

GenAI Code Detection & Attribution

✓ Full Participation: Subtasks 1, 2 & 3

Speech & Language Processing Exam

Giovanni Giuseppe Iacuzzo

Università degli Studi di Enna “Kore”

对学生涯的贡献

 [giovannilacuzzo](#)



Roadmap



1 | Introduzione e Obiettivi

4 | Risultati del Task

2 | Analisi dei Dati

5 | Conclusioni

3 | Architettura e Metodologia

Subtask B: Multi-Class Detection

Subtask C: Hybrid & Adversarial

Il Contesto: Perché SemEval Task 13?

La generazione di codice tramite LLM (GPT-4, StarCoder) è pervasiva. Tuttavia, i detector attuali sono fragili: funzionano bene su ciò che conoscono (Python, Java), ma crollano appena cambia il contesto.

Le sfide aperte:

Il problema O.O.D. (Out-Of-Distribution):

La maggior parte dei dataset contiene codice "Algoritmico" (LeetCode). Nel mondo reale, il codice è vario (Web, Embedded) e scritto in linguaggi rari (Go, PHP, C#).

Attribuzione e Copyright:

Non basta sapere se è stato generato da un'AI. Per questioni legali, dobbiamo sapere *quale* modello lo ha creato (OpenAI vs Open Source).

Scenari Ibridi e Avversari:

Il codice non è più binario (0 o 1). Esiste il codice misto (Human + Co-pilot) e il codice generato appositamente per ingannare i detector (Adversarial).

Subtask A: Binary Detection & OOD

Dataset: 500k Train / 100k Val

Metric: Macro F1

II Core Task

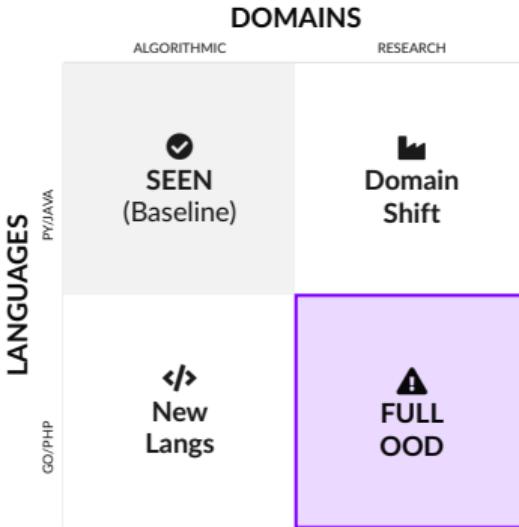
Classificazione binaria: il codice è stato scritto da un umano o generato da un LLM? La difficoltà non sta nella classificazione in sé, ma nella **Robustezza**.

Le Dimensioni della Generalizzazione:

Language Shift Il modello si allena su linguaggi "standard" (*Python, Java*) ma deve funzionare su sintassi completamente diverse (*Go, PHP*). *Deve imparare la semantica, non la sintassi.*

Domain Shift **Train:** Codice algoritmico "pulito" (LeetCode). **Test:** Codice di *Produzione* (sporco, dipendenze, commenti reali).

*Obiettivo: Evitare il "memorization overfitting".



Subtask B: Fine-Grained Attribution

Authorship Forensics Non basta distinguere Umano vs Macchina. Il sistema deve riconoscere l'**impronta stilistica unica** di ogni famiglia di modelli (es. come *Llama* gestisce i cicli vs come li fa *DeepSeek*).

Extreme Class Imbalance

Il dataset è fortemente sbilanciato verso il codice umano, rendendo difficile per il modello apprendere le sfumature delle classi minoritarie (AI).

HUMAN (442k)

SINGLE AI FAMILY (4k)



The "Unseen" Trap

Test su **nuove versioni** di famiglie note.

Esempio: Train su Llama-2 → Test su Llama-3. Il modello deve generalizzare lo "stile Llama" oltre la specifica versione.

Target Families (10+1)

DeepSeek

Qwen

01-ai

BigCode

Gemma

Phi

LLaMA

Granite

Mistral

OpenAI

HUMAN CODE

Subtask C: The Real-World Scenario

Max Dataset: 900k Samples · 4 Classes

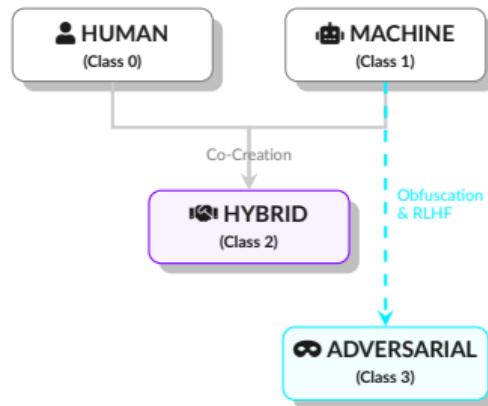
Oltre alla distinzione binaria, introduciamo le **Zone Grigie** dello sviluppo software moderno.

Class 2: HYBRID (Mixed)

"Augmented Intelligence". Codice nato dalla collaborazione: l'umano scrive la logica core, l'AI (Copilot) completa il boilerplate (o viceversa).
→ Sfida: Rilevare i punti di transizione nello stesso file.

Class 3: ADVERSARIAL

"Evasion Attacks". Codice interamente AI, ma generato tramite tecniche avanzate (RLHF, Prompt Engineering) per imitare lo stile e gli errori umani.
→ Sfida: Distinguere la "naturalezza" umana dalla "simulazione".



⚠ **Difficulty:** Adversarial code is mathematically closer to Human distribution than standard Machine code.

Data Distribution: The Training Bias

Analizzando il Training Set (500k samples), emerge un chiaro **sbilanciamento** verso i linguaggi mainstream.

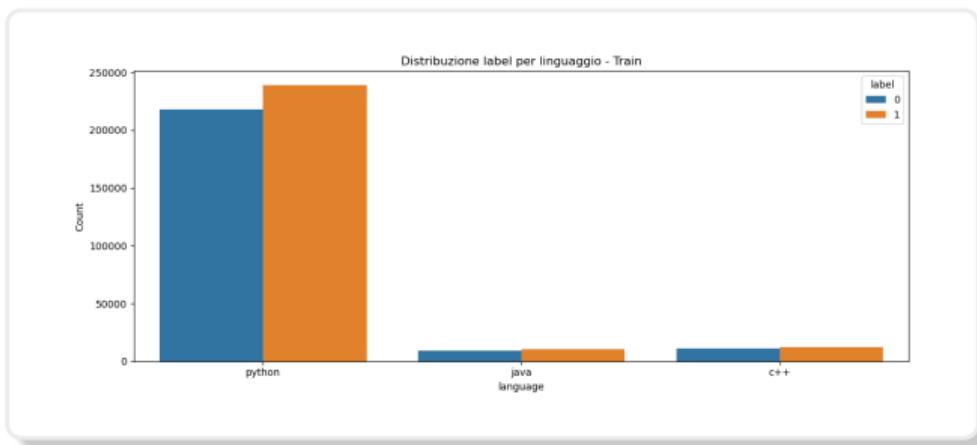


Fig 1. Distribuzione dei linguaggi nel training set.

Composizione



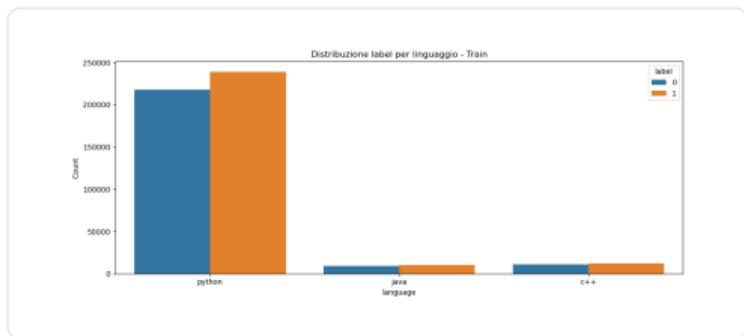
The Risk

Il modello rischia il "**Syntactic Overfitting**": impara a riconoscere le keyword di Python ('def', 'import') invece di capire se il codice è generato da un'AI.

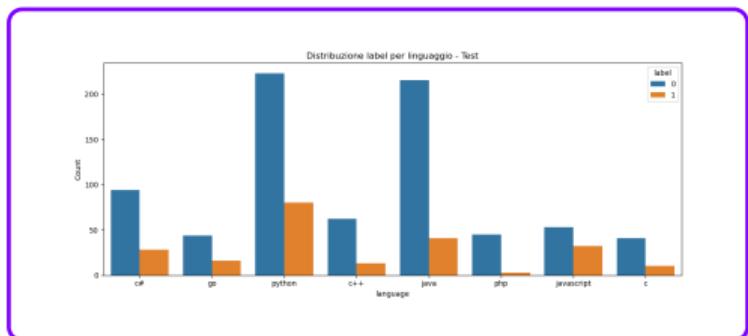
The Shift: Confronto Train vs Test

Il Test Set non è solo "più dati". È un cambio di paradigma.

TRAIN (Seen)



TEST (Unseen)



⚠️ Osservazione Critica: Nel Test set compaiono **Go, PHP e C#**. Se il nostro modello si è basato su keyword specifiche di Python (es. 'def', 'import'), fallirà miseramente qui. Serve astrazione.

Code Properties: Human vs AI Verbosity

Esiste una differenza strutturale tra codice Umano e AI? Sì: la Lunghezza.

Analisi Stilistica



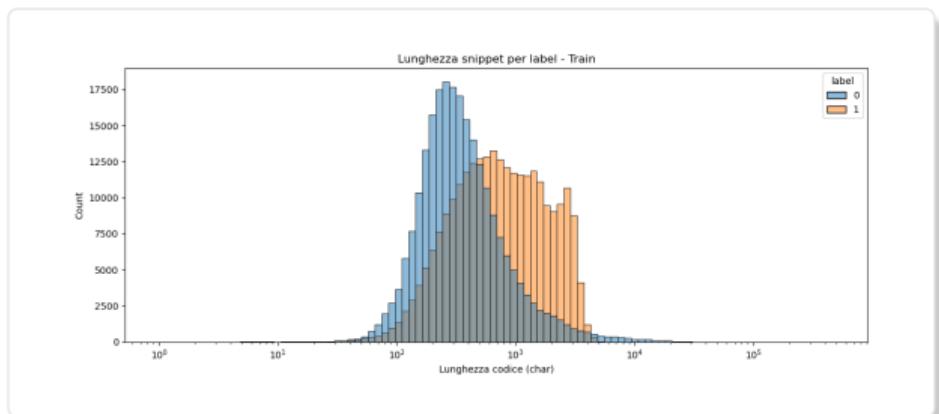
AI Code (Verboso):

I modelli tendono alla sovrastruttura: includono spesso commenti esplicativi, gestione errori standard e codice "boilerplate" non richiesto.



Human Code (Conciso):

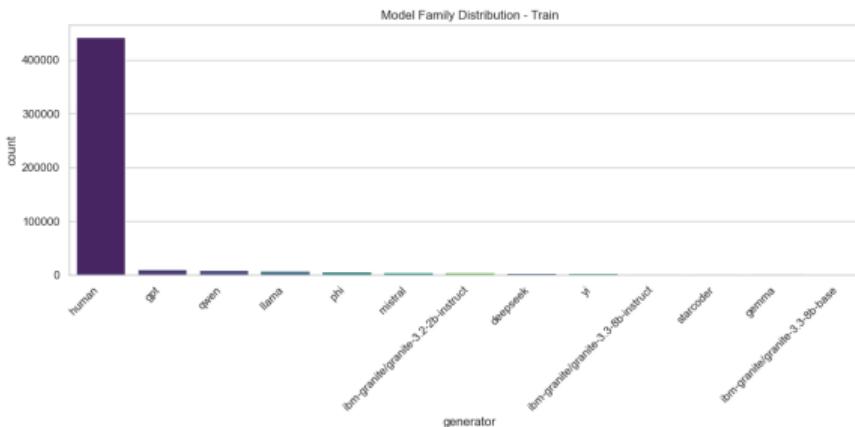
In contesti algoritmici (es. LeetCode), l'umano punta all'essenziale, omettendo commenti e ottimizzando la brevità.



💡 **Insight:** La lunghezza (token count) è un forte segnale predittivo.

Task B: The Imbalance Challenge

Il dataset grezzo contiene 31 varianti di modelli. Abbiamo normalizzato le label in **11 Famiglie**.



Preprocessing

Mapping Strategy:

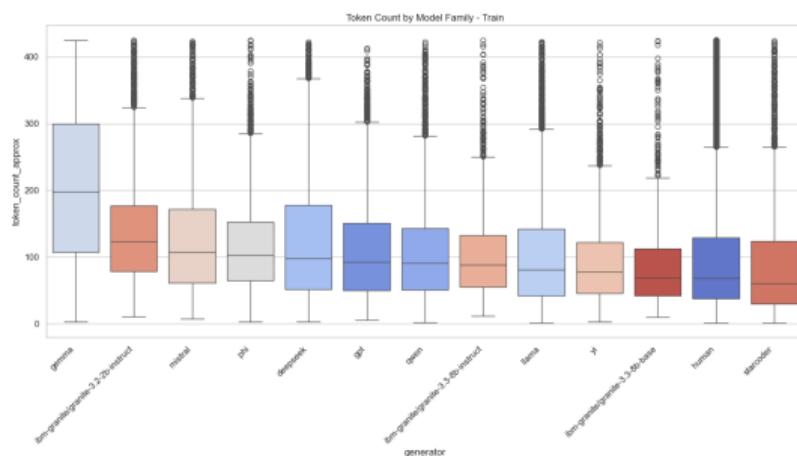
- ▶ `llama-2-7b` → LLaMA
- ▶ `gpt-3.5-turbo` → GPT
- ▶ `deepseek-coder` → DeepSeek

The Issue

La classe **Human** domina (90%). Le classi AI sono rare (<1%).

Task B: Stylistic Fingerprints

Ogni famiglia di modelli ha una "firma" unica? Sì, la lunghezza dei token lo rivela.



Analisi Il grafico mostra la distribuzione dei token per famiglia.

+ Verbose Models

GPT & Claude. Tendono a spiegare molto, producendo outlier lunghi.

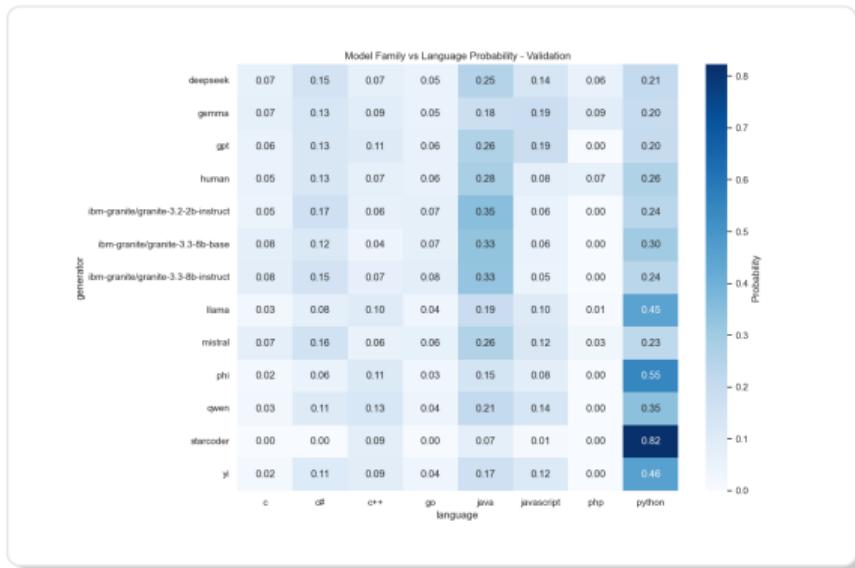
- Concise Models

LLaMA & Mistral. Spesso addestrati per essere diretti o usati per code-completion pura.

Questo pattern aiuta a distinguere GPT da LLaMA anche a parità di correttezza del codice.

Task B: Model Specialization

Esiste una correlazione tra Generatore e Linguaggio? La Heatmap rivela bias specifici.



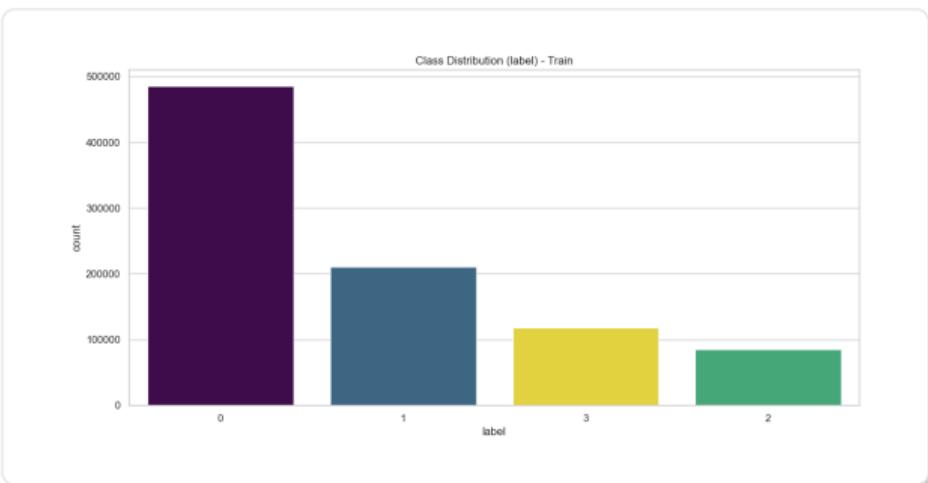
⋮⋮ Probability Signature

Analisi di $P(\text{Language}|\text{Generator})$:

- ▶ **Mono-Linguaggio (StarCoder):** Un caso estremo. Il 82% del suo codice è Python. Se il sample è C# o PHP, è quasi certamente *non* StarCoder.
- ▶ **Enterprise Bias (IBM Granite):** Mostra una preferenza distintiva per Java (0.35), differenziandosi da Llama che preferisce Python (0.45).
- ▶ **Generalisti (GPT):** Bilanciato tra Java (0.26), Python (0.20) e JS (0.19). Più difficile da attribuire basandosi solo sul linguaggio.

Task C: The Class Imbalance Problem

Il Subtask C introduce le "Minority Classes" (Hybrid, Adversarial). L'analisi della distribuzione rivela una sfida critica per il training.



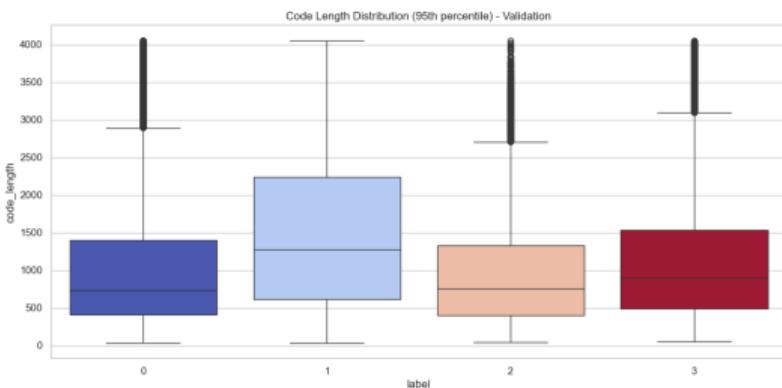
Class Breakdown

- ▶ **Dominanti (Classes 0-1):** Human e Machine pure coprono la maggioranza del dataset.
- ▶ **Rare (Classes 2-3):** Hybrid e Adversarial sono meno frequenti.

✖ **Mitigation Strategy:**
Implementazione di **Class Weights** nella Cross-Entropy Loss per penalizzare gli errori sulle classi rare.

Task C: The Context Window Risk

Per rilevare codice Hybrid, il modello deve vedere l'intero snippet. L'analisi della lunghezza evidenzia il rischio di troncamento.



Architectural Constraint Molti Transformer (es. CodeBERT) hanno un limite di 512 token.

Analysis

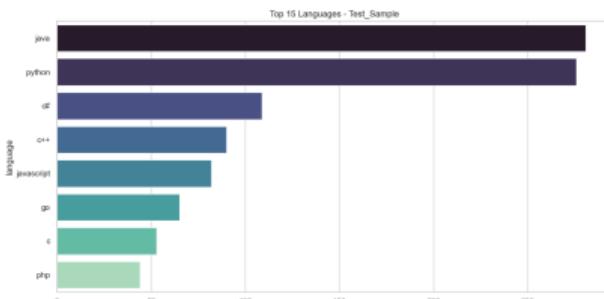
Se un file Hybrid ha la parte AI alla riga 100 (token 800+), un modello standard la taglierà via, predicendo erroneamente "Human".

✓ **Decisione:** Utilizzare modelli Long-Context o strategie di *Sliding Window*.

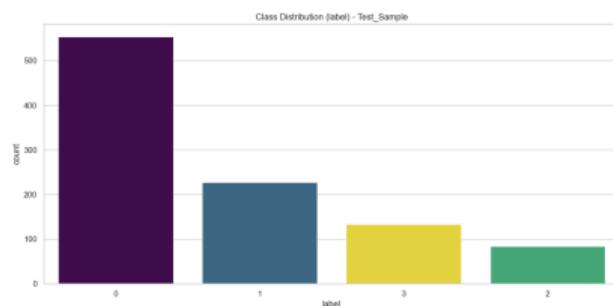
Task C [Test]: Consistency Check

Verifica finale sul Sample Test Set per garantire coerenza distributiva prima della sottomissione.

Ⓐ Test Languages



◎ Target Distribution



Conferma: Il Test set mantiene la frammentazione linguistica e lo sbilanciamento delle classi visto nel Train.

Subtask B: Cascade Inference Strategy

Per gestire il forte sbilanciamento della classe *Human* e la complessità del task multiclasse (11 classi), è stata adottata una strategia a due stadi:

Pipeline Logica

▶ Stage 1: Binary Classifier

- ▶ **Obiettivo:** Filtrare accuratamente *Human* vs *AI Generated*.
- ▶ **Vantaggio:** Protegge dai falsi positivi sulla classe maggioritaria (*Human*).

▶ Stage 2: Family Classifier

- ▶ **Attivazione:** Solo se lo Stage 1 predice "AI".
- ▶ **Obiettivo:** Identificazione Fine-Grained della famiglia (es. GPT, Llama, Mistral).

Architettura Custom: Hybrid Code Classifier

Il cuore del sistema è un modello ibrido basato su **UniXcoder-base**, esteso per catturare sia la semantica che la sintassi stilistica.

Componenti Architetturali:

- 1. Semantic Encoder (UniXcoder):** Utilizza un meccanismo di *Attention Pooling* (somma pesata dei token) per ottenere una rappresentazione del contesto globale, superiore al semplice token [CLS].
- 2. Stylistic Projector:** Una rete feed-forward che proietta 8 feature statistiche manuali in uno spazio latente compatibile con l'encoder.
- 3. Feature Fusion:** Concatenazione degli embedding (Semantico + Stilistico) prima del livello di classificazione finale.

Subtask B: Feature Engineering

Poiché i LLM lasciano "impronte" stilistiche (es. verbosità, convenzioni di naming), il modello analizza le seguenti metriche estratte staticamente:

Stile e Formattazione:

- ▶ Densità dei commenti.
- ▶ Rapporto Snake_case vs CamelCase.
- ▶ Indentazione (media e massima).
- ▶ Lunghezza media delle righe.

Sintassi e Struttura:

- ▶ Token logici (if, for, while).
- ▶ Rapporto spazi bianchi/codice.
- ▶ Densità caratteri speciali.
- ▶ Empty lines ratio.

Nota: Queste feature permettono di distinguere modelli sinteticamente perfetti ma stilisticamente diversi (es. GPT-4 tende a commentare molto più di Llama-2).

Subtask C: La sfida delle "Zone Grigie"

Il Subtask C richiede di distinguere tra 4 classi con confini decisionali sfumati:

- Human (0)** Codice originale scritto da sviluppatori.
- AI-Generated (1)** Codice generato interamente da LLM.
- Hybrid (2) Collaborazione Uomo-Macchina.** Codice molto simile alla classe Human, difficile da isolare semanticamente.
- Adversarial (3) AI Offuscata.** Codice manipolato intenzionalmente per evadere i rilevatori.

Necessità: Un'analisi che vada oltre il contenuto semantico, rilevando anomalie strutturali.

Subtask C: Advanced Feature Extraction

Per identificare l'offuscamento (*Adversarial*) e le modifiche umane (*Hybrid*), è stato implementato un modulo parallelo di estrazione feature:

Metriche di Entropia

- ▶ **Shannon Entropy:** Picchi di entropia indicano nomi di variabili casuali (tipico dell'offuscamento).
- ▶ **Long Strings:** Rilevamento di payload nascosti.

Metriche Strutturali

- ▶ **Nesting Depth:** Gli umani tendono a nidificare di più il codice rispetto alle AI.
- ▶ **Keyword Density:** Distribuzione delle parole chiave del linguaggio.

Strategie di Training e Ottimizzazione

Per entrambi i Subtask, il processo di training è stato ottimizzato per gestire lo sbilanciamento e migliorare la separazione delle classi:

- ▶ **Loss Function Adattiva:** Utilizzo della **Focal Loss** ($\gamma = 2.0$) per penalizzare gli errori sulle classi difficili (Hybrid) rispetto a quelle facili.
- ▶ **Supervised Contrastive Learning (SupCon):** Applicata per compattare i cluster delle stesse classi nello spazio latente e distanziare classi simili (es. Human vs Hybrid).
- ▶ **Robustezza ai Dati (Data Augmentation):** Implementazione del **Random Crop** per input lunghi ($L > 512$):

$$\text{Input} = \text{Code}[idx : idx + L_{max}]$$

Questo costringe il modello ad apprendere dalla logica interna e non solo dagli header/commenti iniziali.

Grazie per l'attenzione

 github.com/giovanniIacuzzo
 giovanni.iacuzzo@unikore.it