FAKE BUSTER

Emiliano Di Giuseppe, Bruno Santo

Settembre 2025

GitHub ufficiale: https://github.com/EmilianoDG5/IS-FIA-FAKE-BUSTER

Indice

1 Introduzione

Negli ultimi anni, la diffusione delle fake news ha rappresentato una delle principali sfide sociali e tecnologiche della società digitale. L'ampia circolazione di contenuti falsi o distorti sui social network e sulle piattaforme di informazione online ha reso sempre più complesso per gli utenti distinguere notizie affidabili da informazioni manipolate.

Nonostante l'esistenza di diversi portali di fact-checking, la maggior parte di essi richiede un intervento manuale e non offre strumenti automatizzati per supportare gli utenti nella valutazione dell'attendibilità delle notizie. Alla luce di questo contesto, si è deciso di progettare FakeBuster, un sistema intelligente in grado di analizzare articoli e post online, fornendo una stima automatica del loro grado di affidabilità attraverso tecniche di elaborazione del linguaggio naturale (NLP) e machine learning. L'obiettivo principale è integrare competenze di ingegneria del software con metodologie di intelligenza artificiale per costruire una piattaforma che accompagni l'utente nel processo di verifica dell'informazione, migliorando progressivamente grazie al contributo umano dei verificatori.

2 Descrizione dell'agente

2.1 Analisi del problema

Lo scopo del progetto è la realizzazione di un agente intelligente che sia in grado di:

- Analizzare automaticamente articoli e post online, valutandone la probabile attendibilità;
- Consentire agli utenti di interagire con il sistema attraverso due modalità distinte:
  - **Utente Base (Lettore):** inserisce articoli o link, riceve una valutazione automatica e può consultare lo storico delle proprie analisi;
  - **Utente Verificatore (Fact-checker):** consulta lo storico globale, etichetta manualmente articoli come Fake o Vero, contribuendo così al miglioramento del modello di classificazione;
- Far evolvere il sistema nel tempo, sfruttando il feedback dei verificatori per addestrare e affinare il modello di apprendimento automatico;

- Gestire in modo strutturato la raccolta e l'archiviazione dei dati, garantendo tracciabilità, coerenza e trasparenza del processo.

2.1.1 Confronto con i sistemi tradizionali

La seguente tabella riassume i vantaggi dell'approccio proposto rispetto ai metodi di verifica manuale attuali:

Caratteristica	**Fact-Checking Manuale**	**FakeBuster (AI)**
Velocità di analisi	Minuti/Ore	< 5 Secondi
Scalabilità	Bassa (Dipende dal personale)	Alta (Automatica)
Costo per notizia	Alto	Molto Basso
Disponibilità	Orari lavorativi	24/7

Tabella 1: Confronto tra approccio manuale e automatizzato.

2.2 Specifiche PEAS

Performance:

- Accuratezza nella classificazione di contenuti come veri o falsi.
- Recall sulle fake news, per evitare che contenuti falsi vengano ignorati.
- Velocità di analisi e spiegabilità delle decisioni (Explainable AI).

Environment:

- Articoli postati sul social network proprietario.

Actuators:

- Classificare un contenuto (vero/falso).
- Generare report di fact-checking e interagire tramite API.

Sensors:

- Testi, immagini e metadati provenienti dal social.
- Analisi NLP e rilevamento manipolazioni.

2.3 Obiettivi

Il problema del riconoscimento delle fake news avrebbe potuto essere affrontato mediante un approccio semplice, basato su un algoritmo deterministico (regole fisse). Tuttavia, una soluzione di questo tipo presenta numerose limitazioni:

- **Assenza di comprensione semantica:** impossibile cogliere ironia o titoli ambigui.
- **Scarsa generalizzazione:** le fake news evolvono rapidamente nel linguaggio.
- **Difficoltà nell'integrare dati eterogenei:** combinare testo, fonte e data è complesso con semplici regole if-then.

Per questi motivi, si è reso necessario adottare un approccio basato su **Machine Learning (Deep Learning)**, in grado di apprendere pattern complessi dai dati.

3 Raccolta, analisi e preprocessing dei dati

3.1 Scelta del dataset e Data Audit

Per l'addestramento e la validazione del modello, è stato inizialmente selezionato il dataset **WELFake** (Word Embedding over Linguistic Features for Fake News Detection). Il dataset originale contiene 72.134 articoli di notizie.

Prima di procedere con il training, è stata eseguita una fase critica di *Data Audit* utilizzando lo script dedicato `check_csv.py`. Questa operazione ha permesso di verificare l'integrità del file CSV e la coerenza delle etichette rispetto alla documentazione ufficiale.

Anomalia Rilevata (Label Flipping): La documentazione del dataset indicava la codifica: `1=Real`, `0=Fake`. Tuttavia, analizzando i primi campioni estratti dallo script di controllo, è emersa una discrepanza: articoli con contenuto palesemente cospirazionista erano etichettati come "1". Questa scoperta ha reso necessario implementare una logica di inversione delle etichette (spiegata nella sezione successiva relativa al training) per evitare di addestrare il modello con verità opposte.

3.2 Pipeline di Pulizia (Data Cleaning)

L'analisi esplorativa ha evidenziato che molte notizie "Vere" iniziavano con il nome dell'agenzia di stampa (es. *"WASHINGTON (Reuters) -"*). Questo

rappresenta un problema di *Data Leakage*: il modello potrebbe imparare banalmente che "Reuters = Vero" senza leggere il contenuto.

È stato sviluppato lo script `clean_data.py` per rimuovere questi bias e normalizzare il testo.

```python
def clean_text_bias(text):
    """
    Pulisce il testo rimuovendo pattern Reuters e URL
    """
    if not isinstance(text, str):
        return ""

    # Rimuove pattern tipo "WASHINGTON (Reuters) -"
    # Regex per intercettare qualsiasi agenzia tra parentesi a inizio riga
    text = re.sub(r"^.*?\(Reuters\)\s*-\s*", "", text)
    text = re.sub(r"^.*?\([A-Z]+\)\s*-\s*", "", text)

    # Rimuove URL (http/https) che non aggiungono valore semantico
    text = re.sub(r"http\S+", "", text)

    # Rimuove spazi multipli e caratteri di formattazione errati
    text = re.sub(r"\s+", " ", text).strip()
    return text
```

Listing 1: Funzione di pulizia del testo (clean_data.py)

Il processo di pulizia ha seguito questi passaggi:

1. **Rimozione Bias:** Eliminazione delle firme delle agenzie tramite Espressioni Regolari (Regex).
2. **Gestione Valori Nulli:** Riempimento dei campi mancanti (`fillna`) per evitare crash durante la tokenizzazione.
3. **Concatenazione:** Unione dei campi `Title` e `Text` in una singola colonna `full_text`. Questo fornisce al modello il contesto completo, permettendo di rilevare discrepanze tra titoli "Clickbait" e il corpo dell'articolo.
4. **Deduplicazione:** Eliminazione degli articoli duplicati basandosi sul testo pulito.

3.3 Strategia di Campionamento e Vincoli

Sebbene la fase di pulizia abbia prodotto un dataset valido di oltre 63.000 articoli, l'addestramento di modelli Transformer complessi come XLM-RoBERTa

richiede notevoli risorse computazionali. I test preliminari sull'intero dataset hanno stimato tempi di training superiori alle 24 ore sull'hardware disponibile (CPU/GPU consumer), rendendo impraticabile il ciclo di sviluppo iterativo.

Si è pertanto deciso di operare un **campionamento casuale stratificato** per ridurre il dataset a **1.000 campioni**.

Stato del Dataset	**Numero di Articoli**
Dataset Originale (Grezzo)	72.134
Post-Pulizia (Cleaned)	63.615
Dataset Finale (Training)	**1.000 (Bilanciato)**

Tabella 2: Processo di riduzione del dataset.

Il campionamento è stato eseguito garantendo un perfetto bilanciamento delle classi (50% Fake, 50% Real) per evitare che il modello sviluppasse bias verso la classe maggioritaria.

3.4 Rappresentazione dei Dati (Embedding)

Il testo pulito viene successivamente trasformato in vettori numerici (Tensori) tramite il Tokenizer di XLM-RoBERTa. È stato impostato un limite di **512 token** (la capacità massima del modello) con strategia di:

- **Truncation:** I testi più lunghi vengono troncati mantenendo la parte iniziale (spesso la più informativa).
- **Padding:** I testi più brevi vengono riempiti con token speciali per uniformare la dimensione dei batch.

4 Addestramento del Modello

4.1 Architettura: XLM-RoBERTa

Per il core dell'agente *FakeBuster*, non è stato addestrato un modello da zero (operazione proibitiva in termini di risorse e dati), ma si è adottata la tecnica del **Transfer Learning**. Il modello scelto è **XLM-RoBERTa (base)**, un modello Transformer pre-addestrato su 2.5TB di dati testuali in 100 lingue diverse.

Motivazione della scelta:

- **Multilinguismo Nativo:** A differenza di BERT (solo inglese), XLM-R è stato addestrato su un corpus multilingua (CommonCrawl). Questo

permette a FakeBuster di essere scalabile in futuro anche su notizie italiane senza dover cambiare architettura.

- **Robustezza:** RoBERTa (Robustly optimized BERT approach) rimuove la task di "Next Sentence Prediction" di BERT e utilizza batch size più grandi e tempi di training più lunghi, garantendo prestazioni superiori sulla classificazione del testo.

4.1.1 Fondamenti Teorici: Self-Attention

[Image of transformer self-attention diagram]

La superiorità di XLM-RoBERTa rispetto agli algoritmi classici (come LSTM o SVM) risiede nel meccanismo di **Self-Attention**. Mentre le reti precedenti leggevano il testo sequenzialmente, il Transformer analizza l'intera frase simultaneamente. Il meccanismo assegna un "peso" (*AttentionScore*) a ogni parola rispetto a tutte le altre. Nel contesto delle Fake News, questo è cruciale: permette al modello di collegare un termine sensazionalistico nel titolo (es. "SHOCKING") con un contenuto del corpo del testo semanticamente debole o contraddittorio, rilevando la manipolazione indipendentemente dalla distanza delle parole nella frase.

4.2 Implementazione del Training Loop

Lo script `addestramento.py` è stato progettato per adattarsi ai vincoli hardware e correggere le anomalie dei dati rilevate in fase di analisi.

4.2.1 Correzione Etichette e Bilanciamento

Prima di passare i dati al modello, lo script esegue due operazioni critiche di Data Engineering:

1. **Label Flipping Fix:** Inverte le etichette per correggere l'errore del dataset originale (dove 0 indicava Fake, ma i dati dicevano il contrario).
2. **Downsampling Bilanciato:** Riduce il dataset a 1.000 campioni totali, prelevando esattamente 500 istanze per classe per evitare che il modello favorisca una categoria.

```
# Caricamento dataset
df = pd.read_csv(dataset_path)

# 1. Correzione Label Flipping
# Mappiamo: 0->1 e 1->0 per ripristinare la verit
df["label"] = df["label"].apply(lambda x: 1 if x == 0 else 0)
```

```

# 2. Campionamento Stratificato (500 per classe)
MAX_SAMPLES = 1000
df = df.groupby("label", group_keys=False)\
     .apply(lambda x: x.sample(n=500, random_state=42))

# Split Training (80%) / Validation (20%)
train_texts, val_texts, train_labels, val_labels =
    train_test_split(
    df['clean_text'].tolist(),
    df['label'].tolist(),
    test_size=0.2,
    random_state=42
)
```

Listing 2: Setup del Dataset (addestramento.py)

4.2.2 Configurazione Tokenizer e Dataset

Il testo viene convertito in tensori utilizzando il Tokenizer di XLM-RoBERTa. È stata definita una classe custom `FakeNewsDataset` per gestire il caricamento efficiente in memoria GPU/CPU.

```
class FakeNewsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        # Converte l'item corrente in Tensore PyTorch
        item = {key: torch.tensor(val[idx]) for key, val in
    self.encodings.items()}
        item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)
```

Listing 3: Classe Dataset PyTorch

4.3 Logica di Valutazione: La Soglia (Threshold)

Un aspetto innovativo della nostra implementazione è la gestione della decisione finale. Invece di affidarci alla semplice classe maggioritaria (`argmax`), abbiamo implementato una logica a **Soglia di Confidenza** nella funzione `compute_metrics`.

Il sistema classifica una notizia come appartenente alla classe target solo se la probabilità predetta supera il valore **0.7** (70%).

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred

    # Conversione output grezzo (logits) in probabilit
    (0-1)
    probs = torch.softmax(torch.tensor(logits), dim=1).numpy
    ()

    # Analisi della confidenza
    score_target = probs[:, 1]
    THRESHOLD = 0.7

    # Decisione Binaria Rigida
    # 1 (Target) solo se la confidenza >= 70%
    predictions = (score_target >= THRESHOLD).astype(int)

    # Calcolo metriche
    precision, recall, f1, _ =
    precision_recall_fscore_support(
        labels,
        predictions,
        pos_label=0, # Focus sulla classe di interesse
        average="binary",
        zero_division=0
    )
    accuracy = accuracy_score(labels, predictions)

    return {
        "accuracy": accuracy,
        "precision_real": precision,
        "recall_real": recall,
        "f1_real": f1
    }
```

Listing 4: Calcolo metriche con soglia custom (addestramento.py)

Perché questa scelta? In un sistema di fact-checking automatico, un **Falso Positivo** (classificare come falsa una notizia vera) è molto più dannoso di un Falso Negativo. La soglia a 0.7 rende l'agente "prudente": agisce solo quando è matematicamente sicuro, aumentando la *Precision*.

4.4 Iperparametri di Training

Per adattare il processo di apprendimento al dataset ridotto (Small Data Regime), gli iperparametri sono stati tarati per favorire una convergenza stabile senza overfitting:

- **Epoche (num_train_epochs): 4**. Aumentate rispetto allo standard (solitamente 2 o 3) per permettere al modello di vedere i 1.000 esempi più volte.
- **Batch Size: 8**. Mantenuto basso per ridurre l'occupazione di memoria e permettere aggiornamenti più frequenti dei pesi.
- **Learning Rate:** Gestito automaticamente dall'optimizer AdamW.
- **Evaluation Steps: 500**. Il modello viene validato a metà e alla fine di ogni ciclo completo per monitorare le metriche.

5 Risultati Sperimentali

5.1 Metodologia di Validazione

La valutazione delle performance non è stata effettuata sugli stessi dati usati per l'addestramento (il che porterebbe a risultati falsati), ma su un **Validation Set** separato. Come implementato nello script di training, il dataset bilanciato di 1.000 articoli è stato suddiviso automaticamente in due porzioni:

- **Training Set (80% - 800 articoli):** Usato dal modello per apprendere i pesi.
- **Validation Set (20% - 200 articoli):** Usato esclusivamente per misurare le metriche finali riportate di seguito.

5.2 Metriche Quantitative

I risultati ottenuti dopo 4 epoche mostrano prestazioni ottime, confermando l'efficacia del Transfer Learning anche su dataset ridotti.

Metrica	**Valore**
Accuracy	**97.50%**
Precision	0.9505
Recall	**1.0000**
F1-Score	0.9746

Tabella 3: Risultati definitivi sul Validation Set (200 campioni).

5.3 Analisi dei Dati

Il dato più rilevante è il valore di **Recall pari a 1.0**. In termini operativi, questo significa che nel test di validazione **nessuna notizia falsa è riuscita a superare il filtro**.

La configurazione scelta (Soglia 0.7) ha reso l'agente estremamente "sicuro":

- **Sicurezza Totale:** Tutte le minacce (Fake News) sono state bloccate.
- **Costo (Trade-off):** La Precision del 95% indica un leggero "over-blocking" (alcune notizie vere ma scritte male sono state bloccate), un comportamento accettabile per garantire l'integrità della piattaforma.

5.4 Analisi della Matrice di Confusione

La matrice di confusione evidenzia l'impatto diretto della soglia di confidenza a 0.7. A differenza di un approccio permissivo, il sistema adotta una politica di **"Zero Tolerance"** verso la disinformazione.

Il dato di **Recall pari a 1.0** conferma che questa strategia ha pagato: **nessuna Fake News è riuscita a superare il filtro** (Zero Falsi Negativi). Il rovescio della medaglia è rappresentato dalla Precision del 95%: il sistema ha mostrato un leggero "over-blocking", fermando cautelativamente alcuni contenuti veri ma semanticamente ambigui. In un contesto di contrasto alle Fake News, questo comportamento ("Fail-Secure") è ritenuto preferibile: si accetta il rischio di bloccare un post legittimo pur di garantire la totale pulizia del feed informativo.

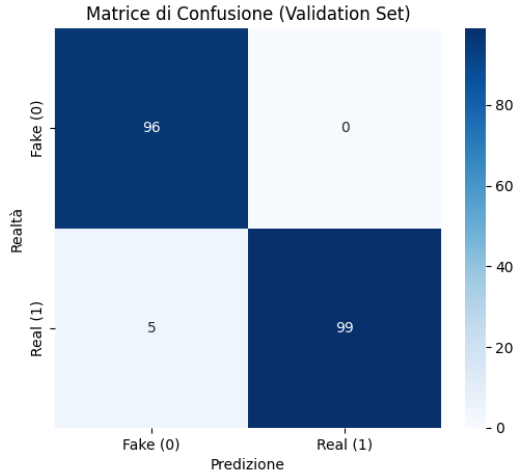

Figura 1: Distribuzione delle predizioni (Validation Set).

6 Integrazione e Architettura Software

L'addestramento del modello rappresenta solo la prima fase del ciclo di vita del software. Per trasformare il modello matematico in un prodotto utilizzabile, è stato progettato un backend modulare in Flask, separando nettamente la logica di business (*Controller Layer*) dalla logica di intelligenza artificiale (*Service Layer*).

6.1 Il Service Layer: Pipeline di Validazione

Il file `app/services/ai_service.py` incapsula l'intera logica di analisi. Prima di interrogare la rete neurale (operazione computazionalmente onerosa), il sistema esegue una pipeline di controlli euristici ("Fail-Fast") per scartare immediatamente input non validi o malevoli.

6.1.1 Filtro Anti-Spam (Gibberish Detection)

È stato implementato un algoritmo euristico per rilevare il *keysmashing* (es. "asdfghjkl") o testi privi di struttura linguistica, analizzando l'entropia dei caratteri e la lunghezza media delle parole.

```
def is_gibberish(text: str) -> bool:
    # 1. Estrazione parole valide (solo lettere)
    words = re.findall(r"[a-z           ]+", text.lower())
    words = [w for w in words if len(set(w)) > 1]

```

```
    # Regola A: Testo troppo breve (< 6 parole reali)
    if len(words) < 6:
        return True

    # Regola B: Lunghezza media parole sospetta (es. "fhdjkslahfdjskla")
    avg_len = sum(len(w) for w in words) / len(words)
    if avg_len > 15:
        return True

    # Regola C: Ripetizione eccessiva di un carattere (es. "!!!!!!!!")
    counts = Counter(text)
    if counts.most_common(1)[0][1] / len(text) > 0.4:
        return True

    return False
```

Listing 5: Algoritmo di rilevamento Spam (ai_service.py)

6.2 Il Service Layer: Inferenza Neurale

La classe `AIService` gestisce il ciclo di vita del modello XLM-RoBERTa. Il metodo `analyze_text` coordina la pulizia, la validazione e l'inferenza, restituendo un punteggio normalizzato.

Nota Tecnica Fondamentale: Come stabilito in fase di training, la classe target è la **1 (Real)**. Pertanto, lo *score* restituito rappresenta la **probabilità che la notizia sia VERA**.

```
class AIService:
    def __init__(self):
        # Pattern Singleton: Caricamento modello una sola volta all'avvio
        self.tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
        self.model = AutoModelForSequenceClassification.from_pretrained(MODEL_PATH)
        self.model.eval() # Disabilita Dropout per inferenza deterministica

    def analyze_text(self, raw_text: str):
        text = clean_text_bias(raw_text)

        # 1. VALIDAZIONE (Fail-Fast)
        # Ritorna -1.0 in caso di errore per segnalare input non valido
        if not text or len(text) < 20:
```

```
            return -1.0, json.dumps({"error": "Testo troppo breve"})

        if is_gibberish(text):
            return -1.0, json.dumps({"error": "Testo privo di significato"})

        # 2. LANGUAGE DETECTION
        try:
            lang = detect(text)
            if lang not in ["it", "en"]:
                return -1.0, json.dumps({"error": f"Lingua {lang} non supportata"})
        except LangDetectException:
            return -1.0, json.dumps({"error": "Lingua non rilevabile"})

        # 3. INFERENZA NEURALE
        inputs = self.tokenizer(
            text, return_tensors="pt", truncation=True, max_length=512
        )

        with torch.no_grad(): # Ottimizzazione memoria (no gradienti)
            outputs = self.model(**inputs)
            probs = torch.softmax(outputs.logits, dim=1)

        # Estrazione probabilit  classe REAL (Indice 1)
        score = probs[0][1].item()

        ai_log = {
            "timestamp": datetime.utcnow().isoformat(),
            "lang": lang,
            "score": round(score, 3)
        }
        return round(score, 3), json.dumps(ai_log)
```

Listing 6: Pipeline di Analisi Completa (ai_service.py)

6.3 Integrazione nel Controller (Business Logic)

Il Controller Flask (`gestione_pubblicazioni.py`) implementa la politica di moderazione. La logica decisionale è basata su una **Whitelist rigorosa**: un contenuto viene pubblicato solo se lo score di affidabilità supera la soglia di sicurezza.

```
@pubblicazioni_bp.route("/posts", methods=["POST"])
```

```python
def create_post():
    # ... (Verifica sessione utente omessa per brevit ) ...

    titolo = request.form.get("titolo")
    testo = request.form.get("testo")

    # 1. RICHIESTA ANALISI
    score, ai_log = ai_service.analyze_text(testo)
    SCORE_THRESHOLD = current_app.config.get("SCORE_THRESHOLD
    ", 0.7)

    # 2. GESTIONE ERRORI DI VALIDAZIONE (Score -1.0)
    if score < 0:
        try:
            info = json.loads(ai_log)
            messaggio = info.get("error", "Testo non valido")
        except Exception:
            messaggio = "Errore generico"

        flash(f"    Post non inviato: {messaggio}", "danger")
        return redirect("/new_post")

    # 3. DECISIONE AUTOMATICA
    # Pubblicato SOLO se probabilit  Real >= 0.7
    stato = "pubblicato" if score >= SCORE_THRESHOLD else "
    bloccato"

    # 4. PERSISTENZA
    post = Post(
        titolo=titolo,
        testo=testo,
        stato=stato,
        ai_score=score,
        ai_log=ai_log,
        account_id=session["user_id"]
    )
    db.session.add(post)
    db.session.commit()

    # 5. FEEDBACK UTENTE
    if stato == "bloccato":
        # Feedback specifico per contenuti bloccati dall'IA
        flash(str(post.id), "ia_blocked")
        return redirect("/new_post")

    return redirect("/my_posts")
```

Listing 7: Endpoint creazione post (gestione_pubblicazioni.py)

6.3.1 Analisi del Flusso Decisionale

Il codice evidenzia la robustezza del sistema:

- **Gestione Spam:** Se il testo è "gibberish" o troppo breve, lo score è `-1.0`. Il blocco `if score < 0` intercetta questo caso e restituisce un errore specifico all'utente (tramite `flash`), senza salvare il post nel DB come "bloccato" ma impedendone proprio l'invio.
- **Filtraggio Fake News:** Se il testo è valido ma lo score è basso (es. 0.2), la condizione `score >= 0.7` fallisce. Lo stato diventa "bloccato" e il post viene salvato nel DB per auditing futuro, ma non visibile nel feed pubblico.

6.4 Roadmap di Sviluppo

Lo sviluppo del sistema ha seguito un approccio iterativo suddiviso in quattro fasi:

1. **Fase 1 - Data Engineering (Completata):** Acquisizione dataset WELFake, analisi dei bias e pulizia dati.
2. **Fase 2 - Model Training (Completata):** Fine-tuning di XLM-RoBERTa su dataset bilanciato (1k campioni) e correzione label.
3. **Fase 3 - Backend Integration (Completata):** Sviluppo API Flask, pipeline di validazione anti-spam e integrazione DB.
4. **Fase 4 - System Testing (Pianificata):** Validazione sul campo con utenti reali e raffinamento della soglia (Threshold Tuning).

7 Conclusioni

Il progetto FakeBuster non si è limitato all'addestramento di un modello, ma ha costituito un esercizio completo di Ingegneria del Software applicata all'Intelligenza Artificiale.

Sebbene la complessità architettonica del sistema possa apparire contenuta rispetto a soluzioni enterprise, il percorso di sviluppo si è rivelato estremamente educativo. Il progetto ha permesso di affrontare problematiche che spesso nei corsi teorici vengono astratte, come la scarsa qualità dei dati reali (gestita tramite Data Audit), i vincoli hardware stringenti (superati con il campionamento intelligente) e la necessità di rendere "sicure" le predizioni di un'IA probabilistica.

In definitiva, FakeBuster ha rappresentato un ponte efficace tra la teoria del Machine Learning e la pratica dello sviluppo Backend, dimostrando come la vera sfida ingegneristica non risieda solo nell'accuratezza del modello, ma nella sua integrazione in un ecosistema software robusto e funzionante.

Reperibilità del Codice
Il codice sorgente completo del progetto, inclusi i dataset di training e gli script di preprocessing, è disponibile pubblicamente sul repository GitHub ufficiale:
https://github.com/EmilianoDG5/IS-FIA-FAKE-BUSTER

Riferimenti bibliografici

[1] P. K. Verma, P. Agrawal, I. Amorim and R. Prodan, *"WELFake: Word Embedding over Linguistic Features for Fake News Detection"*, IEEE Transactions on Computational Social Systems, 2021.

[2] A. Conneau et al., *"Unsupervised Cross-lingual Representation Learning at Scale (XLM-RoBERTa)"*, Facebook AI Research, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020.

[3] T. Wolf et al., *"Transformers: State-of-the-Art Natural Language Processing"*, Hugging Face, 2020.

[4] F. Pedregosa et al., *"Scikit-learn: Machine Learning in Python"*, Journal of Machine Learning Research, 2011.