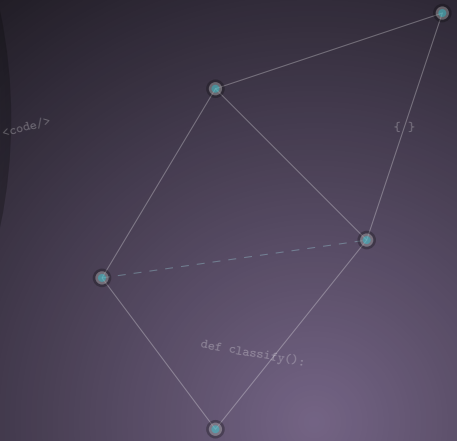
SEMEVAL-2026 // TASK 13

GenAI Code Detection & Attribution



✓ **Full Participation:** Subtasks 1, 2 & 3

Speech & Language Processing Exam

Giovanni Giuseppe Iacuzzo

Università degli Studi di Enna “Kore”

AI & Cybersecurity Student giovannilacuzzo

Roadmap

SemEval-2026

1 | Introduzione e Obiettivi

2 | Analisi dei Dati

3 | Architecture dei modelli

4 | Risultati del Task

5 | Conclusioni

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

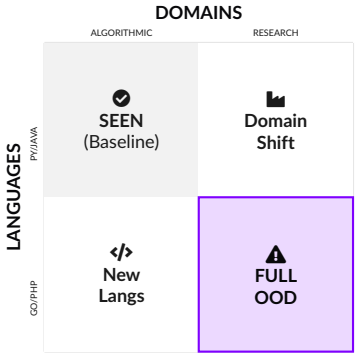
Il Core Task

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).



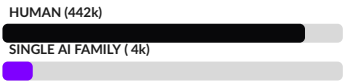
"Obiettivo: Evitare il "memorization overfitting".

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

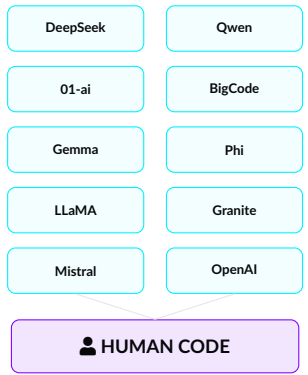
Il dataset è fortemente sbilanciato verso il codice umano, rendendo difficile per il modello apprendere le sfumature delle classi minoritarie (AI).



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)



Subtask C: The Real-World Scenario

Max Dataset: 900k Samples · **4 Classes**

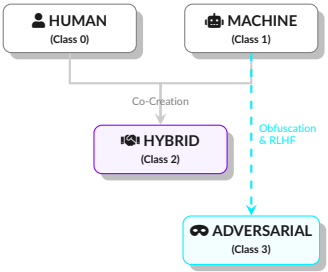
Oltre alla distinzione binaria, introduciamo le **Zone Grigie** dello sviluppo software moderno.

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.

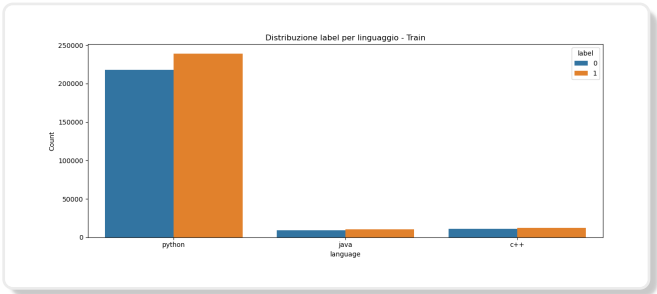


Fig 1. Distribuzione dei linguaggi nel training set.

Composizione

80%

Python

(High Resource Languages)

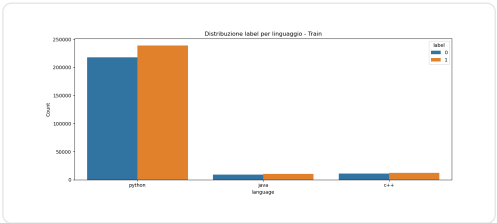
The Risk

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.

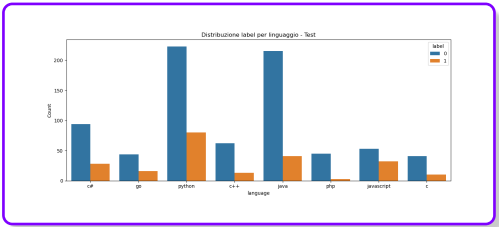
The Shift: Confronto Train vs Test

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

TRAIN (Seen)



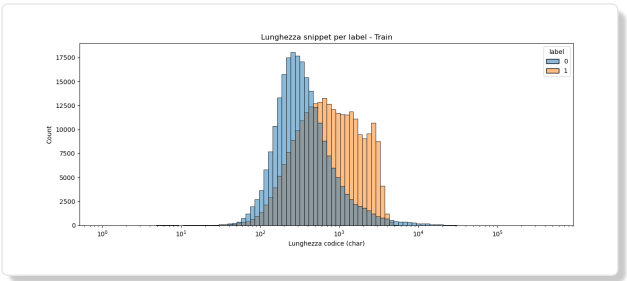
TEST (Unseen)



⚠ **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

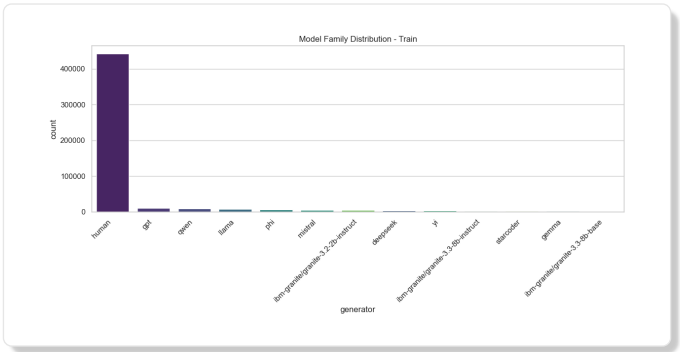
AI Code (Verboso):
I modelli tendono alla *sovrastruttura*: includono spesso commenti esplicativi, gestione errori standard e codice "boilerplate" non richiesto.

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:

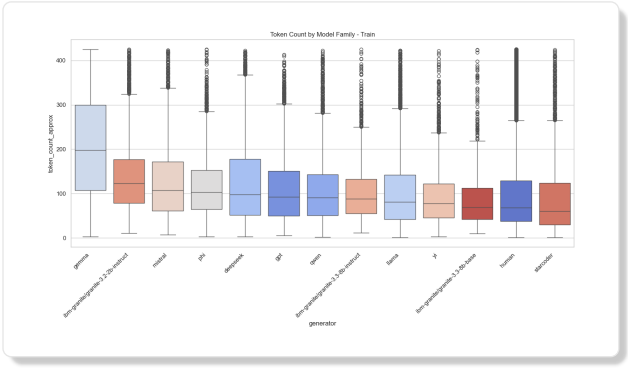
- `llama-2-7b` → **LLaMA**
- `gpt-3.5-turbo` → **GPT**
- `deepseek-coder` → **DeepSeek**

⚖ 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.

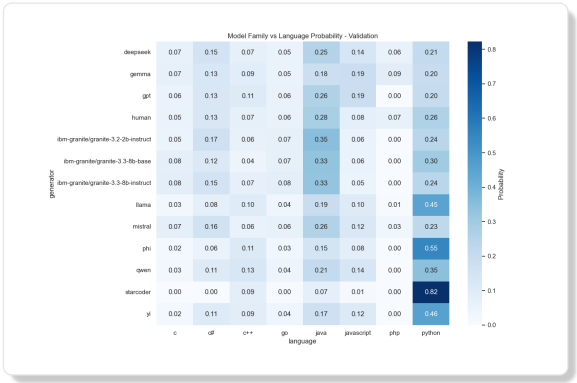
Verbose Models
GPT & Claude. Tendono a spiegare molto, producendo outlier lunghi.

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.



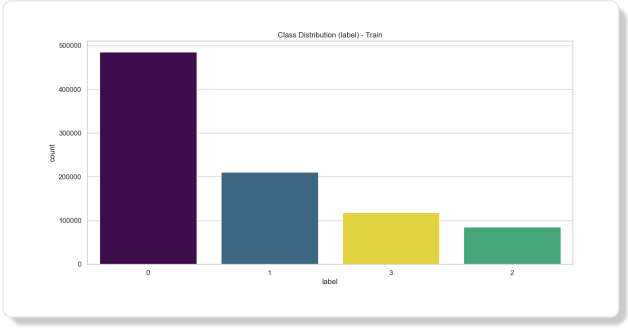
Probability Signature

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.



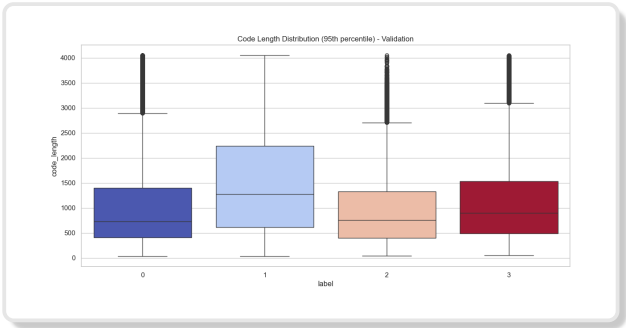
Class Breakdown

- **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

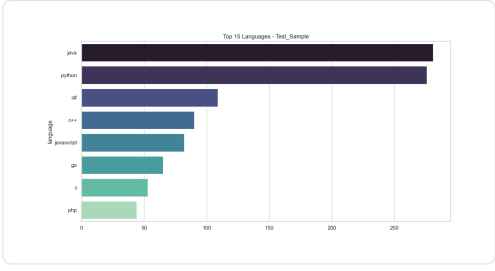
Se un file Hybrid ha la parte AI alla riga 100 (token 800+), un modello standard la taglierà via, predicendo erroneamente "Human".

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

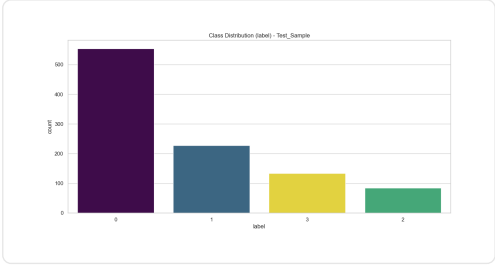
Task C [Test]: Consistency Check

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.

Grazie per l’attenzione

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