Politecnico di Torino

Politecnico di Torino

Computer Engineering: AI & Data Analytics

A.a. 2024/2025

Sessione di laurea Ottobre 2025

Rilevamento e Mitigazione di Allucinazioni in Large Language Models

Relatori:

Flavio Giobergia

Claudio Savelli

Candidato:

Alessandro Romeo

Sommario

La tesi affronta il *rilevamento delle allucinazioni* nei Large Language Models (LLM), ovvero la generazione di contenuti plausibili ma non supportati dai fatti. Il focus è migliorare **affidabilità** e **tracciabilità** a *granularità di token*, utile in scenari applicativi sensibili.

Come riferimento sperimentale si è qui adottato MU-SHROOM (val EN/IT, 50 esempi). A partire dal lavoro iniziale del team MALTO, che combina *probabilità token-level* di un LM con un segnale locale di *Natural Language Inference* (NLI), vengono progettate e valutate tre varianti di pipeline:

- **No Fluency**: solo LM+NLI, senza vincolo di perplessità.
- **Fluency**: LM+NLI con soglia di perplessità per accettare sostituzioni.
- **Mask**: selezione con contraddizione NLI e rifinitura di fluenza.

Protocollo e metriche. Tutti i metodi sono valutati sullo *stesso* split e con lo *stesso* valutatore (allineamento char↔token; **IoU** su hard labels a carattere; **Pearson** su soft labels). SelfCheckGPT e CCP sono adattati al token-level per la comparabilità.

Risultati principali. In inglese, **No Fluency (NLI large)** ottiene **IoU 0.309** e **Pearson 0.337**; *No Fluency (NLI base)* conserva 95% della correlazione (0.320) con IoU inferiore (0.256). Le varianti *Fluency (NLI large)* e *Mask (NLI large)* raggiungono rispettivamente *0.294/0.312* e *0.289/0.299*. In italiano, *No Fluency (NLI large)* ottiene *0.261/0.253*, evidenziando trasferibilità con gap moderato. Le pipeline superano le baseline della challenge (neural) e i metodi esterni (SelfCheckGPT, CCP) con valutatore condiviso.

Contributi. (i) *Varianti di pipeline* per detection/mitigation a token; (ii) *valutazione unificata e riproducibile* su MU-SHROOM EN/IT; (iii) *ottimizzazione sistematica* di η, k, `prob_thresh` e ablation della soglia di fluenza; (iv) *analisi qualitativa* con listing completi in Appendice.

Conclusioni e sviluppi futuri. Il segnale combinato *LM token-prob + NLI* è orientato alla fattualità e guida le migliori prestazioni, con **No Fluency** che rappresenta il miglior compromesso accuratezza/efficienza. Tra le priorità: dataset *token-level* multilingua e scorer di riferimento; NLI robusto ma compatto (anche multilingue); estensione a nuove lingue (es. spagnolo, francese, cinese, giapponese); integrazione di fact-checking e soglie/astensione basate su incertezza.

Ringraziamenti

Desidero ringraziare innanzitutto i miei relatori, il Professor Flavio Giobergia e il Dottor Claudio Savelli, per il supporto, i suggerimenti e la disponibilita' dimostrati durante tutto il percorso di tesi.

Ringrazio inoltre i miei amici più stretti, Giuseppe, Andrea, Marco, Gabriele, Matteo, Guglielmo, Manuel e Matteo, per la loro amicizia e il sostegno che mi hanno offerto da sempre.

Un grazie sentito va anche a mia sorella, Viviana, e i miei cugini Alessio e Roberta e mio zio Tanino, per essere stati una presenza costante e incoraggiante.
Ringrazio tutta la mia famiglia, in particolare i miei nonni, per il loro affetto e il loro sostegno incondizionato.

Ai miei genitori, per avermi cresciuto, facendomi diventare la persona che sono oggi. Vi ringrazio, perché mi avete insegnato a credere in me stesso e perseguire sempre obiettivi ambiziosi. Il vostro supporto e la vostra fiducia in me sono stati fondamentali, e per questo vi sono profondamente grato. Grazie davvero, per essere stati i migliori genitori che potessi desiderare.

Ad Agnese, per la sua pazienza, il suo amore, per la felicità e la serenità che mi ha dato da sempre.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Nel corso degli ultimi anni, i *Large Language Models* (LLMs) hanno trasformato radicalmente il panorama del *Natural Language Processing* (NLP). Modelli come GPT, Claude o LLaMA sono capaci di scrivere testi dettagliati, tradurre fra lingue differenti e rispondere a domande complesse conservando uno stile (ormai, poco) sorprendentemente fluido. Ciononostante, questa capacità di generare un testo coerente e fluido, non coincide sempre con l'affidabilità dei contenuti: gli stessi modelli che producono sentenze grammaticalmente impeccabili possono originare informazioni prive di riscontro reale: tale fenomeno è comunemente definito **hallucination**. Data l'assenza di un effettivo "senso della verità" all'interno dei modelli, vengono posti limiti concreti all'utilizzo degli LLM in domini sensibili, in cui accuratezza e trasparenza rappresentano requisiti essenziali. La letteratura evidenzia come l'accoppiata "competenza linguistica" e "assenza di grounding" (ovvero la mancanza di "ancoraggio" a fonti esterne affidabili) renda il fenomeno trasversale a domini e lingue, alimentando episodi che spaziano dalla generazione di riferimenti legali inesistenti alla citazione di trial clinici mai avviati [1].

1.1 Che cosa si intende per hallucination

Il termine *allucinazione* definisce l'output di un'informazione non supportata da fatti o non coerente con il contesto disponibile, un comportamento riscontrato in quasi tutte le famiglie LLM indipendentemente dalla lingua o dal dominio applicativo [1]. Il fenomeno non si riduce a errori casuali: trae origine dalla natura probabilistica degli LLM, addestrati a prevedere la continuazione più plausibile di una sequenza anche in assenza di prove verificabili. Per questo motivo le allucinazioni si verificano sia in scenari aperti - conversazione, risposta a domande, generazione creativa - sia in compiti apparentemente più vincolati come il riassunto automatico o la semplificazione di testi [2]. In letteratura due macroclassi meritano di essere discriminate [1, 3]:

- **Hallucinations intrinseche**, originate da fallacie cognitive o inferenze erronee generate all'interno del modello, come le tipiche imprecisioni riscontrate nei calcoli aritmetici o nel ragionamento logico [4];
- **Hallucinations estrinseche**, sono causate dalla mancanza di conoscenza esterna o dalla mancanza di riferimento a fonti affidabili, che portano alla creazione di informazioni che diventano inesistenti o distorte.

Le due dimensioni possono esistere simultaneamente nello stesso risultato: una risposta può risultare intrinsecamente errata e, al contempo, rivelare un disallineamento rispetto al contesto

presentato dall'utente. Pertanto, le allucinazioni si manifestano come un problema strutturale che coinvolge il modello, i dati e l'interazione con l'utente, differenziandosi da meri errori tipografici in quanto compromettono direttamente la credibilità semantica del testo.

Un caso concreto aiuta a comprendere la portata del fenomeno:

> **Prompt:** "Quale città ha ospitato le Olimpiadi invernali del 2022?"
> **Risposta del modello:** "Le Olimpiadi invernali del 2022 si sono svolte a Sapporo, in Giappone, e sono state inaugurate dall'imperatore Naruhito."
> **Perché è un'allucinazione:** l'edizione 2022 si è tenuta a Pechino e nessuna cerimonia coinvolgeva Sapporo.

L'output conserva un registro professionale e impiega nomi credibili, tuttavia, presenta un'informazione che è erronea dal punto di vista fattuale e che un lettore poco attento potrebbe accettare senza procedere a verifiche. Tale fenomeno è stato registrato anche in contesti legali, giornalistici e sanitari, comportando effetti a catena che includono la circolazione di citazioni fabricate o diagnosi non verificate [1]. Questi eventi risultano particolarmente insidiosi in ambiti delicati, dove un dato impreciso può provocare conseguenze tangibili.

1.2 Tassonomie e forme principali

La varietà dei fenomeni ha portato la comunità a sviluppare tassonomie che facilitano il confronto e la classificazione del fenomeno [1, 3]. Un primo aspetto riguarda il rapporto con il contesto: si fa riferimento a **factual hallucination** se l'affermazione non è d'accordo con la realtà oggettiva, mentre le **faithfulness hallucinations** violano le informazioni incluse nel messaggio o in eventuali documenti giustificativi. Questa caratterizzazione ha un valore particolare nel riepilogo astratto automatico, dove un testo può essere pienamente coerente ma includere dettagli mai citati nel documento sorgente [2]. Una seconda dimensione distingue il contenuto generato in base alla natura della conoscenza coinvolta: ci sono degli errori **entity-centric** riguardo ai nomi propri e alle date, numerico **numeriche** riguardanti quantità e misure, oppure **reasoning hallucinations** derivanti da una logica di catena imprecisa, spesso responsabile di passaggi impliciti [3]. Altri lavori propongono **pragmatic hallucinations**, in cui il modello in realtà genera risposte formalmente corrette ma inadeguate allo scopo comunicativo, ad esempio trascurando vincoli pragmatici o linee guida etiche [1].

Altre tassonomie si concentrano sulla grana del fenomeno, che può verificarsi a livello di documento, frase o singolo token. I più recenti sistemi di valutazione privilegiano le annotazioni **token-level**, poiché consentono un'analisi dettagliata delle parti effettivamente problematiche e facilitano interventi specifici di correzione degli errori [5]. È inoltre opportuno distinguere tra contenuti completamente fabbricati **(fabrication)**, distorsione dei fatti reali **(distortion)** e risposte evasive/vaghe **(non-fabrication)**, che pur conservando una forma corretta sono informativamente nulle [3, 6]. Queste tassonomie non si escludono a vicenda: a livello pratico, sono combinate per guidare la progettazione dei set di dati, le metriche e gli algoritmi di valutazione e per imporre priorità diverse a seconda del dominio (ad esempio sanitario rispetto a quello conversazionale).

1.3 Origini del fenomeno

Le origini delle allucinazioni sono diversificate e frequentemente si manifestano simultaneamente, come dimostrato dalle principali indagini sull'argomento [1, 3]:

- **Qualità e copertura dei dati di addestramento**: corpora obsoleti, rumorosi o distorti portano il modello a interiorizzare conoscenze imperfette, che riappaiono soprattutto quando vengono richiesti dettagli specifici;
- **Generalizzazione fuori distribuzione**: di fronte a contesti non familiari, il LLM ricorre a relazioni falsamente correlate e genera collegamenti inesistenti, privilegiando modelli superficiali appresi dalla formazione;
- **Ambiguità del prompt e del task**: input ambigui o imprecisi fanno sì che il modello "riempi gli spazi vuoti" partendo da presupposti probabilisticamente ragionevoli per mostrare l'impatto quando l'utente cerca risposte di tipo assertivo;
- **Strategie di decoding**: una temperatura più elevata, un campionamento nucleus aggressivo o la pressione per generare lunghe porzioni di testo possono accentuare le deviazioni locali al punto da trasformarle in errori gravi;
- **Limiti della finestra contestuale e della memoria**: le informazioni distanti vengono dimenticate o distorte, rendendo difficile preservare la coerenza tra generazioni estese;
- **Misalignment tra obiettivi di addestramento e comportamento atteso**: l'ottimizzazione della probabilità del token successivo non garantisce di per sé la veridicità o la conformità contestuale e le tecniche di allineamento come RLHF non sempre tengono conto di tutti i casi d'uso.

Questi fattori incidono sulla percezione di affidabilità dei sistemi e rendono necessarie strategie di controllo speciali, che vanno dal controllo automatico delle fonti alla calibrazione dell'incertezza. L'interazione tra cause interne (come parametri, tokenizzazione e funzioni di perdita) ed esterne (tra cui la tempestività della qualità e la disponibilità dei documenti) rende il fenomeno difficilmente affrontabile con un'unica soluzione, e richiede quindi lo sviluppo di linee di indagine parallele dedicate alla valutazione, alla compensazione e alle interfacce utente [3].

1.4 Impatto applicativo

Gli effetti delle hallucinations vanno oltre la sfera accademica. In ambito **medico** una diagnosi inventata può compromettere la salute del paziente; nel settore **legale** citazioni o precedenti inesistenti possono fuorviare professionisti e utenti; in contesti **educativi** la diffusione di nozioni errate compromette la fiducia negli strumenti didattici automatizzati. Il problema diventa quindi anche etico: affidarsi a un modello che "si inventa" fatti rischia di generare disinformazione e decisioni sbagliate su larga scala.

1.5 Perché rilevare e mitigare

Per mitigare tali rischi, è indispensabile l'adozione di metodi capaci di identificare automaticamente i passaggi non affidabili. La recente letteratura scientifica presenta tre categorie principali di strategie: approcci **auto-referenziali**, che interrogano il modello o sue varianti al fine di valutare il livello di confidenza [7, 4, 8]; metodi **basati su riferimenti**, che confrontano l'output con fonti esterne strutturate o con motori di ricerca [5, 9]; e soluzioni **consapevoli dell'incertezza**, che impiegano ensemble leggeri o misure di entropia per identificare token sospetti [10, 11]. Le pipeline ibride tendono a integrare questi segnali con controlli linguistici o di fluidità al fine di massimizzare la copertura degli errori. Le tecniche di *hallucination detection* svolgono differenti funzioni:

- stanno monitorando la qualità delle risposte, segnalando agli utenti le parti incerte;
- consentono procedure di correzione o rigenerative correlate per migliorare la resistenza del sistema;
- forniscono preziose misure di fiducia per aiutarci a stabilire relazioni di fiducia tra esseri umani e modelli.

La recente competizione **Mu-SHROOM** ha promosso la standardizzazione di questo campo attraverso l'adozione di parametri oggettivi, come Intersection-over-Union, correlazione di Pearson, nonché benchmark multilingue. Un segno distintivo delle soluzioni più recenti è il focus a livello di *token*, considerato una quantità più informativa rispetto alla valutazione aggregata su frasi o paragrafi. Questa strategia corrisponde alla pipeline delineata in questo lavoro e combina indicazioni di segnali di fiducia, controlli di inferenza e controllo lessicale per suggerire interventi puntuali sulle campate problematiche.

1.6 Contributo e struttura della tesi

Il lavoro presentato in questa tesi si inserisce in tale quadro, suggerendo una pipeline per la rilevabilità e la mitigazione **token-level**. Partendo dall'approccio **MALTO** sviluppato per il compito congiunto Mu-SHROOM, vengono introdotte tre varianti successive che incorporano la probabilità derivata da un modello di riferimento, controlli di *Natural Language Inference* e, nei casi più avanzati, controlli di influenza o rigenerazione locale utilizzando modelli più piccoli. Lo scopo è duplice: testare l'efficacia di strategie *fine-grained* nel ridurre le allucinazioni e studiare i compromessi tra copertura degli errori, fluidità testuale ed efficacia computazionale.

Il resto del documento è organizzato come segue:

- il **Capitolo 2** offre una panoramica dei principali contributi della letteratura;
- il **Capitolo 3** descrive in dettaglio la pipeline adottata e le varianti sperimentate;
- il **Capitolo 4** illustra il dataset annotato impiegato per la valutazione;
- il **Capitolo 5** presenta i risultati quantitativi e l'analisi qualitativa dei casi studio;
- il **Capitolo 6** conclude con una sintesi critica e le possibili estensioni future.

Questo quadro cerca di fornire un contesto armonioso, che porta dalle motivazioni iniziali all'analisi dei risultati e alle potenziali direzioni per ulteriori studi.

Capitolo 2

Stato dell'Arte

La generazione automatica del linguaggio si è enormemente beneficiata dell'arrivo dei *Large Language Models* (LLM). A prescindere da prestazioni sbalorditive negli esercizi eterogenei, questi modelli tendono ad emettere contenuto non sorretto dai fatti oppure inconsistente con il contesto presente.Negli anni recenti la comunità dei ricercatori si è così dedicata allo studio delle *hallucinations*, mettendo al centro tassonomie, metriche e metodologie per scoprirle e contenerele. In questo capitolo si presenta una rassegna critica dei contributi più significanti, dando ampio peso agli espedienti token-level che rappresentano lo stato dell'arte più recente.

2.1 Terminologia e tassonomie

Il Capitolo 1 ha già ricostruito le principali categorie di hallucination e ne ha illustrato le ricadute applicative. In questa sezione richiamo le distinzioni più utili per orientare la lettura dello stato dell'arte, soffermandomi sugli aspetti che influenzano progettazione e valutazione degli algoritmi.

- **Factuality vs. Faithfulness**: la prima dimensione valuta l'allineamento con evidenze oggettive, mentre la seconda analizza la coerenza rispetto all'input o alle fonti disponibili. I metodi descritti nelle sezioni successive si situano lungo questo asse, ora enfatizzando la verifica rispetto a una base di conoscenza, ora il confronto con il contesto immediato.

- **Fabrication vs. Non-fabrication**: in aggiunta alle allucinazioni completamente congegnate, la letteratura contempla risposte evasive o imprecise che, sebbene non formulino affermazioni false esplicite, evitano di soddisfare la richiesta dell'utente [6]. Sottolineare questa distinzione è fondamentale per comprendere le ragioni per cui alcune pipeline integrano verifiche fattuali e controlli semantici.

- **Granularità**: l'analisi può essere condotta a livello di documento, frase o token. L'obiettivo di questa ricerca è incentrato sulle tecniche **token-level**, che consentono di identificare con accuratezza gli span problematici e di sostenere procedure di correzione automatica o semi-automatica.

Queste categorie fungono da bussole concettuali: non si escludono a vicenda, ma insieme definiscono presupposti, limiti e obiettivi per ciascuna tecnica.

2.2 Metriche e benchmark

La valutazione delle prestazioni nella discriminazione delle allucinazioni richiede metriche che possano catturare entrambe la precisione nella previsione e la localizzazione. Nelle righe successive, vengono riprese le metriche più utilizzate negli studi analizzati. Nel seguito vengono riepilogate le più utilizzate nei lavori analizzati.
Intersection-over-Union (IoU). Calcola la sovrapposizione tra le tratte di testo trovate dal sistema e quelle annotate dagli esperti, prendendo il rapporto tra l'intersezione e l'unione. Un valore elevato suggerisce che il modello ha indicato pezzi di testo simili a quelli ritenuti effettivamente allucinati. Si può pensare a due evidenziatori: più le aree colorate si sovrappongono, maggiore è l'IoU.
F1 token-level. Prende la media tra precisione e richiamo dei tag binari assegnati a ciascun token, fornendo alcune informazioni sulla capacità della tecnica di rifiutare il meno possibile, senza perdere troppi esempi. È la metrica utilizzata, come standard, nella sfida sulla classificazione a grana fine, essendo la media armonica tra i due indicatori: non appena uno degli indicatori scende, anche la metrica F1 scende.
Correlazioni di Pearson/Spearman. Quando il modello fornisce un punteggio di confidenza continuo sull'allucinazione, le correlazioni valutano il grado di accordo con i punteggi forniti dagli esseri umani, discriminando tra relazioni lineari (Pearson) e monotone (Spearman). In pratica, l'analisi verifica se i passaggi ritenuti più importanti dagli annotatori ricevono punteggi elevati anche dal sistema e viceversa.
Area Under Precision-Recall (AUC-PR). Questo metodo è particolarmente efficace per dataset squilibrati, poiché analizza l'andamento di precisione e richiamo in funzione delle variazioni della soglia decisionale. Una curva che presenta un'area estesa indica che il procedimento conserva prestazioni elevate anche al variare del punto operativo, dimostrando così una certa flessibilità nell'affrontare situazioni con requisiti differenti, come un numero ridotto di falsi positivi o un elevato richiamo.

La mancanza di benchmark condivisi ha ostacolato per anni il confronto tra soluzioni; la shared task **Mu-SHROOM** [12] ha colmato questo vuoto offrendo dataset multilingue con annotazioni token-level, suddivisioni standard in train/dev/test e un protocollo uniforme per riportare tali metriche.

2.3 Approcci basati su conoscenza esterna

Una delle prime categorie di approcci consiste nel verificare la veridicità delle dichiarazioni confrontandole con fonti esterne, come database o documenti trovati online. L'obiettivo è potenziare il modello generativo attraverso un canale di verifica indipendente: le informazioni generate vengono suddivise in unità verificabili, attingendo a prove trovate in fonti affidabili e giudicando in che misura tali prove supportano ciascuna affermazione.

Queste pipeline sono analoghe a un processo giornalistico: si individua una dichiarazione, si ricercano fonti credibili in grado di confermare o smentire tale dichiarazione e si comunica una conclusione. La principale differenza tra i vari approcci risiede nella granularità delle unità da verificare (triplette, frasi, domande) e nella tipologia di fonte consultata (grafi strutturati o pagine web non strutturate).

2.3.1 RefChecker

RefChecker [5] rappresenta le frasi generate sotto forma di triplette $(soggetto, predicato, oggetto)$, che sono estratte tramite un modulo di estrazione dell'informazione, e verifica la loro veridicità in

relazione a un contesto recuperato. Per ciascuna tripletta, viene condotta una ricerca mirata che genera frammenti testuali di evidenza; successivamente, un classificatore neurale determina se tali prove sostengono, contraddicono o sono neutrali rispetto all'affermazione. Il metodo contempla tre scenari (zero, noisy e accurate context) in funzione della qualità del contesto disponibile e, grazie a una decomposizione strutturata, consente di ottenere incrementi significativi in Macro-F1 rispetto a soluzioni a livello di frase che si basano su una singola verifica globale.

Il processo computazionale si può ridurre alle seguenti tre fasi: (i) dividere la risposta dell'LLM in componenti atomiche che possano venire verificati separatamente, (ii) cercare evidenze testuali specializzate nelle singole triplette, (iii) attestare la relazione delle evidenze all'affermazione. La dichiarazione esplicita delle affermazioni consente al sistema anche di trattare anche le risposte ampie, dividendo la verificabilità in micro.

Esempio pratico. Se un modello genera la frase "La Torre Eiffel è stata costruita nel 1889 per i Giochi Olimpici di Berlino", RefChecker estrae la tripla *(Torre Eiffel, costruzione, Giochi Olimpici di Berlino)* e avvia una query specifica. I dati recuperati dal contesto turistico e storico non rivelano alcun legame con i Giochi Olimpici, di conseguenza il classificatore etichetta lo span corrispondente come "contraddice". Pertanto, il sistema fornisce un avviso specifico riguardante il frammento errato senza compromettere il resto della risposta, che potrebbe contenere informazioni accurate.

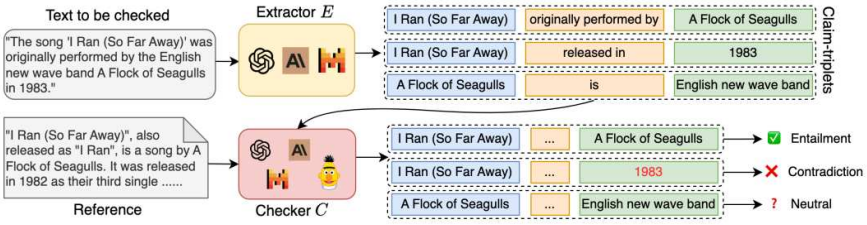

Figura 2.1: Schema semplificato del funzionamento di RefChecker, adattato da [5].

2.3.2 KnowHalu

KnowHalu [6] combina conoscenza strutturata (grafici di certezza e database) e non strutturata (fasi web recuperate) attraverso query orchestrate passo per passo gestite da un controller a controllo neurale. Ogni affermazione viene trasformata in domande atomiche, dove si raccolgono le prove e si calcolano le misure di sostegno. Copre anche le *non-fabrication hallucinations*, dove rileva risposte vaghe o evasive dove nessuna fonte affidabile verifica l'informazione; l'uso di fonti eterogenee aumenta la copertura oltre i metodi basati su un'unica fonte.

Il controllore funziona analogamente a un investigatore che impiega un approccio combinato di interrogazioni strutturate, come quelle su grafi tipo Wikidata, e ricerche testuali in modo graduale: qualora una fonte si riveli inadeguata, vengono predisposti ulteriori passaggi per la raccolta di evidenze. Ad ogni fase, il modello rivede un punteggio di supporto, che può essere interpretato come un semaforo: verde quando le evidenze sono concordi, rosso quando si manifestano contraddizioni e giallo quando l'informazione risulta indistinta.

Esempio pratico. Consideriamo la risposta "Il vaccino contro la malaria RTS,S è stato approvato per uso globale nel 2005". KnowHalu riformula l'affermazione in domande come "Quando è stato approvato il vaccino RTS,S?" e "Da quale organizzazione?". Una query su Wikidata fornisce il 2021 come data di raccomandazione OMS, mentre la ricerca web conferma l'approvazione nel 2021 per alcuni paesi africani. L'assenza di evidenze a supporto del 2005 fa crollare lo score e lo span contenente l'anno errato viene marcato come allucinato.

2.4 Approcci guidati dall'incertezza

Questi metodi stimano la probabilità che un passaggio sia allucinato analizzando l'incertezza del modello stesso. L'assunto è che gli LLM manifestino segnali interni quando "inventano" contenuti: valutare la distribuzione delle probabilità predette o la stabilità delle rappresentazioni consente di attribuire punteggi di affidabilità senza ricorrere a fonti esterne.

Rispetto agli approcci basati sulla conoscenza, l'attenzione si sposta sul "sentiment" del LLM interno: quanto ci si sente sicuri nel generare un token? Tali algoritmi codificano questa domanda come punteggi calcolabili utilizzando la manipolazione logit o creando più varianti della stessa risposta.

2.4.1 Claim-Conditioned Probability

Fadeeva et al. propongono la **Claim-Conditioned Probability** (CCP) [10], facendo uso delle distribuzioni interne dei grandi modelli linguistici (LLM) per stimare l'incertezza relativa a un'affermazione fornita. L'idea centrale è confrontare la probabilità dell'affermazione data con quella delle varianti controfattuali generate mascherando token importanti e rigenerando condizionalmente. Un valore basso suggerisce che il modello stesso ritiene fragile l'affermazione generata. Sebbene l'approccio sia classificato come *white-box*, non vengono utilizzate fonti esterne, come evidenziato dalla competitività con i metodi di verifica dei fatti basati sul recupero, l'ulteriore vantaggio è la funzionalità in tempo reale su sistemi già implementati.

In sostanza, il metodo elabora un esperimento mentale di dimensioni contenute: cela uno o più token della risposta (come ad esempio i nomi propri) e richiede al modello di rigenerarli in più occasioni. Nel caso in cui il LLM attribuisca probabilità simili alle varianti alternative, si può concludere che la scelta iniziale era arbitraria; viceversa, se l'output originale si mantiene come il più probabile, si considera lo span come affidabile. La CCP traduce tale intuizione in un punteggio numerico continuo facilmente soggettabile a soglie.

Esempio pratico. Supponiamo che il LLM dica "La capitale dell'Australia è Sydney". CCP nasconde il token "Sydney" e recupera il termine eliminato. Se tra le opzioni risultanti ci sono solitamente "Canberra" o altre città con probabilità simile, il punteggio dell'affermazione diminuisce e il token viene indicato come sospetto. Su frasi corrette, come "La capitale del Giappone è Tokyo", altre rigenerazioni ricevono probabilità molto inferiori e il sistema non emette falsi allarmi.

2.4.2 LLM Ensemble

Arteaga et al. [11] combinano **BatchEnsemble** e **LoRA** per stimare congiuntamente entropie epistemiche e predittive. Vengono generate più copie in versione leggera del modello, tutte addestrate dalla stessa dorsale, ma con adattatori diversi, e viene esplorata la variazione nelle probabilità emesse dai membri dell'insieme. Una grande dispersione indica che l'LLM ha una scarsa fiducia nelle risposte generate; un calibratore finale converte questo segnale in tag di output a livello di token. L'output è una stima precisa dell'incertezza (precisione del 97,8% su *faithfulness hallucinations*) che ha un costo computazionalmente inferiore dovuto al riutilizzo della stessa base parametrica.

Le componenti fondamentali comprendono: (i) un backbone comune, (ii) adattatori leggeri LoRA che generano variazioni tra i membri, (iii) una testa di calibrazione che converte la varianza dei logit in probabilità di allucinazione. A differenza di un ensemble tradizionale, questa strategia richiede un numero limitato di milioni di parametri supplementari e può essere incorporata in pipeline già esistenti senza la necessità di gestire modelli completi distinti.

Esempio pratico. Si immagini di chiedere a un assistente legale "In quale anno è stata approvata la GDPR?". Se tutte le teste dell'ensemble convergono su "2016" con alta confidenza, il token riceve un punteggio di affidabilità elevato. Se invece alcune teste rispondono "2018" o "2015", l'entropia condivisa aumenta e il calibratore segnala lo span come dubbio, invitando l'utente a verificare la data (correttamente il regolamento è entrato in vigore nel 2018 dopo l'adozione nel 2016).

2.5 Auto-coerenza e verifiche multiple

Un'altra linea di ricerca valuta la coerenza interna del modello generando o analizzando risposte multiple. Se un LLM fornisce soluzioni diverse o contraddittorie quando sollecitato più volte sullo stesso input, è probabile che stia "indovinando" senza basi solide; questi metodi interrogano quindi il modello come un revisore che cerca conferme incrociate.

La metafora consiste nel rivolgere la medesima interrogazione a più esperti autonomi: qualora le opinioni siano concordi, possiamo considerarci affidabili, mentre in caso contrario è opportuno effettuare un'analisi più approfondita. Gli algoritmi descritti in questa sezione attuano automaticamente tale confronto per ciascun token o frase prodotta.

2.5.1 SelfCheckGPT

SelfCheckGPT [7] produce diverse risposte stocastiche alla stessa domanda attraverso l'uso di temperature più elevate o campionamento del nucleo per azzardare varianti plausibili, oltre a giudicare la loro somiglianza semantica attraverso l'aiuto di metriche come i modelli BERTScore o NLI. Grandi disparità indicano la possibile presenza di passaggi allucinatori, che vengono poi emessi come inaffidabili. Questa pratica è molto utile nello scenario della scatola nera, poiché non richiede l'accesso ai log interni e si limita a organizzare richieste ripetute al LLM.

Le operazioni si suddividono nelle seguenti fasi: (i) produrre k risposte alternative preservando il prompt originale, (ii) valutare la somiglianza di ogni frase in relazione alla risposta principale, (iii) combinare le differenze in un punteggio di affidabilità a livello di token. Qualora le risposte selezionate siano concordi, il sistema manifesta fiducia; viceversa, se vi è disaccordo, quest'ultimo viene attribuito ai token che hanno causato la variazione.

Esempio pratico. Alla domanda "Qual è la capitale dell'Australia?" l'LLM principale potrebbe rispondere "Sydney". Generando ulteriori varianti si ottengono frasi con "Canberra" o "Melbourne": la similarità semantica cala proprio sulla parola della città, che viene etichettata come poco affidabile. Su un quesito più certo, come "Chi è l'autore di
textit1984?" tutte le risposte convergono su "George Orwell" e SelfCheckGPT assegna un punteggio di fiducia elevato all'intera frase.

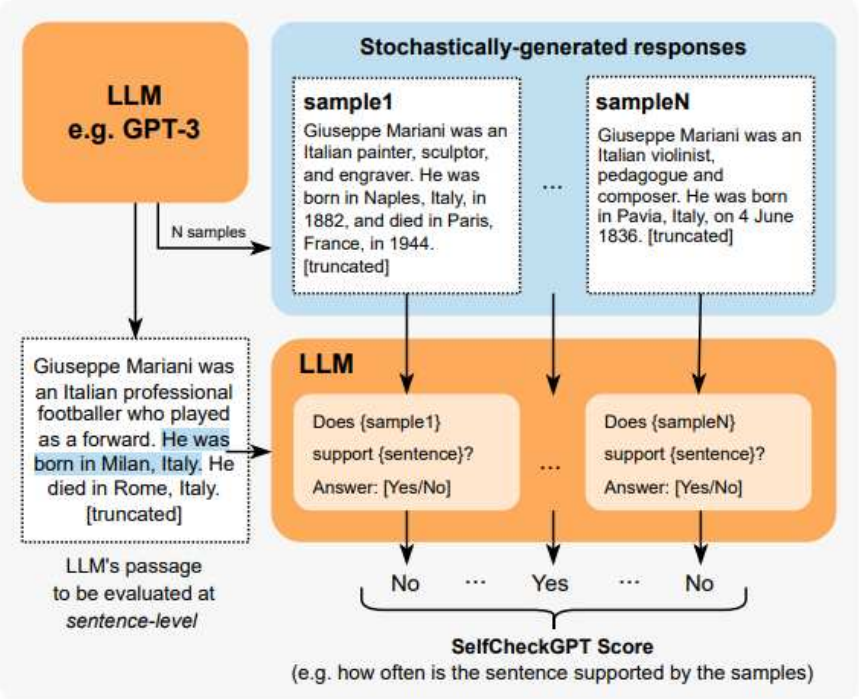

Figura 2.2: Pipeline di SelfCheckGPT basata su risposte multiple, adattata da [7].

2.5.2 SelfElicit

SelfElicit [8] è una tecnica *white-box* che utilizza mappe di attenzione per indicare le parti più importanti del contesto e provocare una rigenerazione controllata. In una fase di "auto-spiegazione" il modello individua i token su cui ha deciso la risposta e li usa come ancore per una seconda generazione controllata; le campate che ottengono supporto sono contrassegnate come potenziali allucinazioni. L'uso di marcatori espliciti dà la possibilità di far crescere la *Exact Match* senza spendere costi di formazione e poco sovraccarico durante l'inferenza.

Il processo alterna due modalità: prima l'LLM produce la risposta annotando i token del contesto con pesi di attenzione; poi un modulo di verifica rigenera ciascun pezzo di risposta forzando il modello a concentrarsi solo sui token altamente rilevanti. Se la rigenerazione fallisce o introduce varianti sostanziali, il sistema deduce che il passaggio originale non era ben supportato dal contesto.

Esempio pratico. In un compito di question answering su un paragrafo biografico, la risposta "Marie Curie ha scoperto il radio nel 1895" viene giustificata dalle frasi del contesto che parlano del 1898. Durante la rigenerazione condizionata ai token "1898" e "radio", SelfElicit nota lo scostamento temporale e segnala il numero errato (1895) come potenziale hallucination, lasciando intatta la parte corretta della frase.

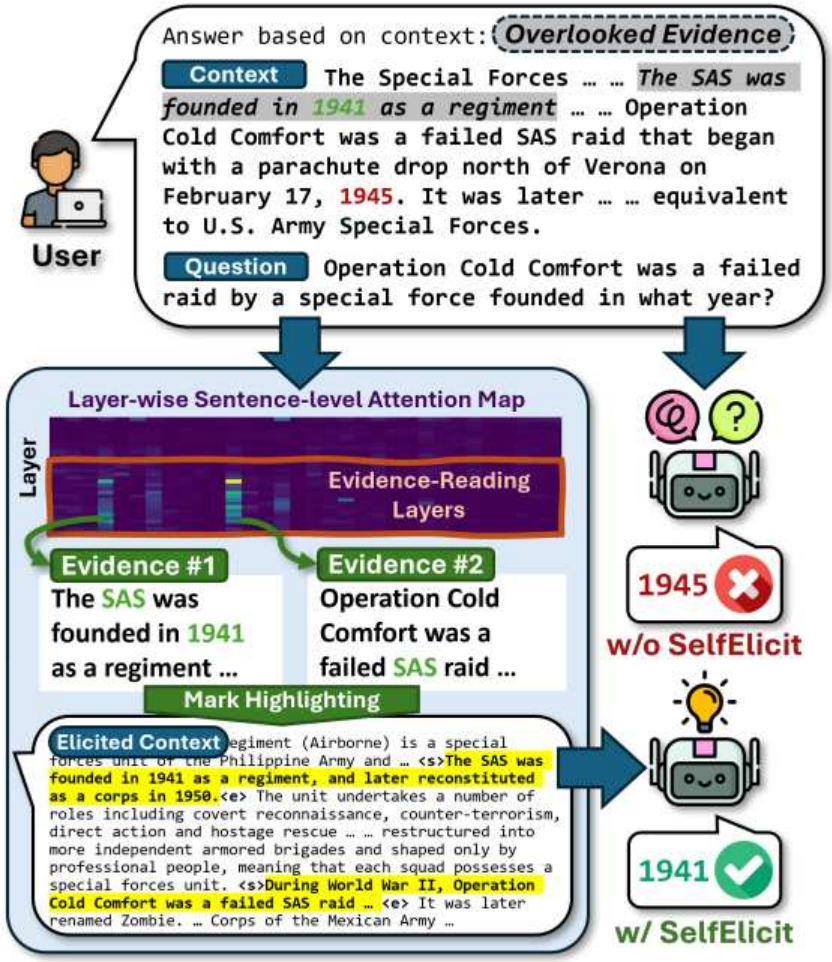

Figura 2.3: Esempio di marcatura delle frasi chiave in SelfElicit, adattato da [8].

2.5.3 Altre tecniche

SelfCheck [4] verifica l'accuratezza del ragionamento matematico ricreando passo dopo passo le conclusioni attraverso altre catene di ragionamento e confrontandole con la logica generata dal modello: una mancata corrispondenza tra le due tracce suggerisce un possibile errore. È concepito come appropriato per situazioni di "chain-of-thought" e come un controllore indipendente che ricalcola da solo.

Esempio pratico. Davanti alla domanda "Calcola 37×24", l'LLM risponde 888 descrivendo i passaggi moltiplicativi. SelfCheck rigenera la catena di pensiero ottenendo 888 e 888 in più iterazioni: l'accordo mantiene alto il punteggio di affidabilità. Se invece una catena alternativa produce 924, la divergenza fa scendere il punteggio proprio sugli step coinvolti.

InterrogateLLM [9], invece, ricostruisce la query originaria secondo l'output creato e giudica il suo carattere di coerenza con quest'ultima attraverso modelli delle similarità semantiche, delineando frasi che si discostano lo scopo dell'utente anche se conservando un tono verosimile.

Questo tipo di trattamento si rivela utilissimo negli chatbot: se la risposta deriva implicitamente da una domanda diversa, l'utente viene avvisato.

Esempio pratico. Un utente chiede "Suggeriscimi ricette vegetariane rapide" e il modello propone "pollo alla griglia". InterrogateLLM sintetizza la query implicita "Come cucinare il pollo alla griglia?" dalla risposta e la confronta con la richiesta originale: la bassa similarità semantica evidenzia l'allucinazione pragmatica e permette di bloccare il consiglio fuori tema.

2.6 Segnali interni e analisi spettrale

Alcuni lavori si focalizzano sullo studio delle rappresentazioni interne del modello alfine di scoprire schematismi anomali prima che si riflettano nel output testuale. Questo tipo di ricerca fa leva sulla struttura delle attention maps o dei logit alfine di discriminare segnali statistici rare che sono associate alla generazione di contenuti infondati.

In termini intuitivi, si possono notare i "campanelli d'allarme" che emergono dalle attivazioni interne del LLM. Qualora la dinamica delle attenzioni o dei vettori latenti subisca un'improvvisa variazione rispetto al comportamento consueto, il metodo contrassegna il segmento come potenzialmente pericoloso.

2.6.1 LapEigvals

LapEigvals [13] interpreta le mappe delle attenzioni come reti, nelle quali i token vengono rappresentati come nodi e i pesi delle attenzioni operano come archi, e computa gli autovalori del Laplaciano al scopo di scoprire comportamenti anomali nello spettro. La distribuzione degli autovalori viene poi utilizzata da un classificatore leggero che stabilisce se c'è output affidabile. Questa feature evidenzia una grande resistenza al rumore, consente una esecuzione con accesso limitato *white-box* (solo le attenzioni vanno bene) e prevale altre statistiche delle attenzioni su cinque dei.

Intuitivamente, se il modello segue il contesto in modo affidabile, il grafico dell'attenzione è ben organizzato: i token informativi mirano a fonti coerenti e gli autovalori mostrano uno schema strutturato. Quando il LLM "vaga", l'attenzione si dilaga su token non informativi che generano pattern spettrali anomali, che il classificatore impara a riconoscere.

Esempio pratico. Nel caso in cui il contesto delinei le fasi della missione Apollo 11, ma il modello includa un particolare fittizio riguardante "un atterraggio nel Mare dei Sogni", le mappe di attenzione indicano che i token pertinenti non sono avvalorati dai fattori storici. Il vettore degli autovalori risultante si discosta da quello registrato in risposte corrette, e il sistema etichetta il segmento "Mare dei Sogni" come allucinato.

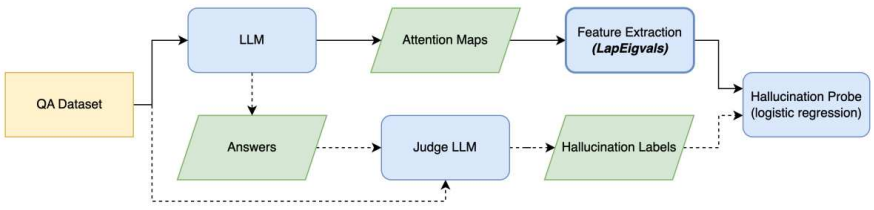

Figura 2.4: Flusso di calcolo degli autovalori nel metodo LapEigvals, adattato da [13].

2.7 Sintesi comparativa

La Tabella 2.1 presenta le caratteristiche più importanti dei metodi analizzati, evidenziando il tipo di accesso al modello, la fonte informativa utilizzata e la granularità dell'analisi.

Metodo	Accesso	Fonte	Granularità
SelfCheckGPT	Black-box	Auto-coerenza	Frase/Passaggio
SelfElicit	White-box	Attention maps	Frase
RefChecker	Hybrid	Knowledge	Triplette
KnowHalu	Hybrid	Multi-form knowledge	Frase
LapEigvals	White-box	Segnali interni	Token
CCP	White-box	Incertezza	Token
Approccio MALTO	Hybrid	Probabilità + NLI	Token

Tabella 2.1: Confronto sintetico dei principali metodi di hallucination detection.

2.8 Sfide aperte

Nonostante i risultati raggiunti, rimangono ancora molti interrogativi di ricerca:

- **Generalizzazione cross-dominio e cross-lingua**: la maggior parte dei lavori in letteratura è limitata a domini specifici e presenta sfide nella trasferibilità cross-linguistica.
- **Costo computazionale**: potrebbe essere costoso utilizzare modelli NLI o sofisticati sistemi di ensemble in applicazioni in tempo reale.
- **Ambiguità semantica**: non è sempre facile distinguere le risposte neutre da quelle contraddittorie, con conseguenze sulla sensibilità dei sistemi.
- **Equilibrio con la fluenza**: tecniche che migliorano la leggibilità non garantiscono automaticamente maggiore fattualità.

Le seguenti sfide esercitano anche un impatto stimolante sull'indagine sui pipeline ibridi e sulla comparsa degli atti diversivi di valutazione.

Capitolo 3

Metodologia

Il presente capitolo presenta un dettagliato descrizione della pipeline ideata come supporto all'identificazione e riduzione delle allucinazioni a livello di token nella shared task **Mu-SHROOM**. Basandosi sull'approccio **MALTO**, viene evidenziato come la combinazione dei segnali probabilistici e controlli semantici possa venire utilizzata al termine dei calcoli per determinare una misura di punteggio di affidabilità su base individuale ad ogni parola. La pipeline viene specificamente delineata come priva delle fasi specifiche di addestramento e validazione, lavorando esclusivamente nella modalità di inferenza. Tre varianti autonome previste al termine come correzioni dinamiche dei token sospetti vengono poi seguite dalla analisi dei maggiori aspetti implementativi.

3.1 Approccio MALTO di riferimento

La pipeline originale, come riportata in Figura 3.1, integra due moduli complementari:

1. **Modello di riferimento**: Utilizziamo un LLM più grande del generatore per produrre distribuzioni di probabilità sui token successivi, quindi calcoliamo in che misura ciascuna parola generata è considerata "prevista" dal modello.

2. **Verifica semantica**: un classificatore di *Natural Language Inference* (NLI) determina se una proposta alternativa è supportata, neutrale o negata dal contesto.

La combinazione di questi segnali crea uno *hallucination score* che misura il grado di affidabilità per ogni parola nella sequenza.

3.1.1 Architettura generale

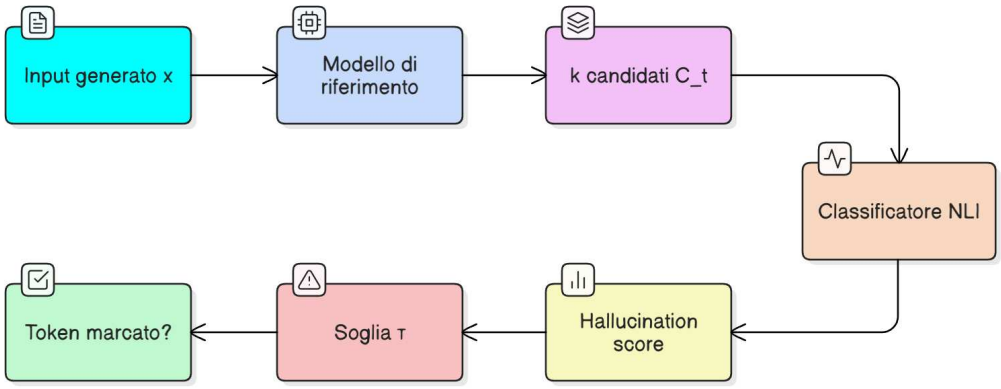

Figura 3.1: Workflow dell'approccio MALTO di riferimento.

Dato un output $x = (x_1, \ldots, x_T)$ generato da un modello m, ogni posizione t viene esaminata interrogando un modello di riferimento M riguardo al prefisso $x_{<t}$. La distribuzione di probabilità $P_M(\cdot \mid x_{<t})$ è impiegata per selezionare un insieme di candidati *top-K* fino a quando la massa cumulativa non supera una soglia k o la probabilità di un token scende al di sotto ρ. Le probabilità dei token che formano la stessa parola vengono sommate per calcolare $p(w)$.

3.1.2 Verifica semantica via NLI

Ogni candidato w viene elaborato con il contesto utilizzando un classificatore di inferenza del linguaggio naturale multilingue (`MoritzLaurer/mDeBERTa-v3-base-mnli-xnli`, ottimizzato su `MNLI` e `XNLI`), che produce le probabilità di *entailment* $P_+(w)$, *neutral* $P_=(w)$ e *contradiction* $P_-(w)$. Per mantenere ragionevole il costo computazionalmente elevato, la frase viene troncata attorno al token candidato in esame, mantenendo una finestra locale sufficientemente ampia in modo da preservare il significato.

Processo di verifica NLI dettagliato. Riprendendo l'esempio precedente, supponiamo che il modello m abbia effettivamente generato *"Andrea"* nella frase *"Il vincitore del premio Nobel per la fisica nel 2020 è stato Andrea Ghez"*. Il classificatore NLI multilingue confronta questa *premessa* con le *ipotesi* ottenute sostituendo *"Andrea"* con ciascuna alternativa:

- **Premessa**: *"Il vincitore del premio Nobel per la fisica nel 2020 è stato Andrea Ghez"*
- **Ipotesi 1**: *"Il vincitore del premio Nobel per la fisica nel 2020 è stato Roger Penrose"* $\Rightarrow P_+ = 0.85$, $P_= = 0.12$, $P_- = 0.03$ (alta probabilità di entailment)
- **Ipotesi 2**: *"Il vincitore del premio Nobel per la fisica nel 2020 è stato Einstein"* $\Rightarrow P_+ = 0.02$, $P_= = 0.18$, $P_- = 0.80$ (alta contraddizione temporale)
- **Ipotesi 3**: *"Il vincitore del premio Nobel per la fisica nel 2020 è stato Marie Curie"* $\Rightarrow P_+ = 0.05$, $P_= = 0.25$, $P_- = 0.70$ (contraddizione storica)

La finestra contestuale è circoscritta alla frase attuale al fine di prevenire la diluizione del segnale NLI in testi estesi, ottimizzando così l'accuratezza semantica e l'efficienza computazionale

3.1.3 Hallucination score e soglia dinamica

Il punteggio attribuito a una parola w è conforme all'approccio della *Claim-Conditioned Probability* proposto da Fadeeva et al. [10]:

$$HS(w) = 1 - \frac{\sum_{i=1}^{N} p_i(w)\, p_i^{\text{nli}}(w)}{\sum_{i=1}^{N} p_i(w)}, \qquad p_i^{\text{nli}}(w) = P_+(w) + P_=(w).$$

Esempio pratico. Consideriamo la frase generata *"Roma è la capitale della Francia"* e analizziamo il token *"Francia"*. Il modello di riferimento Mistral-7B, dato il contesto *"Roma è la capitale della"*, potrebbe generare come alternative più probabili: *"Italia"* ($p_1 = 0.85$), *"Francia"* ($p_2 = 0.08$), *"Spagna"* ($p_3 = 0.04$). Il classificatore NLI multilingue confronta poi la frase originale con ciascuna alternativa:

- *"Roma è la capitale dell'Italia"*: $P_+ = 0.92$, $P_= = 0.06$, $P_- = 0.02 \Rightarrow p_1^{\text{nli}} = 0.98$
- *"Roma è la capitale della Francia"*: $P_+ = 0.02$, $P_= = 0.13$, $P_- = 0.85 \Rightarrow p_2^{\text{nli}} = 0.15$
- *"Roma è la capitale della Spagna"*: $P_+ = 0.05$, $P_= = 0.25$, $P_- = 0.70 \Rightarrow p_3^{\text{nli}} = 0.30$

L'hallucination score risulta: $HS(\text{“Francia”}) = 1 - \frac{0.85\times0.98+0.08\times0.15+0.04\times0.30}{0.85+0.08+0.04} = 1 - \frac{0.857}{0.97} = 0.12$, indicando una bassa probabilità di allucinazione nonostante l'errore fattuale evidente.

I punti accumulati fino al posto t vengono utilizzati al fine di stabilire una soglia adattiva

$$\tau_t = \max\big(0.4, P_{80}(HS_{1:t})\big),$$

dove P_{80} indica l'ottantesimo percentile. Se $HS(w) > \tau_t$, il token viene marcato come potenzialmente allucinato e valutato per la sostituzione.

3.2 Claim-Conditioned Probability

La **Claim-Conditioned Probability** (CCP) distingue il componente dell'incertezza relativo al contenuto della dichiarazione, ignorando la variabilità sintattica. Per un token x_j la formulazione è:

$$\text{CCP}(x_j) = \mathbb{P}\big(\text{Meaning}(x_{1:j}) \mid x_{<j}, \text{ClaimType}(x_{1:j})\big).$$

Nel presente contesto, ciascuna posizione è esaminata attraverso varianti x_j^k, le quali vengono generate sostituendo x_j con i token che risultano più probabili in base al modello di riferimento M (Mistral-7B). Il rapporto tra le probabilità delle varianti supportate e quelle che non vengono contraddette definisce la componente di affidabilità associata al token attuale:

$$\text{CCP}_{\text{word}}(x_j) = \frac{\sum_{\text{NLI}(x_j^k, x_j)=\text{e}} P(x_j^k \mid x_{<j})}{\sum_{\text{NLI}(x_j^k, x_j)\in\{\text{e,c}\}} P(x_j^k \mid x_{<j})}.$$

Esempio applicativo. Consideriamo la frase *"Einstein nacque nel 1985"* e il token critico *"1985"*. Dato il contesto *"Einstein nacque nel"*, Mistral-7B genera alternative come *"1879"* ($P = 0.72$), *"1985"* ($P = 0.15$), *"1880"* ($P = 0.08$). Il classificatore NLI valuta:

- *"Einstein nacque nel 1879"* vs. originale: entailment ($P_+ = 0.89$) $\Rightarrow$ supportata
- *"Einstein nacque nel 1985"* vs. originale: neutral ($P_= = 1.0$) $\Rightarrow$ non contraddetta
- *"Einstein nacque nel 1880"* vs. originale: contradiction ($P_- = 0.78$) $\Rightarrow$ contraddetta

Risulta $\mathrm{CCP}_{\mathrm{word}}(\text{“1985”}) = \frac{0.72}{0.72+0.15} = 0.83$, indicando alta affidabilità nonostante l'errore fattuale, poiché il modello NLI non rileva la contraddizione storica.

Aggregando tutti i token di un claim C si ottiene:

$$\mathrm{CCP}_{\mathrm{claim}}(C) = 1 - \prod_{x_j \in C} \mathrm{CCP}_{\mathrm{word}}(x_j).$$

Questa formulazione costituisce la base teorica dell'hallucination score impiegato nella pipeline.

3.3 Controlli di qualità linguistica

Affinché le sostituzioni siano grammaticalmente e semanticamente ragionevoli, la pipeline contiene anche due meccanismi di controllo complementari che possono essere invocati selettivamente sulle numerose varianti.

3.3.1 Part-of-Speech (POS) Tagging

Il **Part-of-Speech tagging** rappresenta una metodologia di analisi morfologica che attribuisce a ciascun vocabolo la sua corrispondente classe grammaticale (sostantivo, verbo, aggettivo, ecc.). Nell'ambito della rilevazione delle allucinazioni, il monitoraggio POS assicura che le sostituzioni siano conformi alla struttura sintattica della frase originaria.

Implementazione. La pipeline utilizza spaCy per l'analisi morfologica:

```python
import spacy

nlp = spacy.load("en_core_web_sm")

def get_pos_tag(word):
    """Estrae la categoria POS di una parola usando spaCy."""
    doc = nlp(word)
    return doc[0].pos_ if doc else None

# Esempio di controllo durante la sostituzione
original_pos = get_pos_tag("running")     # VERB
candidate_pos = get_pos_tag("beautiful")   # ADJ

if original_pos != candidate_pos:
    # Rifiuta la sostituzione per incoerenza grammaticale
    continue
```

Listing 3.1: Implementazione del controllo POS.

Motivazione. Prendiamo il seguente frame della frase: *"L'atleta **correva** molto veloce"*. Se il sistema interpreta "correva" come allucinatorio, il controllo POS assicura che la sostituzione preservi la funzione verbale, accettando esempi simili.

Categorie POS principali:

- **NOUN**: sostantivi (cat, house, information)
- **VERB**: verbi (run, think, analyze)
- **ADJ**: aggettivi (blue, large, intelligent)
- **ADV**: avverbi (quickly, very, often)
- **NUM**: numerali (five, 2020, first)
- **PROPN**: nomi propri (London, Einstein, Microsoft)

Limitazioni. Il tracking delle componenti delle parti del discorso può finire molto presto sotto severe limitazioni nelle situazioni nelle quali maggiormente ci si aspetterebbe una libertà semantica sulla correttezza astrattamente sintattica. Per tal ragione, alcuni esperimenti configurazionali (denominati *No POS*) disabilitano questo controllo per permettere sostituzioni più aggressive.

3.4 Varianti della pipeline

Contrariamente alla baseline, tutte le varianti proposte sono in grado di rieseguire la pipeline in più fasi. Dopo ciascuna iterazione, i token con $HS(w) > \tau_t$ vengono sostituiti e la sequenza risultante è nuovamente analizzata, consentendo una progressiva raffinazione del testo. Le quattro strategie esaminate sono rappresentate nelle Figure 3.2–3.4. Inoltre, è stata implementata una configurazione sperimentale denominata **No POS** che disabilita il controllo morfologico al fine di valutare l'influenza della coerenza grammaticale sulle prestazioni.

3.4.1 No Fluency

La prima variante utilizza esclusivamente il punteggio probabilistico e la verifica NLI. Quindi, se un token supera la soglia, esso viene sostituito dalla prima opzione che fa parte della stessa categoria morfologica (rinvio a §3.3.1 al riguardo, maggiori informazioni sul controllo del POS). Non vengono imposti vincoli sulla scorrevolezza complessiva.

3.4.2 No POS (Configurazione sperimentale)

Per studiare l'effetto del controllo grammaticale, è stata testata la seguente configurazione che disabilita completamente il filtro POS completo (`use_pos=False`). In questa versione i simboli riconosciuti come allucinatori vengono sostituiti con i primi sostituti che compaiono nel lessico delle parole, ignorando la categoria morfologica. Sebbene questa impostazione consenta sostituzioni più audaci, potrebbe compromettere la correttezza sintattica del testo di output.

```
# Variante No Fluency: solo controllo POS
for alt in sorted(alt_data, key=lambda x: -x["combined_prob"]):
    candidate = alt["full_word"].strip()
    if not is_valid_replacement(candidate):
        continue

    # Controllo POS (Part-of-Speech) per mantenere coerenza grammaticale
    candidate_pos = get_pos_tag(candidate) if use_pos else None
    if use_pos and original_pos and candidate_pos != original_pos:
        continue

    # Accetta la prima alternativa valida senza controlli di fluency
    best_replacement = candidate
    chosen = True
    break

# Applica la sostituzione se trovata un'alternativa valida
if best_replacement != word and chosen:
    updated_words[i:i+1] = best_replacement.split()
    token_info["replaced"] = True
```

Listing 3.2: Implementazione della variante *No Fluency*.

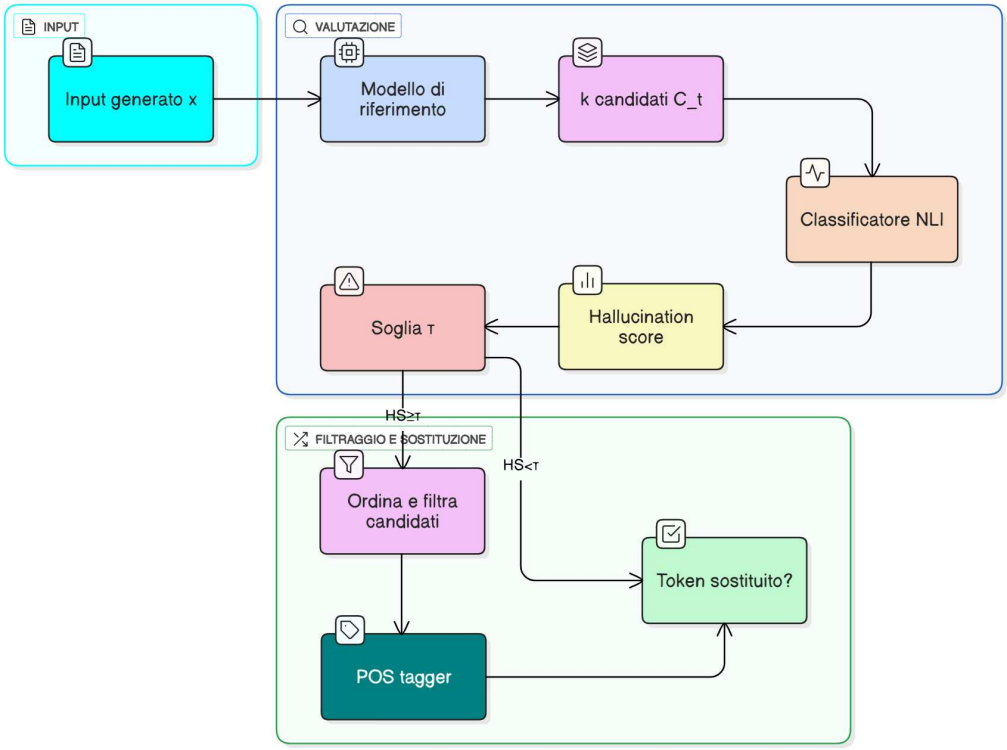

Figura 3.2: Workflow della variante *No Fluency*.

3.4.3 Fluency

La variante *Fluency* funziona con un controllo di scorrevolezza che usa un modello GPT-2 distillato. La sostituzione proposta riceve l'approvazione solo quando la nuova frase mostra una perplessità che non supera il 95% della perplessità della frase originale. In caso contrario, il token viene lasciato invariato.

```
# Variante Fluency: aggiunge controllo di perplexity
for alt in sorted(alt_data, key=lambda x: -x["combined_prob"]):
    candidate = alt["full_word"].strip()
    if not is_valid_replacement(candidate):
        continue

    # Controllo POS standard
    candidate_pos = get_pos_tag(candidate) if use_pos else None
    if use_pos and original_pos and candidate_pos != original_pos:
        continue

    # Test di fluency con GPT-2
    test_words = updated_words.copy()
    test_words[i:i+1] = candidate.split()
    test_text = " ".join(test_words)

    # Calcolo perplexity della frase modificata
    new_perplexity = scorer.score_sample(test_text, score_type="fluency")

```

```
    # Accetta solo se la perplexity non degrada troppo (soglia 95%)
    if new_perplexity <= original_perplexity * 0.95:
        best_replacement = candidate
        chosen = True
        break

if best_replacement != word and chosen:
    updated_words[i:i+1] = best_replacement.split()
    token_info["replaced"] = True
```

Listing 3.3: Implementazione della variante *Fluency*.

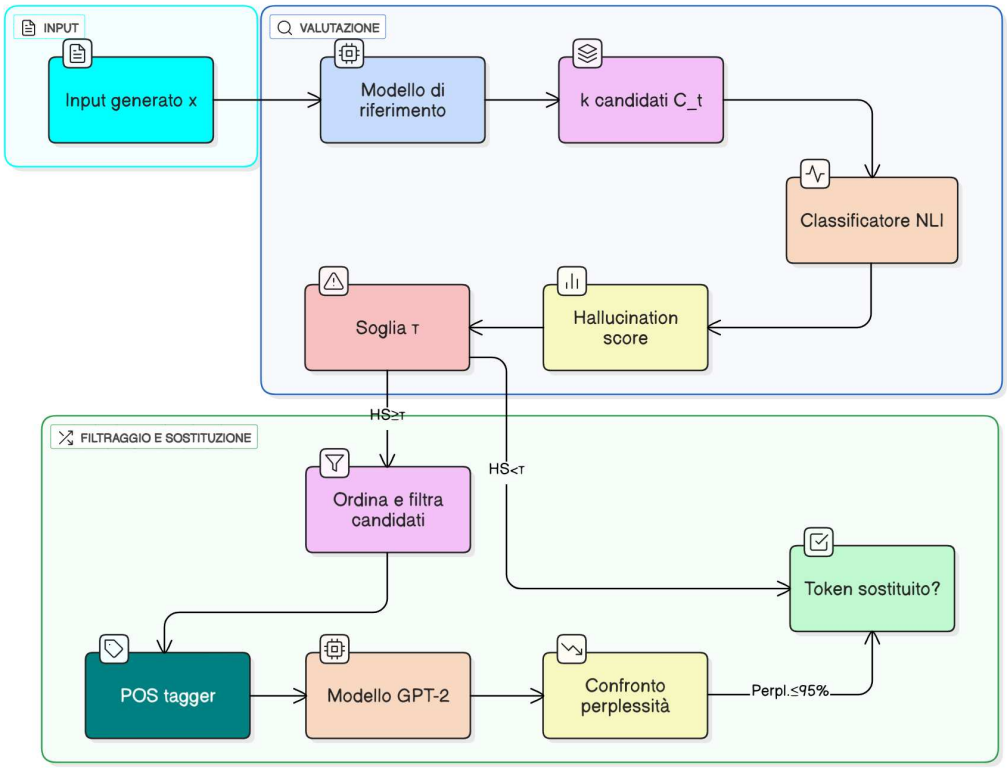

Figura 3.3: Workflow della variante *Fluency*.

3.4.4 Mask GPT-2

La variante *Mask GPT-2* utilizza un metodo ibrido che funziona attraverso due fasi distinte. Nel primo passaggio il sistema individua l'alternativa con il punteggio di contraddizione più alto secondo il modello di riferimento e poi applica GPT-2 per migliorare la fluidità del testo. Il sistema sceglie l'alternativa dal modello grande che produce la massima probabilità di contraddizione NLI per ogni token sospetto. Il sistema valuta tutte le sottostringhe della nuova alternativa che partono dal token intero fino ai prefissi più corti per trovare quella con la perplessità minima secondo GPT-2 mentre mantiene sia la coerenza del significato che la scorrevolezza del testo.

```
# Variante Mask GPT-2: ottimizzazione ibrida contraddizione + fluency
```

```
# Fase 1: trova l'alternativa con massima contraddizione
best_contradiction = max(alt_data, key=lambda x: x["contradiction"])
candidate_base = best_contradiction["full_word"].strip()

# Fase 2: ottimizza la fluency esplorando sottostringhe
best_perplexity = float('inf')
best_candidate = candidate_base

# Esplora tutte le sottostringhe possibili (dal più lungo al più corto)
for length in range(len(candidate_base.split()), 0, -1):
    substrings = [
        " ".join(candidate_base.split()[:length])
        for length in range(1, len(candidate_base.split()) + 1)
    ]

    for substring in substrings:
        # Test di fluency per ogni sottostringa
        test_words = updated_words.copy()
        test_words[i:i+1] = substring.split()
        test_text = " ".join(test_words)

        perplexity = scorer.score_sample(test_text, score_type="fluency")

        # Seleziona la sottostringa con minima perplexity
        if perplexity < best_perplexity:
            best_perplexity = perplexity
            best_candidate = substring

# Applica la migliore sostituzione trovata
if best_candidate != word:
    updated_words[i:i+1] = best_candidate.split()
    token_info["replaced"] = True
```

Listing 3.4: Implementazione della variante *Mask GPT-2*.

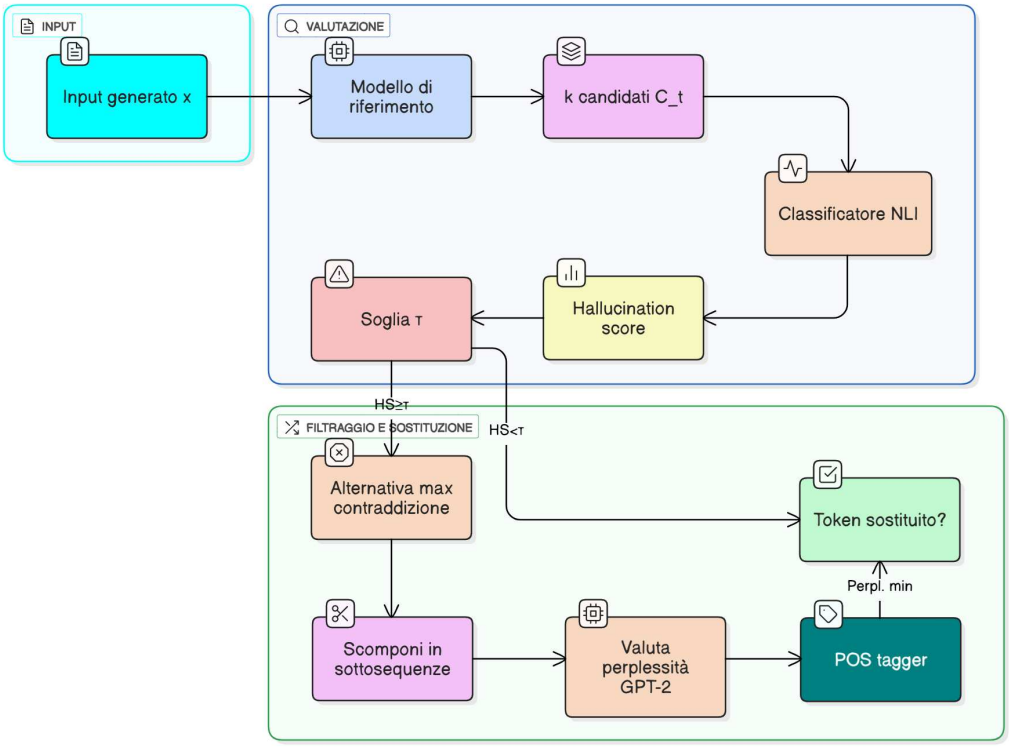

Figura 3.4: Workflow della variante *Mask GPT-2*.

3.4.5 Confronto tra le varianti

Strategia	POS	Sostituzione	Fluenza
Baseline	–	Detection only	–
No Fluency	✓	Token alternativo	–
No POS	×	Token alternativo	–
Fluency	✓	Token alternativo	✓
Mask GPT-2	✓	Alternativa ottim.	✓

Tabella 3.1: Caratteristiche principali delle varianti della pipeline.

3.5 Algoritmo generale

L'algoritmo di detection e mitigazione delle allucinazioni integra i componenti descritti nelle sezioni precedenti in una procedura unificata che opera sequenzialmente su ogni token della sequenza di input. Il sistema combina l'analisi statistica con la verifica semantica e i controlli di qualità per rilevare e gestire le allucinazioni potenziali.

3.5.1 Definizione delle variabili e parametri

Input dell'algoritmo:

- $x_{1:T} = (x_1, x_2, \ldots, x_T)$: l'algoritmo richiede la verifica delle sequenze di testo che genera rispetto a un modello di riferimento utilizzando un classificatore di inferenza del linguaggio naturale.
- M: modello di riferimento (Mistral-7B) per calcolare distribuzioni probabilistiche
- **Classificatore NLI**: `MoritzLaurer/mDeBERTa-v3-base-mnli-xnli` per valutare l'equivalenza semantica tra le opzioni testuali con supporto multilingue
- **Strategia**: variante scelta (No Fluency, Fluency, o Mask GPT-2)

Iperparametri di configurazione:

- K: numero massimo di candidati alternativi da considerare (tipicamente $K = 16$)
- ρ: soglia di probabilità minima per includere un candidato (es. $\rho = 0.95$)
- η: fattore di scaling per l'hallucination score (es. $\eta = 1.48$)

Variabili computate durante l'esecuzione:

- $\mathcal{C}_t$: insieme dei candidati top-K per la posizione t
- HS_t: hallucination score del token x_t calcolato secondo la formula CCP
- τ_t: soglia adattiva basata sull'80° percentile degli score precedenti
- $p^{\text{nli}}(w)$: punteggio semantico NLI per l'alternativa w, definito come $P_+(w) + P_=(w)$

3.5.2 Descrizione dell'algoritmo

Il processo di detection si articola in tre fasi principali per ogni token della sequenza:

Fase 1: Generazione delle alternative (linee 3-4). Il modello genera opzioni alternative per ciascun token attraverso la sua prima fase di rilevamento. Il modello di riferimento M genera una distribuzione di probabilità $P_M(\cdot|x_{<t})$ per ogni posizione del token t. Il sistema seleziona le alternative K più probabili che superano la soglia ρ per formare l'insieme $\mathcal{C}_t$.

Fase 2: Valutazione semantica e calcolo dell'hallucination score (linee 5-6). Ogni candidato $w \in \mathcal{C}_t$ passa attraverso il classificatore NLI per verificare il suo allineamento semantico con il contesto circostante. Il punteggio $p^{\text{nli}}(w) = P_+(w) + P_=(w)$ combina le probabilità di implicazione e neutralità escludendo la contraddizione. Il calcolo del punteggio dell'allucinazione HS_t segue la formula CCP che moltiplica le probabilità dei candidati per i corrispondenti punteggi semantici.

Fase 3: Decision e mitigazione (linee 7-12). Il sistema aggiorna la soglia adattiva τ_t selezionando il valore maggiore tra 0,4 e l'80° percentile dei precedenti punteggi $HS_{1:t}$. Il sistema rileva i token allucinati quando il prodotto di η e HS_t supera τ_t, quindi contrassegna tali token per un'ulteriore elaborazione in base al metodo scelto:

- **No Fluency**: Il sistema sostituisce le parole tramite sostituzione diretta con il candidato migliore che corrisponde alla parte del discorso richiesta.
- **Fluency**: verifica aggiuntiva che la perplessità GPT-2 non degradi oltre il 5%

- **Mask**: ottimizzazione ibrida che massimizza contraddizione e minimizza perplessità

Il sistema esegue più iterazioni che continuano finché non raggiunge la convergenza o completa tre passaggi. Ogni iterazione applica più sostituzioni per migliorare la qualità del testo generato.

Il flusso di lavoro completo è formalizzato nell'Algoritmo 1.

Algorithm 1 Rilevazione e mitigazione token-level

Require: sequenza $x_{1:T}$; modello di riferimento M; classificatore NLI; parametri (K, ρ, η); strategia (No Fluency/Fluency/Mask)

$HS \leftarrow [\,]$
for $t = 1$ **to** T **do**
  $\mathcal{C}_t \leftarrow$ Top-K da M su $x_{<t}$ con cutoff ρ
  $p^{\text{nli}}(w) \leftarrow P_+(w) + P_=(w)$
  $HS_t \leftarrow 1 - \dfrac{\sum_{w \in \mathcal{C}_t} p(w)\, p^{\text{nli}}(w)}{\sum_{w \in \mathcal{C}_t} p(w)}$
  $\tau_t \leftarrow \max\left(0.4, P_{80}(HS_{1:t})\right)$
  if $\eta \cdot HS_t > \tau_t$ **then**
    mark x_t come allucinato
    if strategia = Fluency **then** verifica perplessità
    end if
    if strategia = Mask **then** ottimizza via contraddizione+fluency
    end if
  end if
end for

3.6 Dettagli implementativi e complessità

La pipeline è stata realizzata in Python utilizzando la libreria `Transformers`. Il generatore e il modello di riferimento coincidono con `Mistral-7B` eseguito in modalità *no-gradient*; il classificatore NLI è `DeBERTa-v3-base` fine-tunato su `MNLI`; i controlli di fluenza e le rigenerazioni sono affidati a un GPT-2 distillato. Gli iperparametri principali (k, ρ, τ, η) derivano dalla configurazione dell'approccio MALTO e rimangono invariati durante tutte le prove senza alcuna fase di addestramento o validazione.

Dal punto di vista della complessità, indicando con T il numero di token, K il numero medio di candidati e W l'ampiezza della finestra NLI, il tempo richiesto da una singola passata è $\mathcal{O}(TK + TC_{\text{NLI}}(W))$. La memoria è dominata dai modelli impiegati; il metodo di esecuzione senza gradienti insieme al controllo del batching permette di utilizzare GPU che si trovano nella fascia media del mercato.

Per migliorare l'efficienza pratica sono adottati alcuni accorgimenti:

- caching delle distribuzioni del modello di riferimento per contesti ripetuti;
- batching delle finestre NLI omogenee;
- ricalcolo selettivo degli score solo nei pressi dei token sostituiti;
- terminazione anticipata se la variazione del percentile si stabilizza.

Durante lo sviluppo sono state testate e scartate alcune alternative, come il *nucleus sampling* al posto del top-K (che introduceva eccessiva varianza) e l'uso di soglie fisse per HS (troppo

sensibili alla lunghezza della sequenza). La soglia adattiva basata sui percentili si è dimostrata più robusta in tutti i contesti sperimentati.

Capitolo 4

Dataset

Il presente capitolo approfondisce i dataset annotati della shared task **Mu-SHROOM**, utilizzati nell'analisi delle pipeline menzionate nel Capitolo 3. Vengono descritti sia il dataset inglese principale sia i dataset multilingui utilizzati per test preliminari di trasferibilità. L'accento è interamente posto sui dati: la descrizione del setup sperimentale è spostata al Capitolo 5, dove viene presentata al fianco dei risultati quantitativi e qualitativi.

4.1 Il dataset Mu-SHROOM

Il dataset principale `mushroom.en-val.v2.jsonl`, fornito dalla shared task **Mu-SHROOM** come file di validation, viene utilizzato in questo lavoro per gli esperimenti principali in inferenza e valutazione. Inoltre, vengono utilizzati i dataset multilingui `mushroom.{it,fr,zh}-val.v2.jsonl` per test preliminari di trasferibilità. Ogni file contiene un prompt, la risposta generata da un LLM e le annotazioni token-level che indicano le porzioni allucinate. I dataset sono volutamente compatti: **50 esempi per lingua** selezionati per coprire una varietà di fenomeni, dalle citazioni errate ai numeri inventati. Non sono previsti set di addestramento o di validazione separati.

```
1 {"id":"val-en-1","lang":"EN","model_input":"What did Petra van Staveren win a gold
      medal for?","model_output_text":"Petra van Staveren won a silver medal in the
      2008 Summer Olympics in Beijing, China.","soft_labels":[{"start":10,"prob"
     :0.2,"end":12},...],"hard_labels":[[25,31],[45,49],[69,83]]}
```

Listing 4.1: Estratto dal file `mushroom.en-val.v2.jsonl`.

4.2 Struttura dei dati

Ciascun oggetto JSON comprende i campi principali elencati di seguito:

- `id`: identificativo univoco dell'esempio;
- `model_input`: prompt originale passato al modello generatore;
- `model_output_text`: risposta generata dal modello;
- `soft_labels`: intervalli con probabilità di allucinazione in [0,1];
- `hard_labels`: coppie di offset $(start, end)$ che definiscono gli span allucinati.

Questo schema rende possibile misurare sia la presenza di errori binari sia l'intensità percepita dell'allucinazione attraverso punteggi graduati.

Nel contesto della competizione **Mu-SHROOM**, questa distinzione coincide con la classificazione **soft vs. hard hallucination** introdotta dagli organizzatori [12]. Le etichette soft identificano porzioni potenzialmente dubbie ma non necessariamente errate, mentre quelle hard marcano gli span giudicati inequivocabilmente allucinati. La pipeline analizzata in questa tesi sfrutta entrambe le annotazioni: le prime per calibrare soglie e priorità d'intervento, le seconde per valutare in modo netto la correttezza delle correzioni.

4.3 Processo di annotazione

Le etichette provengono da annotatori umani che hanno seguito linee guida condivise con la comunità di Mu-SHROOM. Gli annotatori hanno identificato gli span allucinati confrontando ciascuna risposta con fonti affidabili e marcando sia gli errori grossolani sia quelli più sottili. Ogni esempio è stato controllato da almeno due annotatori; le discrepanze sono state risolte tramite discussione, garantendo un buon livello di accordo inter-annotatore. L'inclusione di punteggi "soft" permette di catturare incertezze o sfumature semantiche che non emergerebbero con una marcatura puramente binaria.

4.4 Statistiche e distribuzione

Dal punto di vista quantitativo, la Tabella 4.1 presenta alcune delle misure riassuntive calcolate su questo insieme di testi. Gli esempi contano in media una quarantina di parole e tutti contengono un minimo di un'allucinazione.

Misura	Valore
Numero di esempi	50
Output medio (parole)	39.9
Lunghezza minima–massima	1–249 parole
Span allucinati medi	2.62
Lunghezza media span (caratteri)	20.3
Esempi con almeno un'annotazione	48/50

Tabella 4.1: Statistiche principali del dataset Mu-SHROOM.

Questi dati testimoniano la presenza diffusa, anche se concentrata, di allucinazioni: in media si registrano più di due span per risposta, spesso di breve durata ma semanticamente significative.

4.5 Pre-processing e allineamento

Prima dell'analisi è necessario riconciliare le annotazioni con la tokenizzazione adottata dalla pipeline. I passaggi principali sono:

1. **Normalizzazione** del testo per preservare la corrispondenza tra caratteri e offset;
2. **Tokenizzazione** con il tokenizer di `Mistral-7B` senza simboli speciali aggiunti;

3. **Proiezione delle etichette hard**: uno span viene assegnato al token se la sovrapposizione supera il 50%;
4. **Aggregazione delle etichette soft**: i punteggi carattere-level sono mediati sui token corrispondenti.

Quando gli intervalli annotati non coincidono con i confini dei token subword si applica la regola della massima copertura per evitare falsi negativi.

4.6 Limiti e considerazioni etiche

La limitata grandezza dei dataset (50 esempi per lingua) comporta una suscettibilità delle statistiche agli outlier e non permette di effettuare valutazioni statisticamente robuste, specialmente per i test multilingui che costituiscono esplorazioni preliminari. Sebbene i test su italiano, francese e cinese forniscano indicazioni sulla trasferibilità cross-linguistica, la dimensione ridotta limita la generalizzabilità dei risultati ottenuti e evidenzia la necessità di benchmark multilingui estesi. Sebbene tutte le annotazioni siano pubblicate sotto licenza aperta per promuovere la riproducibilità, l'utilizzo dei dati richiede comunque un'attenta considerazione delle normative relative alla privacy e all'impiego appropriato delle informazioni.

Capitolo 5

Setup Sperimentale e Risultati

Questo capitolo presenta una sintesi chiara e coerente del setup sperimentale, dei metodi di valutazione, dei risultati quantitativi e qualitativi, nonché un confronto con lo stato dell'arte attuale. Tutti i dati mostrati appartengono agli script nel repository e possono essere riprodotti utilizzando i comandi menzionati.

5.1 Setup

Dataset. Gli esperimenti principali utilizzano il validation set inglese MU-SHROOM (50 elementi) in `data/mushroom.en-val.v2.jsonl`. Per la verifica multilingue utilizziamo il validation set italiano in `data/mushroom.it-val.v2.jsonl` (50 elementi).

Pipeline. La pipeline implementata stima per ogni token uno score di allucinazione HS_t e, a seconda della variante utilizzata, può applicare sostituzioni mirate. Le tre diverse varianti valutate sono: *No Fluency*, *Fluency*, *Mask*.

Ambiente. Test eseguiti in inferenza, GPU NVIDIA T4 16 GB (fallback CPU quando necessario), Ubuntu 22.04, nessun gradiente.

Modelli. (i) LLM di riferimento per probabilità token-level: `Mistral-7B`. (ii) NLI: `cross-encoder/nli-deberta-v3-large` e `MoritzLaurer/mDeBERTa-v3-base-mnli-xnli`. (iii) Fluency: `distilgpt2` per perplessità nelle varianti *Fluency*/*Mask*.

Librerie. `transformers` ≥ 4.41, `torch` ≥ 2.2.

5.1.1 Metriche

Esaminiamo: (i) **IoU** su hard labels a livello carattere; (ii) **correlazione di Pearson** tra soft labels di riferimento e punteggi HS_t previsti.

5.1.2 Iperparametri e Protocollo

Impostazioni ottimizzate per *No Fluency*: $\eta = 2.0$, $k = 6$ candidati top-k (early stop a $\rho = 0.95$), soglia probabilistica `prob_thresh`$= 0.06$. Il segnale NLI usa $p^{\text{nli}} = P_+ + P_=$ su finestra centrata. Per ogni esempio: allineamento etichette, calcolo e binarizzazione di HS_t, eventuale sostituzione (*Fluency*/*Mask*), computo IoU/Pearson e macro-media su 50 esempi.

5.1.3 Ottimizzazione sistematica degli iperparametri

Per definire i valori operativi sono state esplorate **200+ configurazioni** su MU-SHROOM (val EN, N=50), con una griglia che copre le dimensioni più sensibili:

- $\eta \in \{1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,10.0\}$ (scaling dello score)
- $k \in \{1,3,5,6,8,10\}$ (top-k candidati)
- `prob_thresh` $\in \{0.05,0.06,0.07,0.08,0.10,0.15\}$ (cutoff di probabilità)
- `use_pos` $\in \{\texttt{True}, \texttt{False}\}$ (vincolo morfosintattico)

Processo. Per ciascuna combinazione: calcolo HS_t, soglia adattiva τ_t, marcatura e (se previsto) sostituzione, quindi computo IoU (hard, char-level) e Pearson (soft). Selezione finale via macro-metrica e robustezza.

Configurazione scelta. *No Fluency* con $\eta = 2.0$, $k = 6$, `prob_thresh`=0.06, `use_pos`=False. La scelta riflette: (i) *copertura* su errori numerici/nominali, (ii) *stabilità* dei confini senza dilatazioni spurie, (iii) *assenza* di penalizzazioni di fluenza che non correlano con veridicità (cfr. §5.6).

5.2 Modelli NLI: DeBERTa-v3-large vs mDeBERTa-v3-base

L'analisi end-to-end suggerisce che l'impiego del modello NLI compatto è fattibile con una perdita minima in termini di *correlazione* (0.3204 rispetto a 0.3367, -4.9%) e maggiore riguardo a *IoU* (0.2561 rispetto a 0.3091, -17.1%). La decisione è influenzata dai vincoli computazionali e dalla metrica di interesse: il modello compatto mantiene gran parte della calibrazione, sebbene con un deterioramento nella copertura binaria.

5.2.1 Costo computazionale e limiti

Il collo di bottiglia è il classificatore NLI. La configurazione $k = 6$ bilancia costo e copertura. Limiti principali: ridotte dimensioni del set valido (N=50), dipendenza del modello NLI, variabilità cross-lingua. Per affidabilità statistica si pongono in primo piano bootstrap ed esami paired.

5.2.2 Conclusioni su NLI

- **Miglior configurazione**: *No Fluency* con il modello di NLI DeBERTa-v3-large (IoU 0.3091, Pearson 0.3367).
- **Compatto praticabile**: mDeBERTa-base mantiene 95% della correlazione con risparmio computazionale.
- **Cross-lingua**: i risultati italiani competitivi, confermando la trasferibilità metodologica.

5.3 Confronto con lo Stato dell'Arte

5.3.1 Protocollo di valutazione per SelfCheckGPT e CCP

In modo da assicurarsi che il confronto sia pari, **SelfCheckGPT** e **CCP** sono stati valutati rispetto allo *stesso* split di validazione inglese (`data/mushroom.en-val.v2.jsonl`, 50 esempi) ed

utilizzando le *stesse* metriche usate per le varianti sviluppate. I risultati sono stati raggiunti con implementazioni dedicate (non fine-tunate sul nostro set), descritte qui di seguito.

- **IoU** (hard, char-level): Intersection over Union tra span gold (a carattere) e span predetti proiettati a carattere.
- **Correlazione di Pearson** (soft): correlazione tra vettori di score soft gold e score previsti, allineati alla stessa granularità.

Implementazione CCP. La classe `CCPHallucinationDetector` impiega `GPT-2` per stime di probabilità condizionale:

1. *Estrazione claim*: segmentazione del testo in frasi (dove ogni frase è un claim).
2. *Mascheramento parziale*: sostituzione di una parte di parole con `<mask>` (evitando stopwords) e valutazione delle alternative rigenerate.
3. *Probabilità condizionali*: per ogni parte mascherata, si calcola $P(\text{parola} \mid \text{contesto})$ con il modello; aggregazione tramite media geometrica; hallucination score $= 1 -$ probabilità.
4. *Allineamento ai token*: gli score a livello claim vengono distribuiti ai token (via segmentazione per parole) e confrontati con le soft/hard label gold allineate al char-level (funzione di mapping `char_to_token_alignment`).

Implementazione SelfCheckGPT. La classe `SelfCheckGPT` genera campioni stocastici e misura consistenza semantica fra frasi:

1. *Sampling*: generazione di n varianti con nucleus sampling (top-p, temperatura).
2. *Similarità di frase*: per ogni frase dell'output originale, viene calcolata la massima cosine-similarity rispetto alle frasi dei campioni (`SentenceTransformer all-MiniLM-L6-v2`).
3. *Punteggio*: hallucination $= 1 -$ consistency a livello di frase, poi distribuito ai token per confronto con gold (stesso mapping e stesse regole di allineamento).

Valutatore e allineamento. Le metriche sono misurate con lo *stesso* valutatore (allineamento char-level, IoU su hard, Pearson su soft) usato per le nostre varianti, garantendo comparabilità diretta con i valori in Tab. 5.1. Eventuali file JSONL eventualmente riportati sono solo esportazioni dei risultati generati da queste implementazioni. **Note pratiche**. Esecuzione su GPU/CPU in inferenza, dtype half/float in base alla disponibilità; seed fissato per mascheramento CCP; $n = 3$ campioni per SelfCheckGPT per contenere i tempi; nessun tuning ad-hoc sui metodi di terze parti.

5.4 Risultati Quantitativi

5.4.1 Risultati su MU-SHROOM inglese (val, N=50): confronto varianti e baseline

Questa sezione sintetizza le prestazioni delle nostre varianti (No Fluency, Fluency, Mask) e di metodi esterni sullo *stesso* split inglese, usando **IoU** (hard a carattere) e **Pearson** (soft). Ogni riga indica una configurazione; quando specificato, *NLI large* = DeBERTa-v3-large, *NLI base* = mDeBERTa-v3-base.

Tabella 5.1: Prestazioni su `mushroom.en-val.v2.jsonl`

Approccio	IoU	Pearson
No Fluency (NLI large)	**0.3091**	**0.3367**
Baseline MALTO (detection-only)	0.2609	0.2989
Fluency (NLI large)	0.2935	0.3121
Mask (NLI large)	0.2885	0.2990
No Fluency (NLI base)	0.2561	0.3204
SelfCheckGPT (sentence)	0.2846	0.0079
CCP (token/span)	0.2176	0.0030

Sintesi. *No Fluency* con NLI grande (DeBERTa-v3-large) è la migliore in entrambe le metriche. Rispetto a SelfCheckGPT, la correlazione è 42x maggiore (0.3367 vs 0.0079), a parità di dataset. Con il modello NLI compatto (mDeBERTa-base) la perdita in Pearson è contenuta (0.3204, 95% della correlazione) mentre l'IoU cala di più (0.2561, 83% dell'IoU).

Sensibilità alla soglia di fluenza (Fluency, NLI large). Valutiamo l'effetto del criterio di accettazione delle sostituzioni basato sulla fluenza: *new_score* $\leq$ *original_score* $\times \alpha$, con $\alpha \in \{0.85,0.90,0.95,0.97\}$. Punteggi riportati su `mushroom.en-val.v2.jsonl` (N=50):

Tabella 5.2: Ablation del fattore α per la variante *Fluency* (NLI large). Criterio: *new_score* $\leq$ *original_score* $\times \alpha$.

α	IoU	Pearson
0.85	0.2844	0.3081
0.90	0.2844	0.3002
0.95	0.2935	0.3121
0.97	0.2983	0.3135

Commento. Una soglia più severa ($\alpha \in [0.85,0.90]$) riduce il numero di sostituzioni accettate, abbassando copertura e correlazione. Allentare leggermente il vincolo ($\alpha = 0.97$) migliora sia IoU sia Pearson rispetto a $\alpha = 0.95$, indicando un *sweet spot* intorno a 0.95–0.97. Nonostante il recupero, la variante *Fluency* rimane inferiore a *No Fluency* (IoU 0.3091, Pearson 0.3367), confermando che il vincolo di fluenza non aggiunge segnale di fattualità ma può aiutare a rifinire le sostituzioni quando calibrato con moderazione.

5.4.2 Risultati su MU-SHROOM italiano (val, N=50): trasferibilità della pipeline

Qui valutiamo la *trasferibilità* in italiano mantenendo protocollo, metriche e valutatore identici all'inglese: l'obiettivo è misurare quanto il segnale LM+NLI rimanga informativo cambiando lingua.

Tabella 5.3: Prestazioni su `mushroom.it-val.v2.jsonl`

Approccio	IoU	Pearson
No Fluency (NLI large)	0.2605	0.2528
No Fluency (NLI base)	0.2542	0.2338

Sintesi. Le performance italiane sono competitive e mostrano trasferibilità con gap moderato rispetto all'inglese (circa −16% IoU, −25% Pearson con NLI large).

5.4.3 Baselines ufficiali MU-SHROOM: neural e triviali sullo stesso valutatore

Per completezza riportiamo le **baseline ufficiali della challenge** e le confrontiamo sul *medesimo* dataset/valutatore usato per le nostre varianti: **IoU** (hard a carattere) e **Pearson** (soft). La riga *Neural* indica un classificatore token-level (XLM-R base); *Mark all/none* sono baseline triviali.

MU-SHROOM inglese (val, N=50): baseline neural e triviali. Confronto diretto su `mushroom.en-val.v2.jsonl`: *Neural* (XLM-R token-classification) vs *Mark all/none*. Stesse metriche e stesso valutatore usati nelle Tab. 5.1.

Tabella 5.4: Baselines ufficiali su `mushroom.en-val.v2.jsonl`

Baseline (challenge)	IoU	Pearson
Neural (token classification)	0.118962	0.0310306
Mark all (tutti allucinati)	0.000000	0.3489260
Mark none (nessun allucinato)	0.000000	0.0324675

MU-SHROOM italiano (val, N=50): baseline neural e triviali. Confronto diretto su `mushroom.it-val.v2.jsonl`: *Neural* (XLM-R token-classification) vs *Mark all/none*, con identico protocollo e valutatore delle Tab. 5.3.

Tabella 5.5: Baselines ufficiali su `mushroom.it-val.v2.jsonl`

Baseline (challenge)	IoU	Pearson
Neural (token classification)	0.0799714	0.0104172
Mark all (tutti allucinati)	0.0000000	0.2826150
Mark none (nessun allucinato)	0.0000000	0.0000000

Protocollo di calcolo.

- **Mark all**: si marcano tutti i token come allucinati (hard=1); le soft label derivano da probabilità uniformi massime. Produce IoU=0 (nessuna selettività) ma Pearson alto per via dell'allineamento monotono con soft gold.
- **Mark none**: si marcano tutti i token come non allucinati (hard=0); IoU=0 e Pearson vicino a 0 (o 0) per assenza di segnale.

- **Neural**: modello *token-classification* XLM-R base (`FacebookAI/xlm-roberta-base`). Addestrato su lingue diverse dalla lingua di test (train: val multilingua *leave-one-lang-out*; test: split della lingua target). Tokenizzazione con *offset mapping*, costruzione etichette token-level da `hard_labels` gold (inclusione completa del token dentro lo span), stima *soft* via softmax su label=1. Conversione a *char-level* tramite offset per entrambe le uscite (hard/soft) e valutazione con lo stesso calcolatore di metriche usato per i nostri metodi.

Per la baseline neurale facciamo riferimento allo script di training/testing riportato (XLM-R per token classification, mapping offset→caratteri, emissione di hard/soft labels e file `*.pred.jsonl`); per i baseline triviali, la logica *mark all/none* è implementata direttamente creando etichette costanti. Un ulteriore baseline stocastica (*random guess*) è disponibile nello *starter kit* e produce predizioni soft per carattere/posizione partendo dalla distribuzione osservata, valutata con lo stesso *scorer* (cfr. `baseline_random_guess.py`).
Sintesi comparativa. Quella che è stata confermata precedentemente, la variante migliore (*No Fluency, NLI large*) supera nettamente le baseline della challenge:

- **Inglese**: IoU **0.3091** vs 0.118962 (neural); Pearson **0.3367** vs 0.0310306 (neural). Rispetto a *mark all* mantiene IoU > 0 con una Pearson comparabile ma non triviale.
- **Italiano**: IoU **0.2605** vs 0.0799714 (neural); Pearson **0.2528** vs 0.0104172 (neural). Anche qui sovrasta *mark all* garantendo selettività sugli span (IoU > 0).

5.5 Analisi Qualitativa

L'ispezione dei primi due esempi inglesi mette in luce dei trade-off chiari:

- **No Fluency**: detection aggressiva e copertura elevata; può richiedere filtro linguistico a valle.
- **Fluency**: ovviamente, le sostituzioni tendono ad essere più conservative e grammaticalmente coerenti; migliore sui casi sottili.
- **Mask**: intervento minimo orientato alla precisione; preserva il testo quando il segnale è incerto.

Sul primo esempio italiano (Galizia: "4" vs corretto "7"), *No Fluency* con NLI large localizza e corregge il numero, confermando la robustezza cross-lingua.

Insights qualitativi sui casi studio

Per motivi di sintesi, si propone qui un confronto strutturato tra: (i) la risposta generata, (ii) gli span annotati come allucinati (gold, a livello carattere) e (iii) gli span predetti come allucinati (a livello token) dalle diverse configurazioni. I testi completi e i listing integrali sono rimandati all'Appendice; nel corpo del capitolo vengono mostrati solo estratti rappresentativi e i criteri di valutazione applicati.

Criteri di lettura. Le annotazioni gold sono span a livello *carattere*; i nostri predetti sono a livello *token* e vengono proiettati a caratteri per calcolare l'IoU locale. Questo comporta che: (i) un token in più non presente in gold riduce l'IoU; (ii) un leggero disallineamento dei limiti può abbassare l'IoU anche quando l'errore concettuale viene correttamente colpito.

val-en-1 — Nomi propri e numeri: copertura vs precisione. *Focus: bilanciare la copertura degli errori fattuali (silver, 2008) con il rischio di token extra e confini dilatati.*

- Prompt: *"What did Petra van Staveren win a gold medal for?"*
- Output generato: "Petra van `Stoveren` won a `silver` medal in the `2008` Summer Olympics in Beijing, China."
- Gold (char-level): {silver, 2008, Beijing, China}
- Predetti per configurazione (token-level) e IoU locale:
  - Baseline: {Stoveren} IoU: 0.000 (manca numerico/luogo gold)
  - No Fluency: {Petra, Stoveren, silver, 2008, Summer} IoU: 0.233 (copre silver/2008, non colpisce il luogo)
  - Fluency: {Stoveren, silver, 2008, Olympics} IoU: 0.250 (copre gli errori principali con minori inserzioni spurie)
  - Mask: {Stoveren} IoU: 0.000 (correzione minimale del nome)
- Insight: *No Fluency* massimizza copertura sugli errori fattuali ma genera correzioni aggressive; *Fluency* mantiene copertura sugli errori principali con sostituzioni più grammaticali; *Mask* privilegia precisione sul nome.

val-en-2 — Quantità puntuali: impatto dei token extra sui confini. *Focus: tutti colpiscono il numero "5", ma token adiacenti (es. "order", "genera.") riducono l'IoU locale.*

- Prompt: *"How many genera does the Erysiphales order contain?"*
- Output generato: "... contains `5` genera."
- Gold (char-level): {5}
- Predetti per configurazione (token-level) e IoU locale:
  - Baseline: {order, 5} IoU: 0.167 (colpisce il numero, include token extra)
  - No Fluency: {order, 5} IoU: 0.167 (come sopra)
  - Fluency: {order, 5, genera.} IoU: 0.077 (maggiore copertura ma più extra)
  - Mask: {order, 5, genera.} IoU: 0.077 (intervento minimale in sostituzione)
- Insight: tutte le varianti colpiscono il numero; *Fluency/Mask* tendono a normalizzare ("only one"), ma token extra riducono l'IoU.

val-it-1 — Numerali in italiano: correzione e coda spuria. *Focus: localizzazione del numero errato e rischio di piccole code spurie; utile un controllo di fluenza a valle.*

- Prompt: *"Da quante croci è composto lo stemma della Galizia?"*
- Output generato: "... composto da `4` croci."
- Gold (char-level): {4}
- Predetti per configurazione (token-level): No Fluency (IT, NLI large) {4, croci.} (IoU locale non riportato nel listing)
- Insight: il numero viene localizzato e corretto ("cinque croci"); la coda spuria segnala l'utilità di un controllo di fluenza a valle.

Lettura trasversale. *No Fluency* massimizza il recall su numeri/nomi ma può introdurre noise nelle sostituzioni; *Fluency* migliora leggibilità e plausibilità locale con rischio di aggiunte; *Mask* minimizza interventi non necessari. I pattern osservati spiegano i gap in IoU/Correlazione riportati nelle tabelle quantitative.

5.6 Perché *No Fluency* risulta la migliore

Allineamento tra segnale e obiettivo. L'hallucination score per token HS_t combina (i) la probabilità condizionata del modello di base $p_\theta(y_t \mid x, y_{<t})$ con (ii) un segnale semantico locale da NLI $p^{\text{nli}} = P(\text{entailment}) + P(\text{neutral})$. Entrambi sono *allineati* all'obiettivo di individuare contenuti non supportati dal contesto: il primo misura sorpresa/alternatività del token rispetto al modello, il secondo la (in)compatibilità semantica della variante rispetto al contesto. Il vincolo di fluenza (perplessità) invece è un *proxy di grammaticalità/frequenza locale*, non di veridicità: un frammento molto fluente può essere fattualmente sbagliato. Inserire la fluenza nel criterio decisionale tende quindi a spostare il confine decisionale verso forme frequenti/grammaticali, attenuando il segnale di fattualità che HS_t già cattura. Questo spiega la migliore **correlazione** delle soft label con *No Fluency* (Tab. 5.1).

Effetto sui confini degli span e sulle metriche. Le metriche usate sono sensibili ai confini: l'**IoU** si degrada quando vengono marcati token extra o si dilatano gli span. Le strategie basate su fluency spesso:

- normalizzano espressioni in più token (es. *"only one"*) o includono punteggiatura/parole adiacenti (*"genera."*),
- propongono riscritture grammaticalmente più naturali ma più *ampie* del minimo necessario.

Questi effetti migliorano la leggibilità locale, ma riducono l'IoU locale e possono diminuire la copertura degli span gold stretti. Al contrario, *No Fluency* tende a rilevare e sostituire i token fattualmente sensibili (nomi propri, numeri), massimizzando la sovrapposizione con gli span gold a parità di extra.

Modalità d'errore tipiche con fluenza. Nei casi studio si osservano: (i) *span dilation* (aggiunta di token non gold), (ii) *over-regularization* con sostituzioni grammaticalmente corrette ma semantiche ridondanti, e (iii) *under-correction* quando la perplessità penalizza una correzione necessaria perché meno frequente. Tali fenomeni spiegano perché la fluenza non incrementa la correlazione con le soft label e talvolta riduce l'IoU.

Robustezza cross-lingua. Il modello di fluenza usato (es. `distilgpt2`) è addestrato prevalentemente su inglese e cattura stile/distribuzioni linguistiche, non la veridicità. In scenari italiani l'allineamento è ancora più debole. *No Fluency*, basandosi su probabilità token e NLI, mantiene invece un segnale semanticamente più stabile cross-lingua, coerente con i risultati della Tab. 5.3.

Efficienza computazionale. *No Fluency* è anche la variante più **efficiente**. Nella variante *Fluency*, ogni token rilevato come allucinato, candidato quindi alla sostituzione, richiede il calcolo della perplessità di una *frase intera* (prefisso+token candidato), con un forward pass del LM di fluenza per ciascuna alternativa e talvolta per più lunghezze di tokenizzazione. Questo introduce un costo additivo di ordine $\mathcal{O}(C \cdot L_{\text{sent}})$ per token sotto esame, dove C è il numero di candidati e L_{sent} la lunghezza della frase. *No Fluency* evita completamente questi passaggi, riutilizzando i

segnali già disponibili (probabilità top-k e NLI su finestra), e si limita a una soglia su HS_t. In pratica, si elimina un modello addizionale nel loop e decresce sensibilmente il tempo di inferenza per esempio.

Sintesi. *No Fluency* massimizza l'allineamento con la task (fattualità), preserva confini stretti sugli span, evita modalità d'errore legate alla grammaticalità e riduce il costo computazionale. Questi fattori congiunti spiegano le migliori performance in **IoU** e soprattutto in **correlazione** con le soft label, confermate sperimentalmente nei dataset inglese e italiano.

Capitolo 6

Conclusioni e Sviluppi Futuri

6.1 Sintesi dei Risultati

Questa tesi ha affrontato la rilevazione delle allucinazioni negli LLM con due obiettivi: (i) chiarire il fenomeno sul piano concettuale e sperimentale; (ii) progettare e validare una pipeline token-level, pratica e riproducibile, per individuazione e correzione mirata.

Sul primo obiettivo, i Capitoli 1 e 2 offrono una lettura sistematica del problema, distinguendo fattualità vs faithfulness, livelli di granularità e famiglie metodologiche (conoscenza esterna, incertezza, auto-coerenza, probing). Sul secondo obiettivo, i Capitoli 3 e 5 presentano e valutano varianti della pipeline **MALTO** che combinano: (a) probabilità token-level di un modello di riferimento; (b) validazione semantica NLI; (c) controlli opzionali di fluenza/rigenerazione.

I risultati chiave, ottenuti sul subset inglese `mushroom.en-val.v2.jsonl` e verificati su italiano, sono:

- **Prestazioni**: la variante *No Fluency* con NLI large raggiunge le migliori metriche (IoU 0.3091, Pearson 0.3367), con forte vantaggio in calibrazione rispetto a SelfCheckGPT e CCP sullo stesso dataset.
- **Trade-off modello NLI**: il modello NLI compatto (mDeBERTa-base) conserva ≈95% della correlazione a costo computazionale ridotto, accettando un calo di IoU.
- **Robustezza cross-lingua**: su italiano i risultati sono competitivi (IoU 0.2605, Pearson 0.2528), a conferma della trasferibilità metodologica.
- **Riproducibilità**: tutti gli script e i listing qualitativi sono inclusi e documentati per la rigenerazione dei risultati.

Contributi. Riassumendo:

- una revisione critica dello stato dell'arte con enfasi sui metodi token-level e sulle metriche adeguate;
- una pipeline ibrida probabilistico-semantica a grana fine con varianti operabili in sola inferenza;
- un tuning sistematico dei parametri e un confronto controllato NLI large vs compatto;
- strumenti per l'analisi qualitativa (colorazione gold/pred/repl) utili al debug e alla comunicazione dei risultati.

6.2 Limiti

Pur migliorando la baseline, restano alcune criticità:

- **Dataset limitati**: i valid set (50 esempi/lingua) non consentono stime robuste della varianza né studi esaustivi per classe di errore.
- **Dipendenza da NLI**: la sensibilità del sistema a errori/deriva del classificatore NLI incide direttamente sull'hallucination score.
- **Hallucinations sottili**: lievi alterazioni semantiche possono essere classificate come neutral, riducendo il richiamo su casi difficili.
- **Costo**: la validazione NLI è il collo di bottiglia; varianti con rigenerazione introducono ulteriore latenza.
- **Generalizzazione**: i test multilingui sono preliminari; servono set più ampi e diversi domini applicativi.

6.3 Sviluppi Futuri

Le direzioni più promettenti includono:

- **Benchmark estesi**: costruzione/adozione di benchmark multilingui con migliaia di esempi/lingua per analisi statisticamente robuste (bootstrap, test appaiati) e ablation affidabili.
- **NLI più robusti e calibrati**: calibrazione specifica per lingua/dominio; esplorazione di modelli NLI contrastivi e di tecniche di temperature scaling/Platt per migliorare la probabilità calibrata.
- **Retrieve-then-verify**: combinare il rilevamento token-level con RAG e verifica citazionale per coprire meglio i casi sottili.
- **Soglie adattive più informative**: soglie per-POS/entità e strategie percentile dinamiche per bilanciare precision/recall in base al contesto.
- **Efficienza**: distillazione/quantizzazione del classificatore NLI; batching e caching per ridurre la latenza; early-exit quando il segnale è univoco.
- **Human-in-the-loop**: interfacce per accettare/rifiutare correzioni con tracciamento delle modifiche; esport dei log per audit.

6.4 Raccomandazioni Operative

Per un impiego pratico, la variante **No Fluency** con NLI large è la scelta consigliata quando si prioritizza la massima copertura a parità di qualità; il modello NLI compatto è preferibile in scenari vincolati da risorse, dato l'ottimo compromesso in correlazione. Laddove l'esperienza utente imponga output sempre ben formati, l'opzione **Fluency** può fungere da filtro di qualità; **Mask** è utile in impostazioni conservative.

6.5 Considerazioni Finali

Il lavoro mostra che un approccio token-level ibrido, ben calibrato e supportato da strumenti qualitativi trasparenti, consente di migliorare in modo misurabile la rilevazione delle allucinazioni e di portare l'analisi a un livello di granularità utile per la correzione mirata. Il quadro proposto è un passo verso pipeline più affidabili e interpretabili, in grado di coniugare accuratezza, efficienza e usabilità in contesti reali.

Appendice A

Materiale supplementare

A.1 Griglia di iperparametri

Componente	Parametro	Valori esplorati	Scelto
Candidati	K	8, 16, 32	16
Candidati	ρ (cum. prob.)	0.90, 0.95, 0.98	0.95
NLI	Finestra (token)	corta, media, lunga	media
Soglia	Percentile P_q	70, 80, 85	80
Iterazioni	$I_{\max}$	1, 2, 3	2

Tabella A.1: Griglia di iperparametri considerati e scelte operative.

A.2 Template di prompt e NLI

Esempio schematico di costruzione delle coppie per NLI locale:

```
Premessa:  "... contesto con finestra centrata su w ..."
Ipotesi:   "... variante con w -> w' ..."
Obiettivo: massimizzare P(entailment) + P(neutral)
```

A.3 Allineamento gold-hard/soft ai token

Regole pratiche adottate:

- Un token è marcato hard=1 se l'overlap carattere ̆2192 token supera 0.5.
- I valori soft per token sono medie degli score per carattere coperti.
- In parità di copertura si privilegia hard=0 per ridurre falsi positivi.

A.4 Dettagli implementativi

Versioni principali: `transformers` $\geq$ `4.41`, `torch` $\geq$ `2.2`. Inferenza in FP16, no-gradient, batch ridotti per NLI. Tokenizzazione con offset per proiezione delle etichette.

A.5 Codice di riferimento

La Listing A.1 mostra il cuore del calcolo dello *hallucination score*, estratto dal modulo `HallucinationScorer`. L'implementazione combina le probabilità dei token fornite dal modello di riferimento con i punteggi NLI delle alternative per stimare l'affidabilità di ciascuna parola.

```python
import numpy as np

class HallucinationScorer:
    def __init__(self, eta=1.0):
        self.eta = eta  # calibration factor (1.0 for LAV, >1.0 for RAV)

    def compute_score(self, token_probs, pnli_probs):
        """Calcola lo hallucination score per una parola."""
        p_i = np.array(token_probs)
        pnli_i = np.array(pnli_probs)
        denom = np.sum(p_i) + 1e-8
        weighted = np.sum(p_i * pnli_i)
        hs = 1.0 - (weighted / denom)
        return min(self.eta * hs, 1.0)

    def score_word(self, word, context, alt_words, prob_model, nli_model):
        token_probs, pnli_probs = [], []
        for alt in alt_words:
            if alt == word:
                continue
            p_alt, _ = prob_model.get_word_probability(alt, context)
            hyp = context + " " + alt
            nli_result = nli_model.get_nli_probs(context + " " + word, hyp)
            token_probs.append(p_alt)
            pnli_probs.append(nli_result["pnli"])
        if not token_probs:
            return 0.0
        return self.compute_score(token_probs, pnli_probs)
```

Listing A.1: Estratto da `models/hallucination_scorer.py`

La funzione `run_pipeline`, riportata in Listing A.2, implementa il flusso operativo descritto nel Capitolo 3. Il codice legge gli esempi, calcola gli score token-level e, nella variante con mascheramento, sostituisce o rigenera le parole sospette.

```python
def run_pipeline(data_path, model_name_prob, device="cuda", eta=1.0,
                 top_k=10, prob_thresh=0.005, debug=False,
                 output_path=None, use_pos=True, use_fluency=True,
                 use_contradiction=False, use_mask=False):

    data = load_mushroom_dataset(data_path)
    already_processed_ids = set()
    processed = []

    if output_path and os.path.exists(output_path):
        with open(output_path, "r", encoding="utf-8") as f:
            for line in f:
                try:
                    saved = json.loads(line)
                    already_processed_ids.add(saved["id"])
                    processed.append(saved)
                except json.JSONDecodeError:
                    continue

    prob_model = LargeModelWrapper(model_name=model_name_prob,
```

```
                                device=device, top_k=top_k,
                                prob_thresh=prob_thresh,
                                quantize=True)
    nli_model = NLIWrapper(device=device)
    scorer = HallucinationScorer(eta=eta)

    for example in tqdm(data, desc="Processing examples"):
        if example["id"] in already_processed_ids:
            continue

        output_text = example["model_output_text"].strip()
        words = output_text.split()
        updated_words = words.copy()
        hs_values = []

        for i, word in enumerate(words):
            prefix = example["model_input"].strip() + " " + " ".join(updated_words
    [:i])
            alt_data = prob_model.generate_alternatives(prefix)
            alt_words = [a["full_word"] for a in alt_data if a["full_word"] !=
    word]
            hs = scorer.score_word(word, prefix, alt_words, prob_model, nli_model)
            hs_values.append(hs)
            threshold = max(0.4, np.percentile(hs_values, 80))
            if hs < threshold:
                continue

            original_score = score_sentence_fluency(f"{prefix} {word}".strip()) \
                if use_fluency and not use_contradiction else None
            original_pos = get_pos_tag(word) if use_pos else None
            best_replacement = word
            best_score = float("inf") if use_fluency and not use_contradiction
    else -1.0
            chosen = False

            if use_mask:
                premise = f"{prefix} {word}".strip()
                best_alt = None
                best_contrad_score = -1.0
                for alt in alt_data:
                    candidate_full = alt["full_word"].strip()
                    if not is_valid_replacement(candidate_full):
                        continue
                    scores = nli_model.get_nli_probs(premise, f"{prefix} {
    candidate_full}".strip())
                    contr_prob = scores["contradiction"]
                    if contr_prob > best_contrad_score:
                        best_contrad_score = contr_prob
                        best_alt = alt
                if best_alt is not None:
                    tokens = best_alt["full_word"].strip().split()
                    best_score = float("inf")
                    for k in range(len(tokens), 0, -1):
                        candidate = " ".join(tokens[:k])
                        if not is_valid_replacement(candidate):
                            continue
                        new_sent = f"{prefix} {candidate}".strip()
                        new_score = score_sentence_fluency(new_sent)
                        if new_score < best_score:
                            best_score = new_score
                            best_replacement = candidate
                    if best_replacement != word:
```

```
                        token_info = {"replaced": True,
                                      "replacement": best_replacement,
                                      "contradiction_prob": best_contrad_score}
                        updated_words[i] = best_replacement
                        per_token_info.append(token_info)
                        continue
            elif use_contradiction:
                premise = f"{prefix} {word}".strip()
                for alt in alt_data:
                    candidate = alt["full_word"].strip()
                    if not is_valid_replacement(candidate):
                        continue
                    scores = nli_model.get_nli_probs(premise, f"{prefix} {
candidate}".strip())
                    contr_prob = scores["contradiction"]
                    if contr_prob > best_score:
                        best_score = contr_prob
                        best_replacement = candidate
                if best_replacement != word:
                    updated_words[i] = best_replacement
                    per_token_info.append({"replaced": True,
                                           "replacement": best_replacement,
                                           "contradiction_prob": best_score})
                    continue
            else:
                for alt in sorted(alt_data, key=lambda x: -x["combined_prob"]):
                    full = alt["full_word"].strip()
                    tokens = full.split()
                    for k in range(len(tokens), 0, -1):
                        candidate = " ".join(tokens[:k])
                        if not is_valid_replacement(candidate):
                            continue
                        candidate_pos = get_pos_tag(candidate) if use_pos else
None
                        if use_pos and original_pos and candidate_pos !=
original_pos:
                            continue
                        if use_fluency:
                            new_sent = f"{prefix} {candidate}".strip()
                            new_score = score_sentence_fluency(new_sent)
                            if new_score < best_score:
                                best_score = new_score
                                best_replacement = candidate
                            if new_score <= original_score * 0.95:
                                chosen = True
                                break
                        else:
                            best_replacement = candidate
                            chosen = True
                            break
                    if chosen:
                        break
                if chosen and best_replacement != word:
                    updated_words[i] = best_replacement

        example["updated_output"] = " ".join(updated_words)
        soft = calculate_soft_labels(words, hs_values, eta)
        hard = recompute_hard_labels(soft)
        example["soft_labels"] = soft
        example["hard_labels"] = hard
        processed.append(example)

```

```
138     return processed
```

Listing A.2: Estratto da `pipeline.py`: funzione `run_pipeline`

Questi estratti forniscono una visione concreta delle componenti software alla base della pipeline sperimentale descritta nei capitoli principali.

A.6 Listing qualitativi riproducibili

Per completezza, includiamo qui gli output testuali generati dagli script nel repository per i casi di studio discussi nel Capitolo 5.

La legenda colori per gli span evidenziati:

- verde = etichette gold (riferimento/ground truth);
- rosso = token/span segnalati dalla pipeline (predetti);
- ciano = parole sostitutive proposte (replacements).

I file sorgente dei listing si trovano in `analysis_archive/` e sono riproducibili con gli script indicati in Sezione 5.1.2.

```
1
2   === Sample ID val-en-1 ===
3   Prompt: What did Petra van Staveren win a gold medal for?
4
5   Generated output (original):
6   Petra van Stoveren won a silver medal in the 2008 Summer Olympics in
        Beijing, China.
7
8   Gold labels:
9   [(25, 31), (45, 49), (69, 83)]
10  Petra van Stoveren won a silver medal in the 2008 Summer Olympics in Beijing,
        China.
11
12  === MALTO Baseline ===
13  IoU vs gold: 0.000
14  Pred labels:
15  [(10, 18)]
16  Petra van Stoveren won a silver medal in the 2008 Summer Olympics in
        Beijing, China.
17
18  Per-token summary:
19  (no per-token analysis available)
20
21  === No Fluency ===
22  IoU vs gold: 0.233
23  Pred labels:
24  [(0, 5), (10, 18), (25, 31), (45, 49), (50, 56)]
25  Petra van Stoveren won a silver medal in the 2008 Summer Olympics in Beijing,
        China.
26
27  Updated output:
28  Petra van Staveren is won a gold medal in the 20 medal in the 2008 Summer
        Olympics in Beijing, China.
```

```
29
30 Replacements (cyan):
31 Petra van Staveren is won a a gold medal in medal in the the
       20 Summer Olympics in Beijing, China.
32  - pos 2: 'Stoveren' -> 'Staveren is'
33  - pos 5: 'silver' -> 'a gold medal in'
34  - pos 9: '2008' -> 'the 20'
35
36 Per-token summary:
37 Tokens flagged (hs >= thr) or replaced:
38  - #2 'Stoveren' HS=1.000 thr=0.710 -> 'Staveren is'
39  - #5 'silver' HS=1.000 thr=1.000 -> 'a gold medal in'
40  - #9 '2008' HS=1.000 thr=1.000 -> 'the 20'
41
42 === With Fluency ===
43 IoU vs gold: 0.250
44 Pred labels:
45 [(10, 18), (25, 31), (45, 49), (57, 65)]
46 Petra van Stoveren won a silver medal in the 2008 Summer Olympics in Beijing,
       China.
47
48 Updated output:
49 Petra van Staveren is won a gold medal in silver medal in the 2008 Summer
       Olympics in Beijing, China.
50
51 Replacements (cyan):
52 Petra van Staveren is won a a gold medal
       in medal in the 2008 Summer Olympics in Beijing, China.
53  - pos 2: 'Stoveren' -> 'Staveren is'
54  - pos 5: 'silver' -> 'a gold medal in'
55
56 Per-token summary:
57 Tokens flagged (hs >= thr) or replaced:
58  - #2 'Stoveren' HS=1.000 thr=0.680 -> 'Staveren is'
59  - #5 'silver' HS=1.000 thr=1.000 -> 'a gold medal in'
60
61 === Mask ===
62 IoU vs gold: 0.000
63 Pred labels:
64 [(10, 18)]
65 Petra van Stoveren won a silver medal in the 2008 Summer Olympics in
       Beijing, China.
66
67 Updated output:
68 Petra van Staveren won a silver medal in the 2008 Summer Olympics in
       Beijing, China.
69
70 Replacements (cyan):
71 Petra van Staveren won a silver medal in the 2008 Summer Olympics in
       Beijing, China.
72  - pos 2: 'Stoveren' -> 'Staveren'
73
74 Per-token summary:
75 Tokens flagged (hs >= thr) or replaced:
```

```
 - #2 'Stoveren' HS=1.000 thr=0.680 -> 'Staveren'
```

Listing A.3: compare_approaches (val-en-1)

```

=== Sample ID val-en-2 ===
Prompt: How many genera does the Erysiphales order contain?

Generated output (original):
The Elysiphale order contains 5 genera.

Gold labels:
[(30, 31)]
The Elysiphale order contains 5 genera.

=== MALTO Baseline ===
IoU vs gold: 0.167
Pred labels:
[(15, 20), (30, 31)]
The Elysiphale order contains 5 genera.

Per-token summary:
(no per-token analysis available)

=== No Fluency ===
IoU vs gold: 0.167
Pred labels:
[(15, 20), (30, 31)]
The Elysiphale order contains 5 genera.

Updated output:
The Elysiphale order contains 13 gener contains 5 genera.

Replacements (cyan):
The Elysiphale order contains 1 contains 13 gener genera.
 - pos 2: 'order' -> 'order contains 1'
 - pos 4: '5' -> '13 gener'

Per-token summary:
Tokens flagged (hs >= thr) or replaced:
 - #2 'order' HS=0.978 thr=0.591 -> 'order contains 1'
 - #4 '5' HS=1.000 thr=0.983 -> '13 gener'

=== With Fluency ===
IoU vs gold: 0.077
Pred labels:
[(15, 20), (30, 31), (32, 39)]
The Elysiphale order contains 5 genera.

Updated output:
The Elysiphale order contains only one genus, E genera.

Replacements (cyan):
The Elysiphale order contains only one one genus, E
```

```
 - pos 4: '5' -> 'only one'
 - pos 5: 'genera.' -> 'one genus, E'

Per-token summary:
Tokens flagged (hs >= thr) or replaced:
 - #2 'order' HS=0.483 thr=0.400
 - #4 '5' HS=1.000 thr=0.586 -> 'only one'
 - #5 'genera.' HS=1.000 thr=1.000 -> 'one genus, E'

=== Mask ===
IoU vs gold: 0.077
Pred labels:
[(15, 20), (30, 31), (32, 39)]
The Elysiphale order contains 5 genera.

Updated output:
The Elysiphale order contains only one genera.

Replacements (cyan):
The Elysiphale order contains only one one
 - pos 4: '5' -> 'only one'
 - pos 5: 'genera.' -> 'one'

Per-token summary:
Tokens flagged (hs >= thr) or replaced:
 - #2 'order' HS=0.483 thr=0.400
 - #4 '5' HS=1.000 thr=0.586 -> 'only one'
 - #5 'genera.' HS=1.000 thr=1.000 -> 'one'
```

Listing A.4: compare_approaches (val-en-2)

```

=== Sample ID val-it-1 ===
Prompt: Da quante croci  composto lo stemma della Galizia?

Generated output (original):
Lo stemma della Galizia, regione autonoma della Spagna,  composto da 4
    croci.

Gold labels:
[[70, 71]]
Lo stemma della Galizia, regione autonoma della Spagna,  composto da 4
    croci.

=== No Fluency (IT, NLI large) ===
IoU vs gold: (see quant results)
Pred labels:
[[70, 71], [72, 78]]
Lo stemma della Galizia, regione autonoma della Spagna,  composto da 4
    croci.

Updated output:
Lo stemma della Galizia, regione autonoma della Spagna,  composto da
    cinque croci. Qu croci.
```

```
20
21 Replacements:
22  - pos 11: '4' -> 'cinque croci'
23  - pos 12: 'croci.' -> 'croci.  Qu'
24
25 Per-token summary:
26  - #11 '4' HS=1.000 thr=0.400 -> 'cinque croci'
27  - #12 'croci.' HS=0.498 thr=0.400 -> 'croci. Qu'
```

Listing A.5: inspect (val-it-1, No Fluency)

Il codice, i dati e gli script per riprodurre gli esperimenti sono disponibili su GitHub:

github.com/ale-romeo/Hallucination-Detection

Bibliografia

[1] Ziwei Ji et al. «Survey of Hallucination in Natural Language Generation». In: *ACM Computing Surveys* 55.12 (2023), pp. 1–38. DOI: 10.1145/3571730 (cit. alle pp. 1, 2).

[2] Joshua Maynez, Shashi Narayan, Bernd Bohnet e Ryan McDonald. «On Faithfulness and Factuality in Abstractive Summarization». In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 1906–1919. DOI: 10.18653/v1/2020.acl-main.173 (cit. alle pp. 1, 2).

[3] Vijay Rawte, Rashi Mehrotra e Tanmoy Chakraborty. «A Survey on Hallucination in Natural Language Generation: Causes, Evaluation, and Mitigation». In: *arXiv preprint arXiv:2309.07864* (2023) (cit. alle pp. 1–3).

[4] Xingyao Zhao, Tianyi Zhang et al. «SelfCheck: Step-by-step Reasoning Error Detection in Large Language Models». In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2023 (cit. alle pp. 1, 3, 11).

[5] Ziyu Hu et al. «RefChecker: Reference-based Fine-grained Hallucination Checker and Benchmark for Large Language Models». In: *arXiv preprint arXiv:2405.14486* (2024) (cit. alle pp. 2, 3, 6, 7).

[6] Li Zhang et al. «KnowHalu: Multi-form Knowledge-based Hallucination Detection in Large Language Models». In: *arXiv preprint arXiv:2404.02935* (2024) (cit. alle pp. 2, 5, 7).

[7] Potsawee Manakul, Adian Liusie e Mark Gales. «SelfCheckGPT: Zero-resource Black-box Hallucination Detection for Generative Large Language Models». In: *arXiv preprint arXiv:2303.08896* (2023) (cit. alle pp. 3, 9, 10).

[8] Chengwei Xu et al. «Long-form Hallucination Detection with Self-elicitation». In: *arXiv preprint arXiv:2502.08767* (2024) (cit. alle pp. 3, 10, 11).

[9] Yifan Chen et al. «InterrogateLLM: Backward Query Reformulation for Hallucination Detection». In: *arXiv preprint arXiv:2403.02889* (2024) (cit. alle pp. 3, 11).

[10] Ekaterina Fadeeva, Aleksandr Rubashevskii, Artem Shelmanov et al. «Fact-Checking the Output of Large Language Models via Token-Level Uncertainty Quantification». In: *arXiv preprint arXiv:2403.04696* (2024). URL: https://arxiv.org/abs/2403.04696 (cit. alle pp. 3, 8, 16).

[11] Gabriel Arteaga, Wei Zhao, Jonathan Smith e Minho Lee. «Uncertainty-based Hallucination Detection in Large Language Models via Lightweight Ensembles». In: *arXiv preprint arXiv:2409.02976* (2024) (cit. alle pp. 3, 8).

[12] MALTO Team. «SemEval-2025 Task 3: Multilingual Shared Task on Hallucination Detection in Large Language Models (Mu-SHROOM)». In: *Proceedings of the International Workshop on Semantic Evaluation (SemEval)*. 2025 (cit. alle pp. 6, 28).

[13] Adam Nowak et al. «Hallucination Detection in LLMs Using Spectral Features of Attention Maps». In: *arXiv preprint arXiv:2502.17598* (2025) (cit. a p. 12).

[14] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes e Yejin Choi. «The Curious Case of Neural Text Degeneration». In: *International Conference on Learning Representations* (2020). URL: `https://openreview.net/forum?id=rygGQyrFvH`.