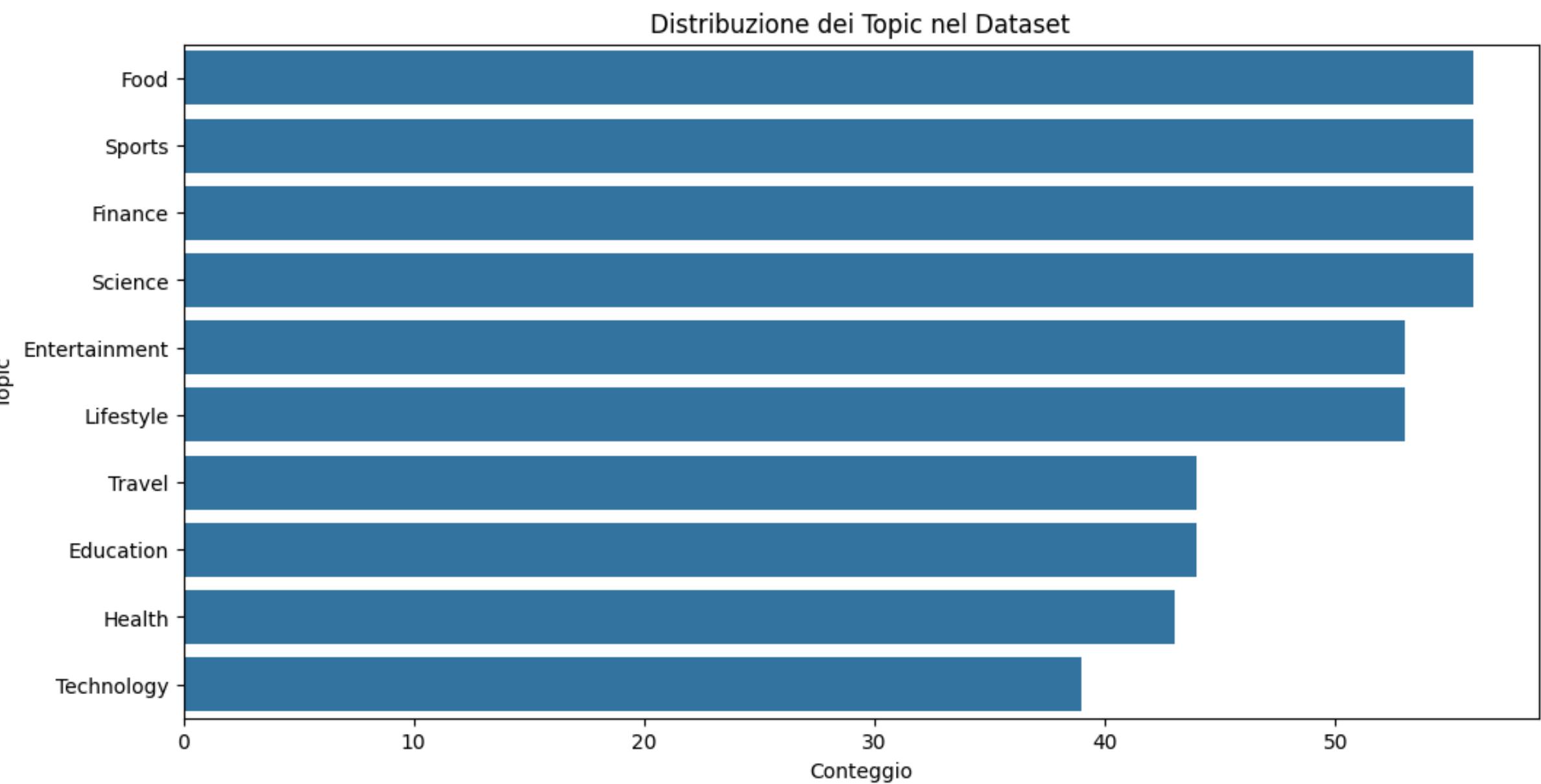


Grafico a Barre Orizzontali: Distribuzione dei Topic nel Dataset

Fa un conteggio di quanti testi appartengono a ciascun argomento (topic).

Permette di vedere quali sono gli argomenti più comuni nel dataset e se la loro distribuzione è uniforme.

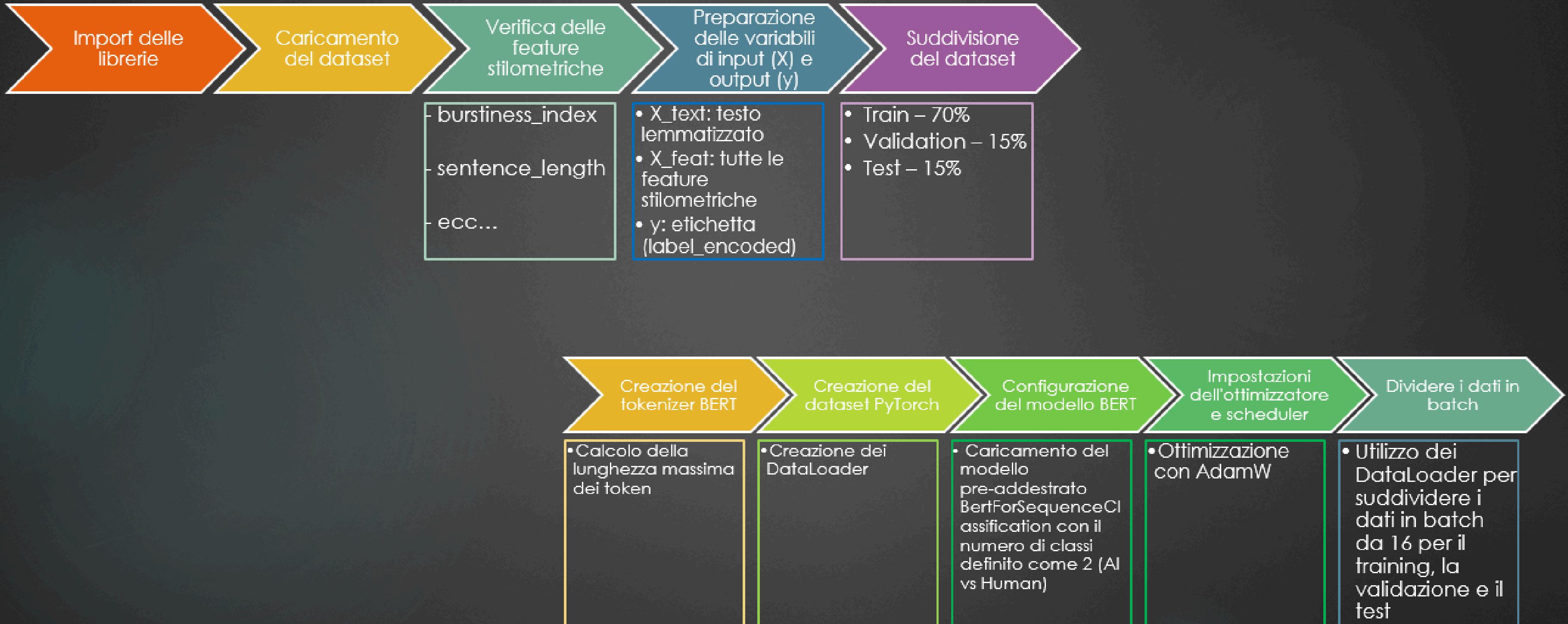
```
# Visualizza la distribuzione dei topic
plt.figure(figsize=(12, 6))
sns.countplot(y='topic', data=df, order = df['topic'].value_counts().index)
plt.title('Distribuzione dei Topic nel Dataset')
plt.xlabel('Conteggio')
plt.ylabel('Topic')
plt.show()
```



Modeling

SVILUPPO DI MODELLI IBRIDI: INTEGRAZIONE DI SEMANTICA E STILOMETRIA PER IL RICONOSCIMENTO DEL TESTO

Preprocessing del Dataset e Creazione delle Feature



BERT: Architettura e Rappresentazione del Testo

- Modello Base:
 - Utilizzo di bert-base-uncased pre-addestrato (12 layer, 768 hidden size).
- Pre-processing:
 - Tokenizzazione: BertTokenizer con un vocabolario di ~30k token.
 - Analisi Lunghezza: Il 90° percentile dei testi è di 718 token, ma sono stati troncati a 512 token (limite architettonico di BERT) per efficienza.
- Rappresentazione:
 - Il testo viene convertito in input_ids e attention_mask.
 - Il token speciale [CLS] viene usato per catturare la rappresentazione semantica globale della frase, essenziale per la classificazione.

```
import numpy as np
import pandas as pd
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader

from transformers import (
    BertTokenizer,
    BertForSequenceClassification,
    get_linear_schedule_with_warmup
)
from torch.optim import AdamW # Moved AdamW import here

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score,
    precision_recall_fscore_support,
    roc_auc_score,
    confusion_matrix
)
```

BERT FINE TUNING

- Strategia: Addestramento end-to-end per adattare i pesi di BERT alla distinzione "Human vs AI".
- Configurazione Training:
 - Optimizer: AdamW con learning rate 2e-5.
 - Epochs: 3.
 - Batch Size: 16.
 - Loss Function: Cross-Entropy (gestita internamente da BertForSequenceClassification).
- Risultati (Test Set):
 - Accuracy: 96.48%
 - F1-Score: 0.9645
 - Il modello mostra una convergenza molto rapida, con un'accuracy di validazione che tocca il 98.6% già alla prima epoca.

```
optimizer = AdamW(model_bert.parameters(), lr=2e-5, eps=1e-8)
epochs = 3
total_steps = len(train_loader) * epochs
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,
    num_training_steps=total_steps)

# Training function
def train_epoch(model, dataloader, optimizer, scheduler, device):
    model.train()
    total_loss = 0
    for batch in tqdm(dataloader, desc="Training"):
        optimizer.zero_grad()
        outputs = model(
            input_ids=batch['input_ids'].to(device),
            attention_mask=batch['attention_mask'].to(device),
            labels=batch['labels'].to(device)
        )
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()

    return total_loss / len(dataloader)
```

BERT Test Performance:
Accuracy: 0.9675
Precision: 0.9694
Recall: 0.9675
F1-Score: 0.9677

BERT-HYBRID + STYLOMETRIC

- Combinare le capacità semantiche di BERT con metriche stilistiche esplicite.
- Costruzione del Vettore Ibrido (781 dimensioni):
 - BERT Embeddings (768 dim): Estrazione del vettore [CLS] dal modello pre-addestrato.
 - Feature Stilometriche (13 dim): Aggiunta di metriche come Entropy, Burstiness, Sentence Length CV.
- Classificatore Finale: Random Forest addestrato su questi vettori concatenati.
- Prestazioni Superiori:
 - Accuracy: 97.02% (Migliore di BERT puro).
 - ROC-AUC: 0.9981.
 - Questo approccio corregge gli errori marginali di BERT sfruttando pattern statistici "umani" che il modello neurale potrebbe perdere.

```
# =====
# 5. HYBRID MODEL (BERT + Stylometric Features)
# =====

print("=" * 70)
print(" MODEL 2: HYBRID (BERT + Stylometric)")
print("=" * 70)

def get_bert_embeddings(texts, model, tokenizer, device, max_length, batch_size=16):
    model.eval()
    embeddings = []
    with torch.no_grad():
        for i in tqdm(range(0, len(texts), batch_size), desc="Extracting embeddings"):
            batch = texts[i:i+batch_size]
            encoded = tokenizer(
                list(batch),
                add_special_tokens=True,
                max_length=max_length,
                padding='max_length',
                truncation=True,
                return_tensors='pt'
            )
            outputs = model.bert(
                input_ids=encoded['input_ids'].to(device),
                attention_mask=encoded['attention_mask'].to(device)
            )
            cls_embeddings = outputs.last_hidden_state[:, 0, :].cpu().numpy()
            embeddings.append(cls_embeddings)
    return np.vstack(embeddings)
```

```
# Train Random Forest su hybrid features
print("Training Random Forest...")
rf_hybrid = RandomForestClassifier(
    n_estimators=200,
    max_depth=20,
    min_samples_split=5,
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)
rf_hybrid.fit(X_hybrid_train, y_train)
```

Hybrid Test Performance:

Accuracy: 0.9783
Precision: 0.9783
Recall: 0.9783
F1-Score: 0.9783
ROC-AUC: 0.9986

BASELINE: STYLOMETRIC ONLY

- Questo modello di baseline utilizza esclusivamente feature stilometriche, senza informazioni semantiche profonde come quelle fornite da BERT, per valutare quanto lo stile di scrittura sia discriminante nella distinzione tra testi umani e generati da AI.
- Le feature catturano aspetti statistici e strutturali del testo, tra cui variabilità delle frasi, uniformità stilistica, ricchezza lessicale, complessità sintattica e presenza di pattern ripetitivi.
- Il classificatore scelto è un Random Forest, che garantisce buone prestazioni e soprattutto interpretabilità, grazie all'analisi delle feature importance.
- Questo approccio funge da termine di confronto con modelli più complessi e permette di individuare i principali segnali stilistici associati alla generazione artificiale del testo.

===== MODEL 3: BASELINE (Stylometric Only) =====

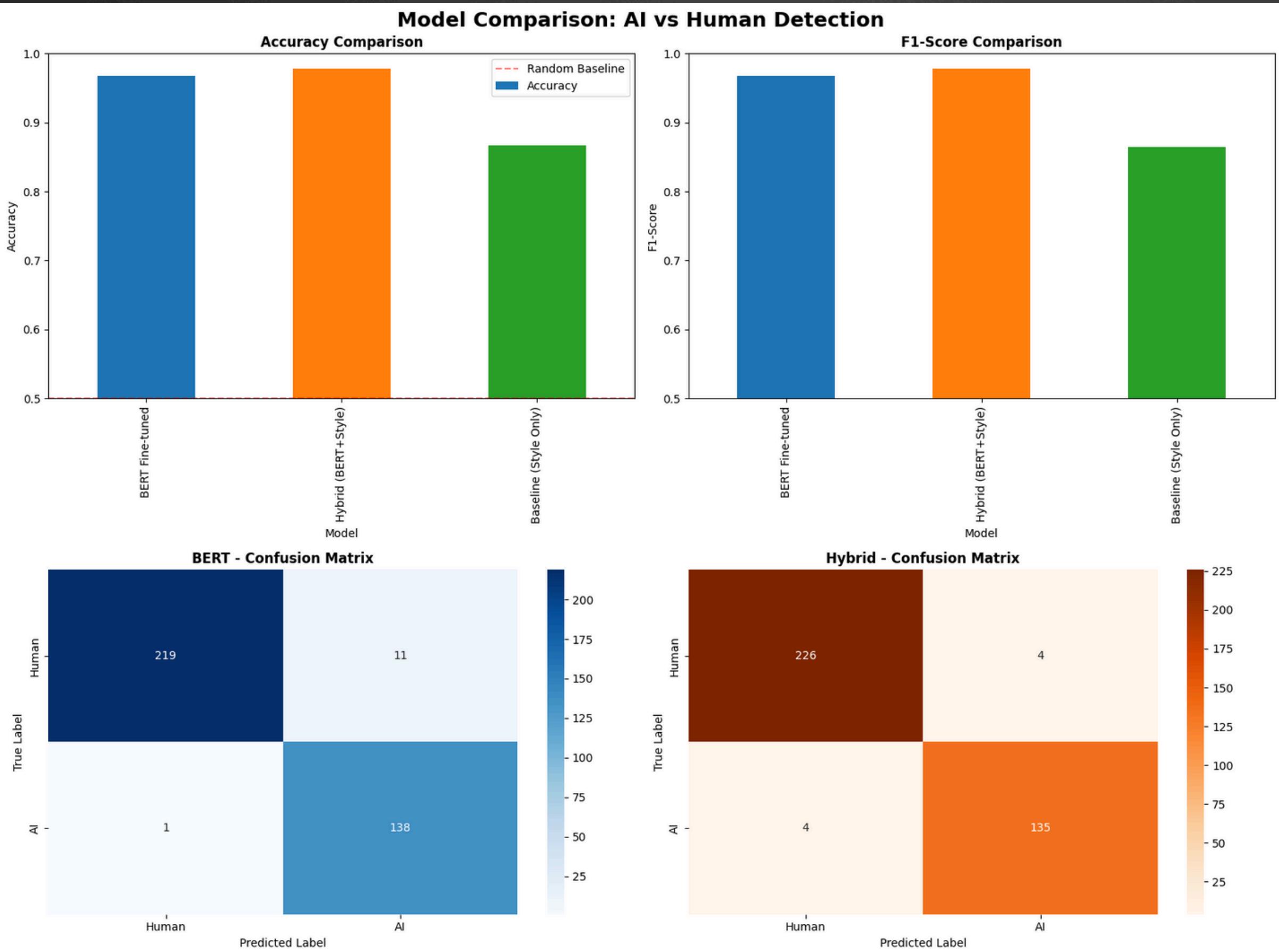
Baseline Test Performance:

Accuracy: 0.8672
Precision: 0.8678
Recall: 0.8672
F1-Score: 0.8650
ROC-AUC: 0.9364

Top 5 Most Important Stylometric Features:

	feature	importance
1	pos_bigram_entropy	0.235092
2	sentence_length_cv	0.113427
3	burstiness_index	0.110065
4	clause_density	0.093765
5	function_word_ratio	0.084303

MODEL COMPARISON



MODEL COMPARISON

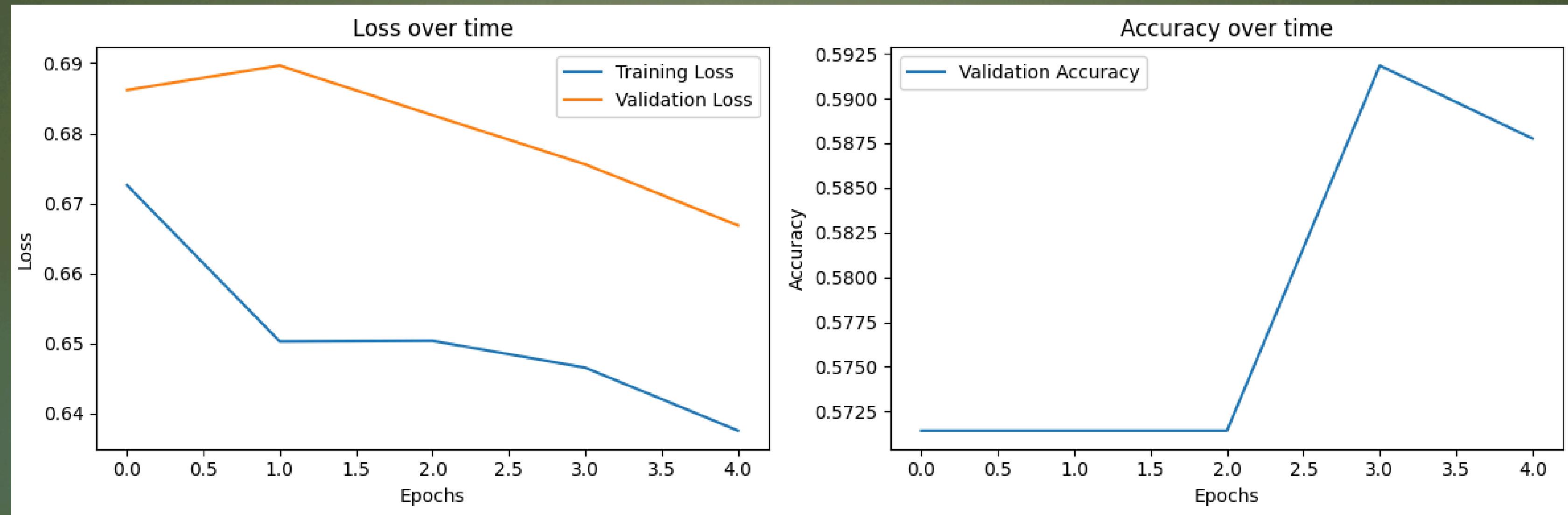
Model	Accuracy	F1-Score
BERT Fine-tuned	0.967480	0.967683
Hybrid (BERT+Style)	0.978320	0.978320
Baseline (Style Only)	0.867209	0.864972

Reti Neurali

RETI NEURALI PER IL MODELING SEQUENZIALE E LOCALE DEL LINGUAGGIO

LSTM

Analizza la sequenza del testo. Legge le parole una dopo l'altra e mantiene una "memoria" di ciò che ha letto prima.



'Loss' diminuisce drasticamente già dopo la prima epoca, il che significa che il modello capisce molto velocemente le differenze strutturali tra i testi. L'accuratezza di validazione sale stabilmente, confermando che il sistema non sta solo ripetendo ciò che ha visto, ma ha davvero imparato a riconoscere lo stile dell'intelligenza artificiale.

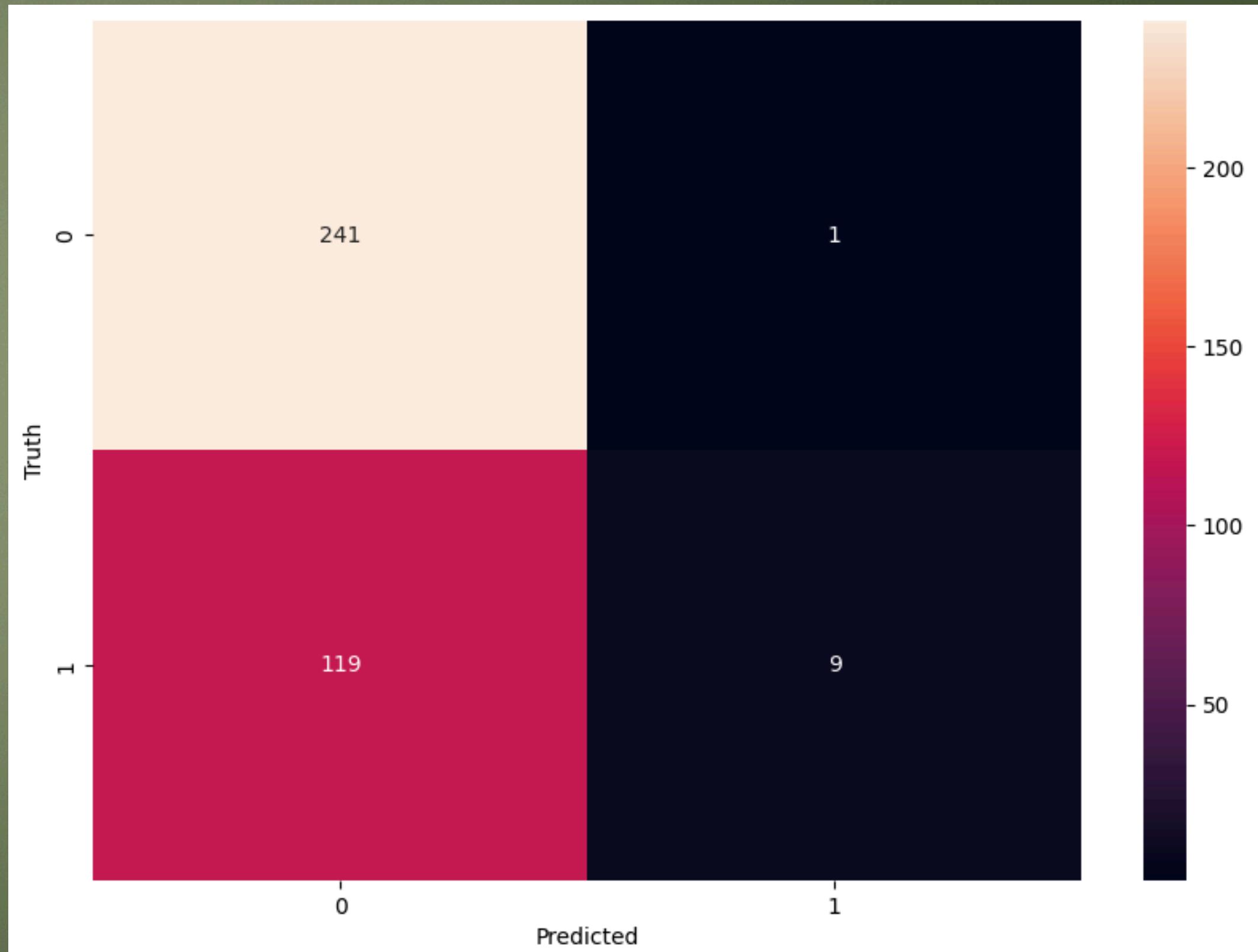
LSTM

Il modello viene messo in modalità "lettura". Si disattivano funzioni come il Dropout perché ora non dobbiamo più addestrare, ma solo ottenere risposte. Senza Gradienti viene indicato a PyTorch di non calcolare i gradienti (il "motore" dell'apprendimento). Le previsioni vengono arrotondate: se il valore è sopra 0.5 è considerato AI (1), altrimenti Umano (0). Oltre all'accuracy, vengono generati il Classification Report (per precisione e richiamo) e la Matrice di Confusione.

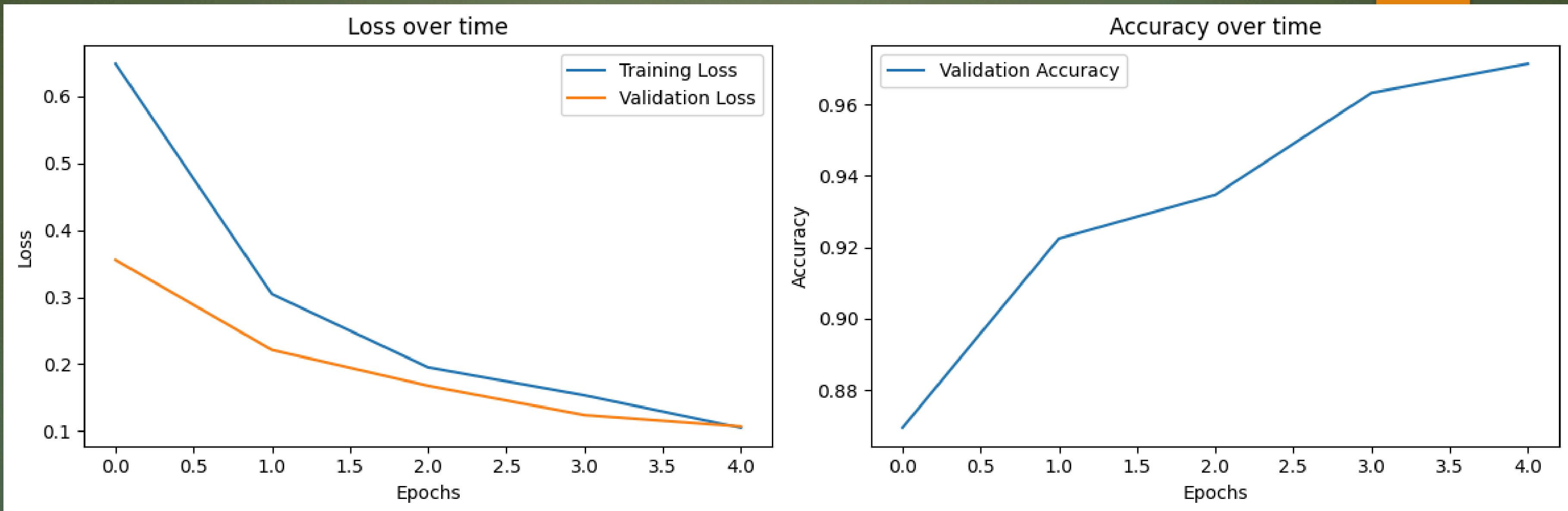
Test Loss: 0.627, Test Accuracy: 0.676

	precision	recall	f1-score	support
0.0	0.67	1.00	0.80	242
1.0	0.90	0.07	0.13	128
accuracy			0.68	370
macro avg	0.78	0.53	0.47	370
weighted avg	0.75	0.68	0.57	370

Questa è la fase più importante: si osserva se il nostro modello LSTM ha davvero imparato a riconoscere lo stile dell'AI o se ha solo memorizzato degli schemi. La Matrice di Confusione dice esattamente dove il sistema è più forte e dove invece fatica, permettendo di capire quanto sia possibile fare affidamento delle sue decisioni.



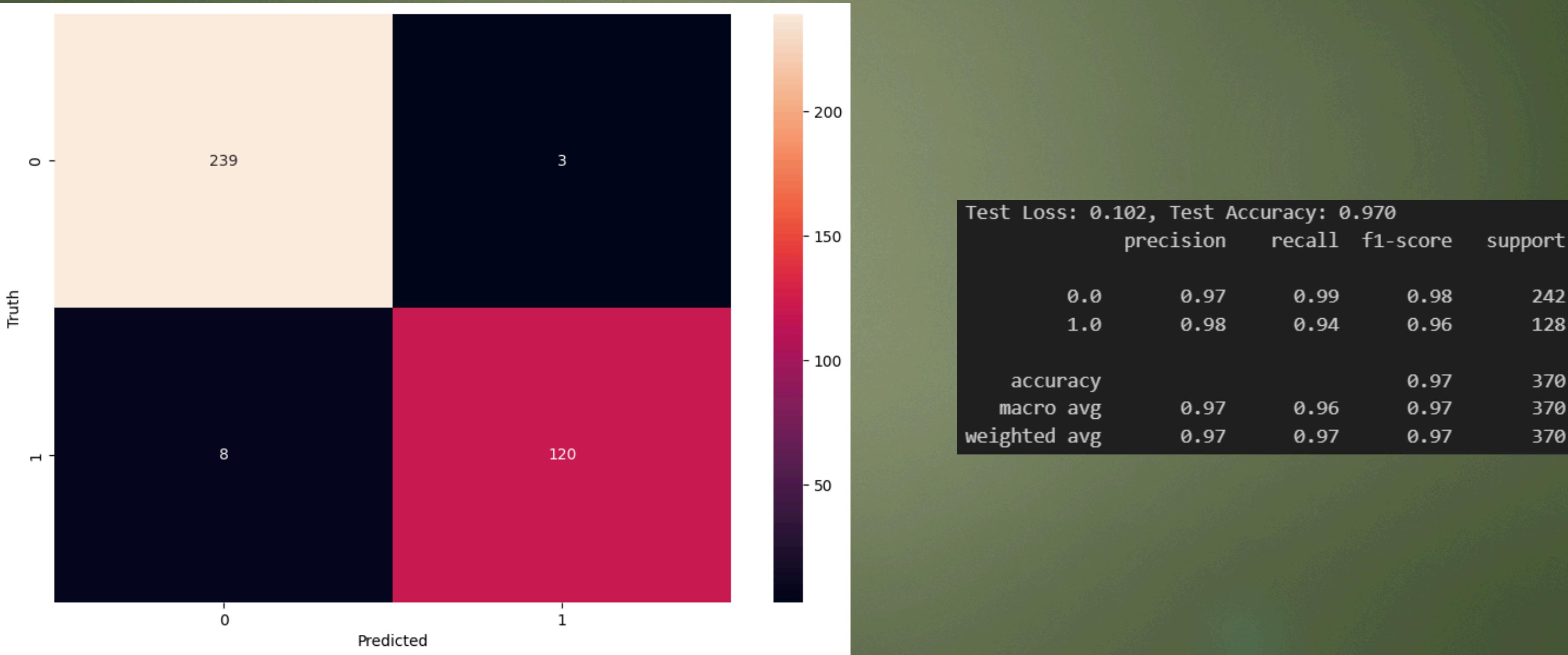
CNN



Mentre la LSTM legge il testo come un racconto, questa CNN lavora più come un correttore di bozze esperto: scansiona il testo cercando combinazioni di parole che l'AI usa tipicamente. Come si nota dai grafici, questo metodo è molto diretto ed efficace, raggiungendo un'ottima precisione in pochissimo tempo.

CNN

Viene eseguito dopo che l'allenamento è terminato, utilizzando il `test_loader`, ovvero una porzione di dati che il modello non ha mai incontrato durante le epoche precedenti. L'`evaluate_model(...)` è la funzione che interroga il modello su centinaia di nuovi esempi. Per ogni testo, il modello fornisce una probabilità; se è maggiore di 0.5, lo classifica come AI, altrimenti come Umano. Il `test_loss` e `test_acc` sono i due "voti" finali. La `Loss` indica quanto il modello era incerto/distante dalla risposta corretta, l' `Accuracy` indica la percentuale netta di testi classificati correttamente. Il `sns.heatmap(conf_matrix...)` è la parte visuale. Crea una mappa di calore della Matrice di Confusione, che è lo strumento più potente per capire dove il modello si confonde (ad esempio, se scambia spesso gli umani per AI o viceversa).

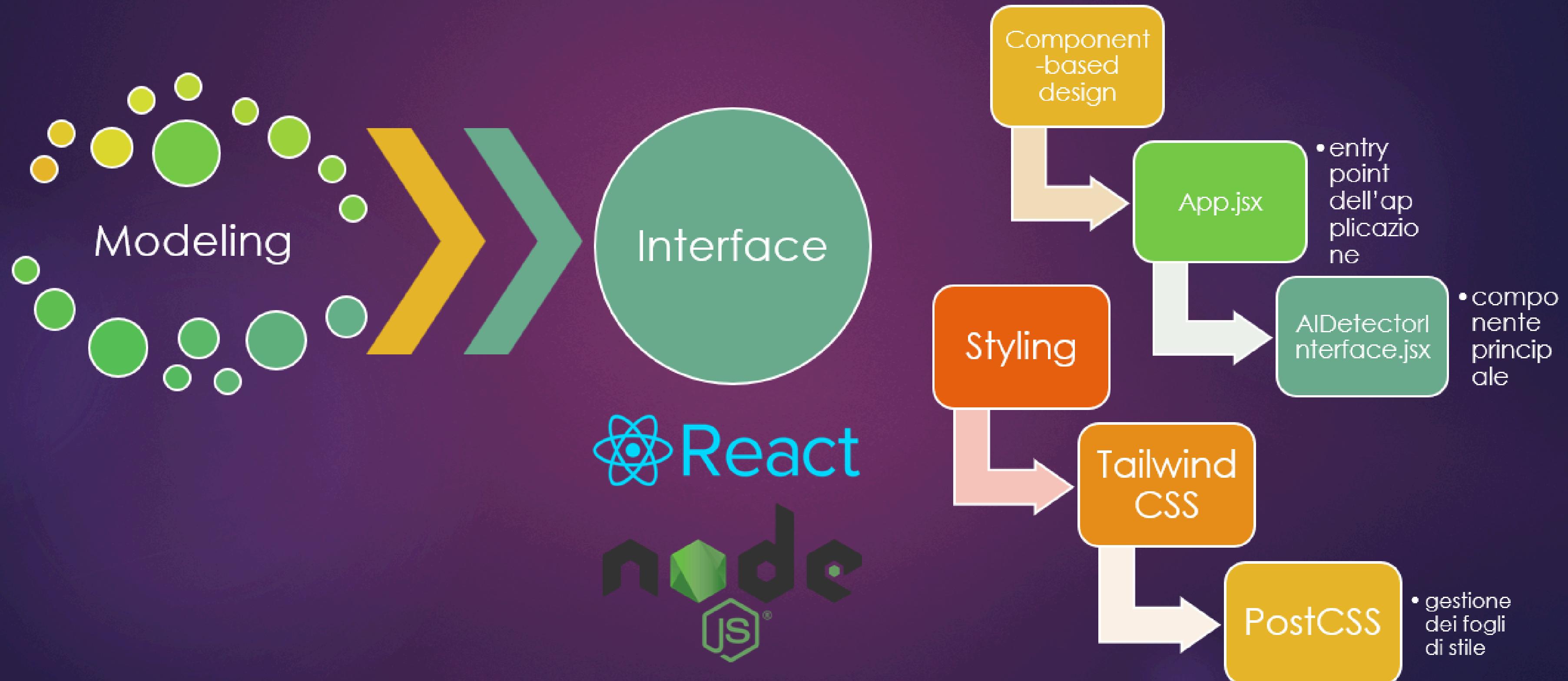


User Interface del sistema di AI Detection

DAL MODELLO ALL'ESPERIENZA UTENTE

Architettura dell'interfaccia

Dopo aver costruito il modello, è stata progettata un'interfaccia semplice che permette all'utente di interagire direttamente con il sistema di detection.



User Interaction Flow



- L'interfaccia è un ponte tra modello e utente
- Il modello lavora su feature stilometriche
- L'utente riceve una decisione chiara e immediata

Presentazione interfaccia

Input Analysis

Initiate Neural Scan

Neural Report

Download Report

NEURAL PROBE

Advanced AI Detection System

Input Analysis

Enter Text for Analysis
Paste your text here... (minimum 50 characters)

Characters: 0 / 50 minimum

Load AI Example Load Human Example

Initiate Neural Scan

Powered by Advanced Neural Analysis • Hybrid BERT + Stylometric Detection

NEURAL PROBE

Advanced AI Detection System

Input Analysis Neural Report

AI Generated Neural Analysis Complete

97.7% Confidence

Stylometric Signature

Structural Repetition	97.7%
Lexical Poverty	97.7%
Buzziness Index	56.8%

Linguistic Metrics

Lexical Diversity	2.3%
Avg Sentence Length	8 words
Buzziness	0.50
Similarity	97.7%

Neural Interpretation

The text exhibits high structural uniformity (97.7% similarity) and limited lexical variation (2.3% diversity), typical patterns of AI-generated content. The sentence construction follows predictable templates with low buzziness (0.50), indicating algorithmic origin.

Download Neural Report

Powered by Advanced Neural Analysis • Hybrid BERT + Stylometric Detection

NEURAL PROBE

CERTIFIED AI CONTENT ANALYSIS

AI GENERATED

Executive Forensic Summary

Analysis identified high structural uniformity and low lexical variation. The linguistic DNA matches patterns typically found in generative pre-trained transformers (GPT).

98% Confidence

Linguistic Metrics

Lexical Diversity	2.7%
Syntactic Complexity	68.6%
Buzziness (Entropy)	0.6%
Probability Score	97.7%

Stylometric Distribution

Tokens Analyzed: 80
Sentence Mean: 8.8
Unique Lemmas: 37

REPORT ID: NP-8987654321
Generated by Neural Probe AI Forensic Engine v2.0

Page 1 of 1

CASI DI TEST

Categoria di test	Origine testo	Tipologia	Output Interfaccia	Interpretazione
AI standard	Gemini	50 parole	AI – 92%	AI chiaramente identificata anche su testi brevi
AI standard	Gemini	200 parole	AI – 92%	Predizione stabile con più contesto
AI standard	Gemini	500 parole	AI – 92%	Firma stilometrica consistente
AI camuffata	Gemini	50 parole	AI – 89.3%	Camuffamento parziale, firma ancora visibile
AI camuffata	Gemini	200 parole	AI – 95%	Separazione migliorata con struttura testuale
AI camuffata	Gemini	500 parole	AI – 92%	Robustezza su testi lunghi
AI ultra-camuffata	ChatGPT	~300 parole	AI – 85%	Camuffamento efficace ma non sufficiente
AI ultra-camuffata	Gemini	~200 parole	AI – 86%	Traccia generativa persistente
AI narrativa	ChatGPT	50 parole	AI – 82%	Ambiguità maggiore su testi brevi
AI narrativa	ChatGPT	200 parole	AI – 78.1%	Confidenza ridotta ma classificazione corretta
AI narrativa	ChatGPT	500 parole	AI – 83.4%	Migliore separazione con più contenuto

CASI DI TEST

Categoria di test	Origine testo	Tipologia	Output Interfaccia	Interpretazione
Camuffato da umano	ChatGPT	50 parole	AI – 73.4%	Zona grigia stilometrica
Camuffato da umano	ChatGPT	200 parole	AI – 81%	Firma AI emergente
Camuffato da umano	ChatGPT	500 parole	AI – 82%	Coerenza stilometrica rilevata
Testo storico	Costituzione USA	Articolo breve	Human – 52%	Stile formale, borderline
Testo storico	Costituzione USA	Articolo medio	Human – 72%	Firma umana correttamente riconosciuta
Testo storico	Costituzione USA	Articolo lungo	Human – 79%	Maggiore confidenza su testi strutturati
Testo storico	Costituzione USA	Estratto elezioni	Human – 69%	Linguaggio normativo stabile
Testo storico	Costituzione USA	Sistema giudiziario	Human – 56.1%	Ambiguità dovuta a stile rigido

Confronto diretto: Gemini vs ChatGPT

Aspetto	Gemini	ChatGPT
Regolarità sintattica	Molto alta	Media
Variabilità lessicale	Più bassa	Più alta
Resistenza al camuffamento	Bassa	Media
Stabilità della firma	Elevata	Moderata
Ambiguità stilometrica	Minima	Maggior

- **Gemini** risulta più facilmente riconoscibile perché più “ordinato”.
- **ChatGPT** è più difficile da classificare nei testi brevi e narrativi, ma non riesce a eliminare completamente la firma generativa.

Nel complesso, i risultati mostrano che:

- la lunghezza del testo è un fattore chiave per la robustezza
 - il camuffamento riduce la confidenza ma non inganna il modello
 - testi umani storici possono apparire artificiali a livello stilometrico
 - ogni generatore AI lascia una firma riconoscibile nel tempo
- Il detector non riconosce “cosa” viene scritto, ma “come” il testo prende forma



AI_DETECTOR DATABASE

PERSISTENZA E TRACCIABILITÀ DELLE PREDIZIONI TRAMITE DATABASE
EMBEDDED

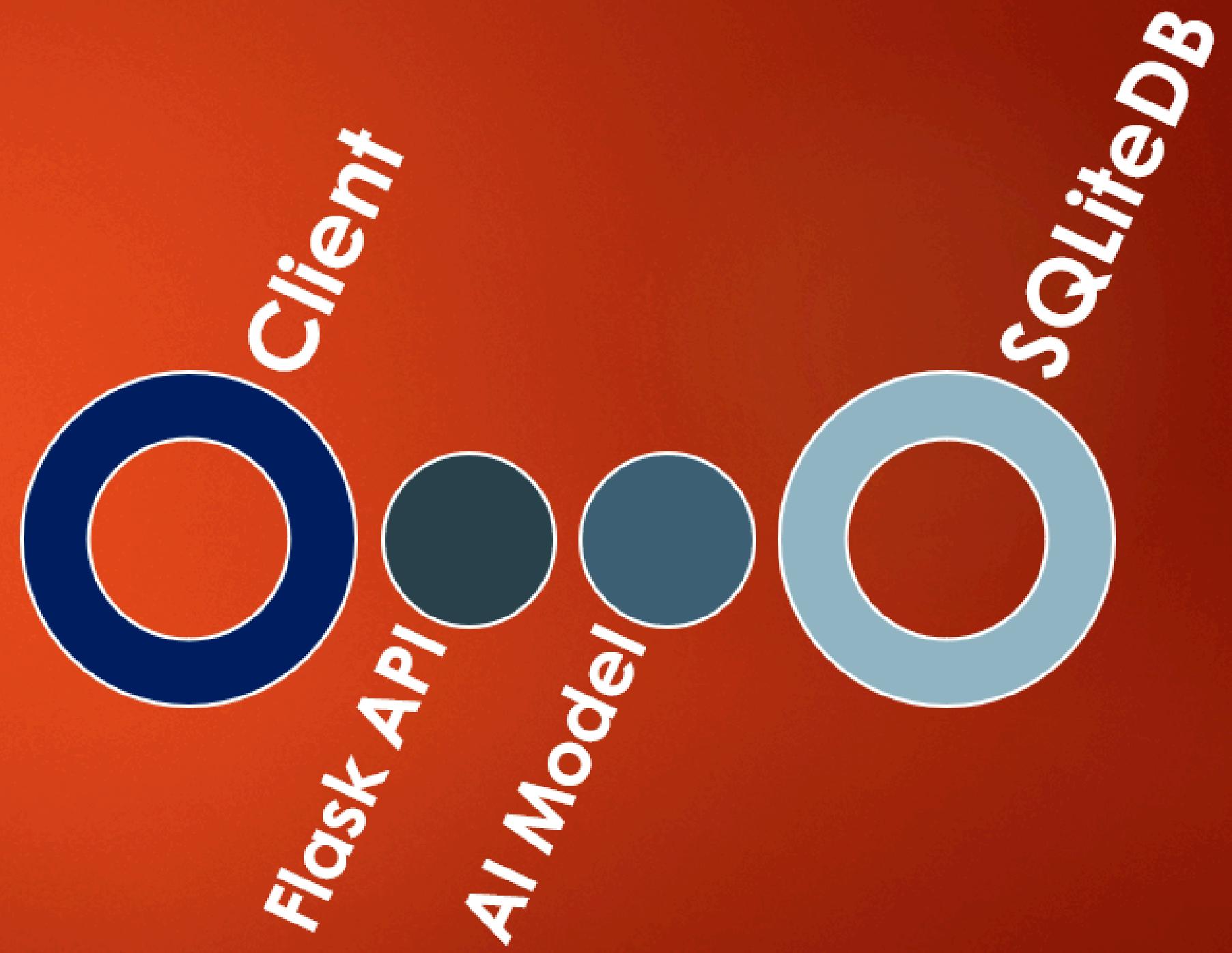
Ruolo di SQLite nel progetto

Memorizza tutte le predizioni del modello

Permette:

- analisi storica
- audit dei risultati
- debug e validazione

Separa logica di AI e persistenza dei dati



Struttura del Database

Schema della tabella sequence

Tabella: sqlite_sequence

	name	seq
	Filtro	Fil...
1	predictions	10

Schema della tabella predictions

Campo	Tipo	Descrizione
id	INTEGER	Chiave primaria
timestamp	TEXT	Data e ora (UTC)
prediction	TEXT	AI / Human
confidence	REAL	Confidenza (%)
lexical_diversity	REAL	Feature stilometrica
burstiness	REAL	Feature stilometrica
avg_sentence_length	REAL	Feature stilometrica
model_version	TEXT	Versione modello

Tabella: predictions

Filtrare in ogni colonna

	id	timestamp	prediction	confidence	lexical_diversity	burstiness	avg_sentence_length	model_version
	Fil...	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	1	2025-12-31T22:51:53.332282	AI	97.7151210055575	0.9	-0.8333333333333333	0.0909090909090909	Hybrid
2	2	2025-12-31T22:52:01.440348	Human	60.9518286550195	0.842105263157895	-1.0	0.0	Hybrid
3	3	2025-12-31T22:55:16.788167	AI	99.5	0.862745098039216	-0.733683726352458	0.15361295119719	Hybrid
4	4	2025-12-31T22:55:28.768593	AI	100.0	0.738916256157635	-0.60364367336633	0.247159848048818	Hybrid
5	5	2025-12-31T22:55:51.816546	AI	100.0	0.549800796812749	-0.48611124215314	0.345794273854168	Hybrid
6	6	2025-12-31T22:56:09.025016	Human	56.6411831043626	0.686567164179104	-0.7333333333333333	0.153846153846154	Hybrid
7	7	2025-12-31T22:56:25.095683	AI	84.1100091513499	0.678048780487805	-0.537223584768038	0.301046913290621	Hybrid
8	8	2025-12-31T22:56:37.173314	AI	85.9462803055463	0.638190954773869	-0.575990886722654	0.269042871281504	Hybrid
9	9	2025-12-31T22:56:57.168753	Human	52.312101277205	0.84	-1.0	0.0	Hybrid
10	10	2025-12-31T22:57:09.065330	Human	71.727129775735	0.498293515358362	-0.365889252277027	0.464247556429534	Hybrid



Docker Container

AMBIENTE RIPRODUCIBILE E DEPLOYMENT DEL SISTEMA

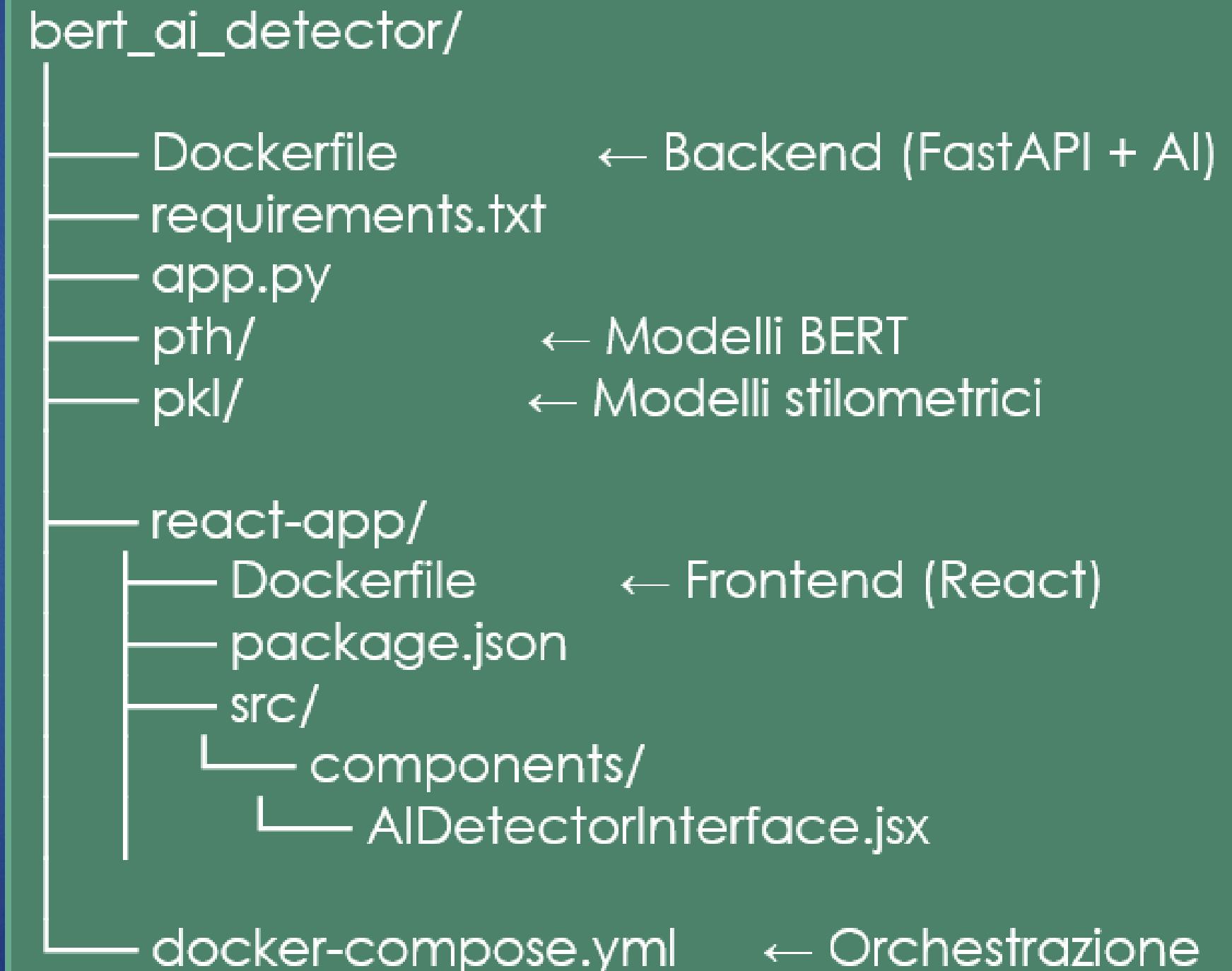
Containerizzazione del sistema AI Detector

Docker ci permette di eseguire lo stesso sistema su qualsiasi macchina senza problemi di configurazione

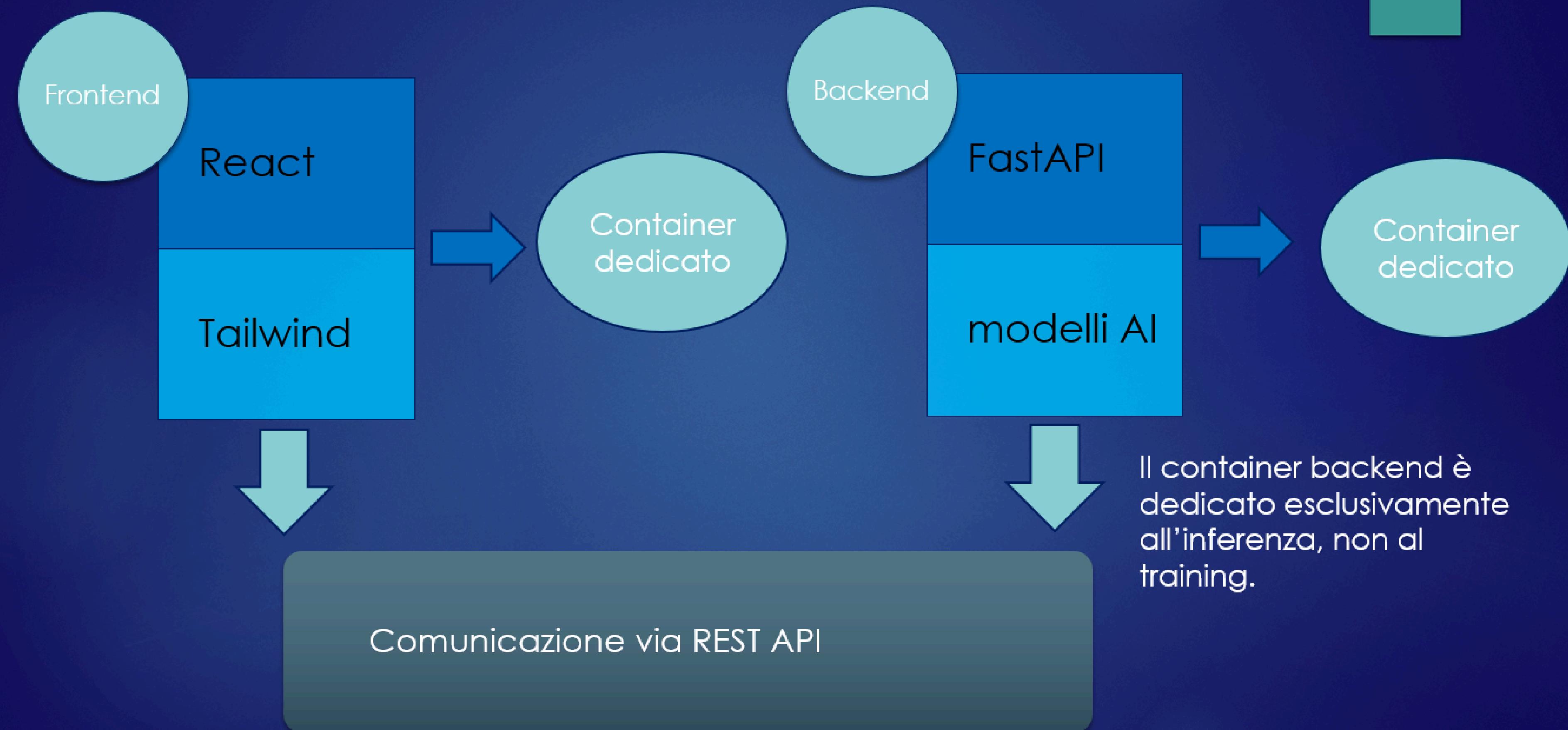
Necessità di ambienti riproducibili per modelli AI

Complessità di dipendenze (Python, PyTorch, React, Node)

- isolamento dell'ambiente
- portabilità
- riproducibilità degli esperimenti



Architettura Full-Stack



Docker Compose

Orchestrazione con Docker Compose

Docker Compose consente di orchestrare applicazioni composte da più container, permettendo di definire in modo centralizzato tutti i servizi del sistema.

Nel nostro progetto viene utilizzato per gestire backend AI e frontend React come servizi separati ma interconnessi.

Compose

avvio dei container nell'ordine corretto

espone le porte necessarie

L'intero sistema può essere avviato con un solo comando, rendendo il deploy semplice e riproducibile

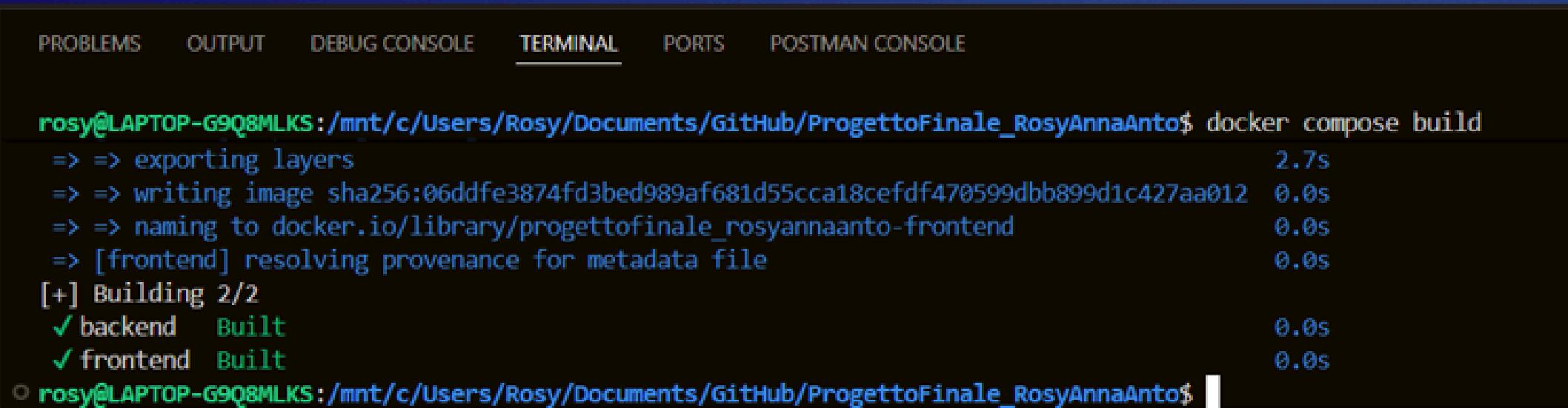
`docker-compose build`

`docker-compose up`

Docker

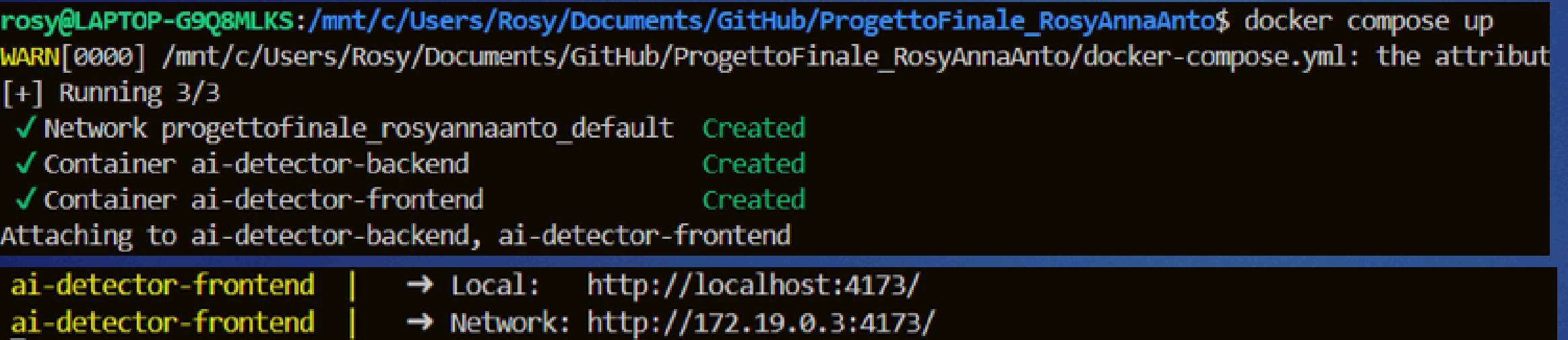
1. docker compose build → conferma la build delle immagini per backend e frontend.

Docker Compose ha completato la build delle immagini senza errori, con il backend e il frontend correttamente configurati.



```
rosy@LAPTOP-G9Q8MLKS:/mnt/c/Users/Rosy/Documents/GitHub/ProgettoFinale_RosyAnnaAnto$ docker compose build
=> => exporting layers                                         2.7s
=> => writing image sha256:06ddfe3874fd3bed989af681d55cca18cefdf470599dbb899d1c427aa012 0.0s
=> => naming to docker.io/library/progettofinale_rosyannaanto-frontend                0.0s
=> [frontend] resolving provenance for metadata file                           0.0s
[+] Building 2/2
✓ backend   Built                                                 0.0s
✓ frontend  Built                                                 0.0s
rosy@LAPTOP-G9Q8MLKS:/mnt/c/Users/Rosy/Documents/GitHub/ProgettoFinale_RosyAnnaAnto$
```

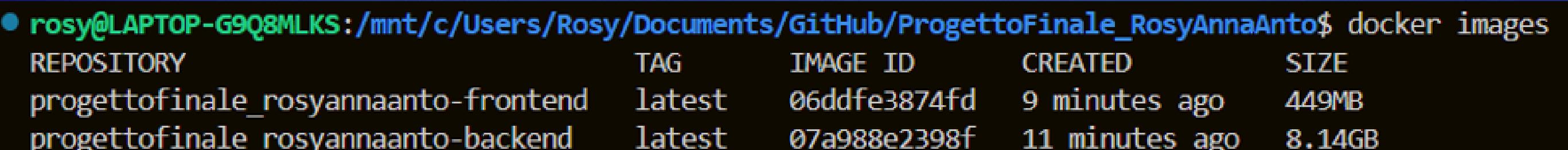
2. docker-compose up → avvia i container per il frontend e il backend



```
rosy@LAPTOP-G9Q8MLKS:/mnt/c/Users/Rosy/Documents/GitHub/ProgettoFinale_RosyAnnaAnto$ docker compose up
WARN[0000] /mnt/c/Users/Rosy/Documents/GitHub/ProgettoFinale_RosyAnnaAnto/docker-compose.yml: the attribut
[+] Running 3/3
✓ Network progettofinale_rosyannaanto_default  Created
✓ Container ai-detector-backend               Created
✓ Container ai-detector-frontend              Created
Attaching to ai-detector-backend, ai-detector-frontend
ai-detector-frontend |    → Local:  http://localhost:4173/
ai-detector-frontend |    → Network: http://172.19.0.3:4173/
```

3. docker images → le immagini Docker per il backend e frontend sono state correttamente buildate.

La build è completata senza errori, e le immagini sono pronte per l'uso.



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
progettofinale_rosyannaanto-frontend	latest	06ddfe3874fd	9 minutes ago	449MB
progettofinale_rosyannaanto-backend	latest	07a988e2398f	11 minutes ago	8.14GB

Conclusioni

Abbiamo dimostrato che distinguere testi umani e generati da AI è possibile senza affidarsi a parole chiave, scorciatoie o black box.

Il nostro approccio si basa su un'idea chiave:
non riconoscere cosa viene scritto, ma come nasce la scrittura. Attraverso:

- una firma stilometrica interpretabile
- modelli ibridi (BERT + feature linguistiche)
- una pipeline robusta e riproducibile

il sistema riesce a separare umano e AI anche quando il contenuto è neutro o camuffato.

Il risultato non è solo un modello accurato, ma un sistema completo:

- spiegabile
- tracciabile
- deployabile

pronto per un utilizzo reale.

La scrittura artificiale evolve, ma lascia sempre una traccia. Il nostro lavoro mostra come renderla misurabile.