

UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI ECONOMIA "GIORGIO FUÀ"

Corso di Laurea Magistrale in Data Science per l'Economia e le Imprese

Scoperta di dipendenze tra corsi universitari tramite LLM locali

Discovering dependencies between university courses using local LLMs

Relatore:
Prof. Domenico Potena
Correlatrice:
Dott.ssa Chiara Gobbi

Tesi di Laurea di:
Martina Leporati

Anno accademico 2025-2026

INDICE

INTRODUZIONE 3
CAPITOLO I 8
CONTESTO TEORICO 8
I.1 Elicitation di modelli di processo 8
I.2 Large Language Models 13
I.3 Tecniche di prompting per task strutturati 16
I.4 Stato dell'arte 18
I.5 Baseline Metodologica 20
CAPITOLO II 27
METODOLOGIA PROPOSTA 27
II.1 Preparazione dell'input: la segmentazione dei syllabus 27
II.2 La pipeline adattata: rimozione dello Step di revisione 30
II.3 La progettazione del prompt dello STEP 2 31
II.4 Il peso p e il filtraggio delle dipendenze 35
II.5 La valutazione singola e multipla 36
CAPITOLO III 37
CASO DI STUDIO E SETUP SPERIMENTALE 37
III.1 Il corso di laurea in Ingegneria Informatica e dell'Automazione 37
III.2 Dataset dei syllabus e formato di input 40
III.3 Il grafo dei docenti 42
III.4 Infrastruttura e configurazione del modello 44
CAPITOLO IV 50
ADATTAMENTO AL MODELLO LOCALE 50
IV.1 Il comportamento del modello sulla pipeline originale 50
IV.2 Rimozione step di revisione 54
IV.3 Interventi su prompt e input 56

CAPITOLO V 61
INSTABILITA' DEL SINGOLO RUN, ENSEMBLE E RANDOMIZZAZIONE 61
V.1 Il comportamento non deterministico del modello locale 61
V.2 L'ensemble come rimedio 67
V.3 Le prove di randomizzazione 78
V.4 Analisi degli errori 89
CONCLUSIONI 98
BIBLIOGRAFIA 102
SITOGRAFIA 106
APPENDICE A 107

INTRODUZIONE

Il presente lavoro di tesi si pone l'obiettivo di sviluppare una metodologia in grado di estrarre in modo automatico le dipendenze di propedeuticità tra corsi universitari, utilizzando Large Language Models (LLM) eseguiti in locale. Il caso specifico è costituito dal corso di laurea triennale in Ingegneria Informatica e dell'Automazione dell'Università Politecnica delle Marche, del quale si intende ricostruire, a partire dai syllabus dei diversi insegnamenti, le relazioni di prerequisito che intercorrono tra essi.

La progettazione di un corso di laurea è un'attività complessa, nella quale uno degli aspetti più delicati è la definizione dell'ordine in cui gli insegnamenti vengono erogati. Tale ordine non è arbitrario: corsi diversi condividono spesso conoscenze e competenze acquisite in precedenza che devono essere necessariamente individuate, sia per garantire una progressione didattica coerente, sia per supportare lo studente nella pianificazione del proprio piano di studi.

Tradizionalmente, l'individuazione di queste dipendenze è svolta dai docenti, attraverso un processo collegiale che richiede tempo e che dipende dalla conoscenza che ognuno di essi ha del proprio insegnamento e di quelli ad esso correlati. Automatizzare tale attività può ridurre e migliorare questo processo, sia verificando la coerenza del manifesto degli studi sia semplificando il compito dei professori.

Inoltre, può diventare uno strumento di supporto agli studenti, in particolare in contesti privi di propedeuticità formali, nei quali gli esami possono essere sostenuti in ordine libero. Infine, assume particolare rilevanza negli ambienti di apprendimento online dinamici, dove l'offerta formativa cambia con frequenza e la rivalutazione manuale delle propedeuticità a ogni aggiornamento del catalogo diventa rapidamente impraticabile.

Il presente lavoro si colloca nell'area dell'*educational data mining*, la disciplina che sviluppa metodi per esplorare i dati provenienti dagli ambienti educativi al fine di comprendere meglio gli studenti e i contesti in cui apprendono[1]. Più nello specifico, il problema qui affrontato può essere formalizzato come un caso di *process model elicitation*[2], nell'ambito della ricerca sul *process mining*[3]. Riprendendo l'analogia adottata nell'articolo "*Leveraging LLMs to Discover Causal Dependencies: A Case Study on a University Program*"[4], da cui la tesi prende le mosse, un corso di laurea è interpretato come un processo, gli insegnamenti che lo compongono come attività

[1] Definizione dell'International Educational Data Mining Society

[2] Attività di definizione di un modello di processo a partire dai requisiti o dalla conoscenza di esperti di dominio, in assenza di un event log da cui estrarlo. Cfr. Diamantini et al. (2025).

[3] Disciplina che analizza i log degli eventi per scoprire, verificare e migliorare i processi reali. Cfr. van der Aalst, W.M.P. (2016). *Process Mining: Data Science in Action*, 2ª ed., Springer.

[4] Diamantini, C., Gobbi, C., Mele, A., Potena, D., Rossetti, C. (2025). *Leveraging LLMs to Discover Causal Dependencies: A Case Study on a University Program*, pp. 237–248.

e le relazioni di propedeuticità come dipendenze causali tra di esse: una dipendenza A->B esprime il fatto che la piena comprensione dei contenuti di B richiede le competenze acquisite in A. L'obiettivo dell'elicitazione consiste allora nell'individuare, in assenza di un *event log*[5] delle attività effettivamente svolte, l'insieme di tali dipendenze. Per farlo, la metodologia qui presentata sfrutta i Large Language Models, modelli di deep learning addestrati su vaste quantità di dati e capaci di comprendere e generare linguaggio naturale.

Il punto di partenza è costituito dall'approccio metodologico proposto nell'articolo di Diamantini et al. già citato, ulteriormente esteso in un precedente lavoro di tesi dal titolo *"Definizione di un grafo di dipendenze tramite LLM, con applicazione a corsi universitari"* di Salvatori Marika[6]. Rispetto a tali lavori, basati sull'uso di modelli commerciali di tipo cloud, nel presente ambito si vuole trasferire e adattare la metodologia al contesto di un LLM locale, ovvero un modello eseguito integralmente su infrastruttura controllata dall'istituzione (nel caso specifico, una macchina dedicata del dipartimento con il modello gpt-oss:20b servito tramite

[5] Nel *process mining*, un *event log* è l'insieme delle tracce che documentano ciò che è realmente accaduto durante l'esecuzione di un processo: ogni traccia raccoglie, in ordine, le attività eseguite nell'ambito di un singolo caso.

[6] Salvatori, M. (2024-2025). *Definizione di un grafo di dipendenze tramite LLM, con applicazione a corsi universitari*. Tesi di Laurea Magistrale in Data Science per l'Economia e le Imprese, UNIVPM, relatore Prof. D. Potena, correlatrice Dott.ssa C. Gobbi.

l'ambiente Ollama[7]). Tutto questo al fine di superare alcuni limiti intrinseci dei modelli cloud: *vincoli di privacy,* che impediscono di fornire al modello materiale didattico riservato (come dispense, slide o estratti di libri di testo) e limitando quindi l'input ai soli syllabus pubblicati sui siti istituzionali; *costi e limitazioni d'uso*, legati al consumo di token e ai vincoli imposti dai fornitori del servizio; *assenza di controllo sulla pipeline*, in quanto il modello è opaco e soggetto ad aggiornamenti non documentati che possono modificarne il comportamento nel tempo.

Allo stesso tempo, tale transizione introduce nuove sfide: un LLM locale ha capacità di ragionamento e ampiezza di contesto inferiori ed è dunque verosimile che il prompt sviluppato per un LLM commerciale non sia direttamente trasferibile e debba essere riprogettato per essere efficace.

A tale fine gli obiettivi specifici della tesi possono essere articolati come segue:

1. Adattare alla nuova architettura la metodologia sviluppata in precedenza, valutando e individuando, ove necessario, semplificazioni del prompt;
2. Esplorare le strategie metodologiche abilitate dal contesto locale, tra cui la stratificazione del dataset per ridurre l'ampiezza dell'input fornito in ciascuna chiamata, le tecniche di ensembling tra più run, volte ad attenuare

[7] Ollama — *Get up and running with large language models locally*. https://ollama.com. Modello gpt-oss:20b: https://ollama.com/library/gpt-oss.

la variabilità tra esecuzioni e il fenomeno delle allucinazioni. Il venir meno del vincolo di riservatezza apre inoltre, la possibilità di arricchire l'input con materiale didattico aggiuntivo (slide, dispense).

3. Valutare in modo critico i risultati ottenuti rispetto al grafo dei docenti, considerando le opportune metriche di qualità.

Per quanto riguarda la struttura, l'elaborato si articola in cinque capitoli.
Il primo si sofferma sul contesto teorico di riferimento; il secondo presenta i lavori usati come base di partenza; nel terzo si introduce il caso di studio e il relativo setup sperimentale; il quarto ripercorre l'adattamento della pipeline al modello locale e infine il quinto, affronta l'instabilità della singola esecuzione, l'ensemble come rimedio e le prove di randomizzazione. Chiudono l'elaborato le conclusioni.

CAPITOLO I

CONTESTO TEORICO

I.1 Elicitation di modelli di processo

I.1.1 Process mining: dai log ai modelli di processo

Il process mining si colloca tra il data mining e la modellazione e analisi dei processi, applicando tecniche di analisi dei dati a una particolare tipologia di dato: il processo, inteso come insieme di attività collegate e coordinate, sequenziali, cicliche o parallele. Il suo assunto di partenza è che uno stesso processo, rappresentabile come un workflow di attività, può essere eseguito in modi diversi. In termini più specifici, le sue singole esecuzioni concrete (dette *tracce*), differiscono. Basti pensare, ad esempio, al contesto universitario, nel quale gli studenti seguono ritmi e attività diverse. Tracciando e unendo in modo ordinato queste tracce è possibile ottenere *l'event log*, che rappresenta la base da cui è poi possibile ricostruire come il processo si svolge effettivamente, anziché supporlo.

Tradizionalmente, si distinguono tre tipi di process mining[8].

Il *Process discovery* è una tecnica che ricava il modello di processo dall'event log senza utilizzare alcuna conoscenza a priori.

[8] Van der Aalst, W.M.P. (2016). *Process Mining: Data Science in Action*, 2ª ed., Springer

Il *Conformance checking* prende insieme i modelli esistenti e i log per individuare eventuali deviazioni tra comportamento prescritto e comportamento osservato, misurando l'allineamento tra il modello e la realtà. Infine, l'*enhancement* utilizza il log per estendere o migliorare un modello definito a priori, arricchendolo con informazioni sul processo reale registrate in un log degli eventi.
Tutti e tre presuppongono la disponibilità di event log di qualità adeguata ed è proprio questo elemento a venire meno nel contesto affrontato in questo lavoro.

I.1.2 Quando l'event log non esiste: la process model elicitation.

Ci sono situazioni in cui un modello di processo deve essere definito senza poter osservare il processo in esecuzione. Durante la fase di progettazione, ad esempio, il processo non è ancora stato avviato e non esistono quindi esecuzioni da registrare, sono noti solo i requisiti e gli obiettivi che esso deve soddisfare. In tali scenari il modello non può essere derivato dai dati ma deve essere *elicitato*, cioè ricostruito a partire dai requisiti e dalla conoscenza di esperti di dominio che li interpretano e li formalizzano in una rappresentazione strutturata (cfr. Diamantini et al., 2025).

L'elicitation presenta due difficoltà ricorrenti, perfettamente esemplificabili in tale contesto. La prima è che la conoscenza necessaria è distribuita tra più esperti, qui i docenti, ciascuno dei quali conosce bene il proprio insegnamento e i suoi requisiti ma solo approssimativamente gli altri. Ricomporre queste visioni parziali in un

modello coerente richiede un confronto collegiale oneroso. La seconda è che tale conoscenza rimane in larga parte tacita, tradotta in prassi consolidate più che in documentazione esplicita, risultando difficile da verificare, aggiornare e trasferire.

Il caso di studio riportato ricade quindi esattamente in questo scenario, in quanto le propedeuticità tra insegnamenti non sono osservabili in un log: la carriera di uno studente registra l'ordine in cui gli esami sono stati sostenuti, ma tale ordine riflette scelte individuali, non la dipendenza concettuale tra i contenuti dei corsi. L'unica traccia documentale delle conoscenze erogate e richieste è il syllabus, cioè un testo in linguaggio naturale. Il problema si configura dunque come elicitation a partire da documentazione testuale, ed è per questo che, nel Capitolo I.2, si ricorrerà a strumenti in grado di interpretare testo non strutturato.

I.1.3 Il Directly-Follows Graph

Un modello di processo può essere espresso usando diversi formalismi, tra cui tra i più diffusi e tra i più facilmente interpretabili anche da un non esperto, il Directly-Follows Graph (DFG)[9]. Un DFG è un grafo orientato i cui nodi corrispondono alle attività e gli archi a relazioni di successione immediata tra le attività.

[9] van der Aalst, W.M.P. (2019). *A practitioner's guide to process mining: Limitations of the directly-follows graph*. Procedia Computer Science, 164, 321–328. DOI: 10.1016/j.procs.2019.12.189.

Nel presente lavoro il DFG viene adottato come formalismo di output: il risultato dell'estrazione è un grafo orientato i cui nodi sono gli insegnamenti e i cui archi rappresentano le dipendenze di propedeuticità. Va precisato che la relazione codificata dagli archi non è qui la relazione di successione diretta osservata in un log, ma una relazione di precedenza concettuale inferita dai contenuti dei corsi: il grafo conserva la struttura del DFG, mentre la sua semantica è quella di un modello di prerequisiti. Rilevanti sono inoltre, le proprietà strutturali del grafo, in particolare l'assenza di cicli e la ridondanza degli archi transitivi, per le quali si rimanda al Capitolo I.5.3, dove verranno discusse in relazione alla valutazione dei risultati.

I.1.4 Educational process mining e analisi curriculare

L'educational process mining consiste nell'applicazione di tecniche di process mining a dati provenienti da ambienti educativi[10].

Al suo interno, l'analisi curriculare (o *curriculum mining*) si concentra specificamente sul percorso formativo seguito dagli studenti per conseguire gli obiettivi di un programma di studi, con finalità di supporto alle decisioni istituzionali e di miglioramento della qualità della didattica.

[10] Bogarín, A., Cerezo, R., Romero, C. (2018). *A survey on educational process mining*. WIREs Data Mining and Knowledge Discovery, 8(1), e1230. DOI: 10.1002/widm.1230.

Una recente rassegna di Calegari, D., Delgado, A. (2025). *A Systematic Review on Process Mining for Curricular Analysis*[11], ha esaminato ventisette studi primari dedicati all'applicazione del process mining all'analisi curriculare, classificandoli in cinque obiettivi di ricerca: la scoperta delle traiettorie formative, l'individuazione di deviazioni nel comportamento degli studenti, l'analisi dei colli di bottiglia, lo studio dell'abbandono e la generazione di raccomandazioni. Tutti e cinque partono dai dati di carriera degli studenti e ricostruiscono, per via induttiva, come il curriculum viene effettivamente percorso. La letteratura sull'analisi curriculare si occupa del comportamento osservato degli studenti; mentre l'obiettivo qui perseguito è la ricostruzione delle relazioni di dipendenza tra i contenuti degli insegnamenti, a prescindere dall'ordine in cui essi vengono di fatto sostenuti.

Non si tratta di descrivere un processo osservato, ma di elicitare il modello che dovrebbe governarlo. È un problema complementare a quelli censiti dalla rassegna, che richiede una fonte informativa diversa: i syllabus, anziché i registri di carriera e, di conseguenza, strumenti di analisi diversi, che saranno oggetto del paragrafo successivo.

[11] Calegari, D., Delgado, A. (2025). *A Systematic Review on Process Mining for Curricular Analysis*. WIREs Data Mining and Knowledge Discovery, 15. DOI: 10.1002/widm.70005.

I.2 Large Language Models

I.2.1 Architettura e capacità

I Large Language Models (LLM) sono modelli linguistici di grandi dimensioni basati sull'architettura Transformer[12]. Tale tipologia consente di superare uno dei limiti delle architetture tradizionali basate su reti neurali *ricorrenti* (RNN/LSTM) o *convoluzionali*, con struttura encoder-decoder, le quali elaborano le parole in sequenza impedendo la parallelizzazione e rendendo l'addestramento lento, soprattutto su frasi lunghe.

Tale limite viene superato introducendo un'architettura basata unicamente su meccanismi di *attention*, che permettono al modello di mettere in relazione qualsiasi coppia di parole in una frase, indipendentemente dalla loro distanza, e di farlo in parallelo. Nello specifico, il cuore del meccanismo prende il nome di *self-attention*: ogni parola "guarda" tutte le altre per costruire una rappresentazione che tenga conto del contesto.

Per quanto riguarda invece l'addestramento di un LLM, esso avviene in due fasi principali: nella prima, il modello apprende da grandi quantità di testo, acquisendo competenza linguistica e conoscenza generale; nella seconda viene addestrato a seguire istruzioni formulate in linguaggio naturale ed è quest'ultima proprietà a

[12] Vaswani, A. et al. (2017). *Attention Is All You Need*. Advances in Neural Information Processing Systems 30. arXiv:1706.03762

diventare la principale leva di controllo sul comportamento del modello. I syllabus sono testi eterogenei per struttura, lunghezza e livello di dettaglio, e non si prestano a essere elaborati con schemi di estrazione predefiniti. Un LLM può invece interpretarli così come sono, sulla base di istruzioni espresse a parole.

I.2.2 La generazione come processo stocastico

Per generare un testo, un LLM assegna a ogni passo una probabilità a ciascuna delle possibili continuazioni e il testo viene costruito scegliendo iterativamente tra di esse. La regola con cui avviene questa scelta, detta strategia di decodifica, è configurabile: si può selezionare sempre la continuazione più probabile (decodifica greedy) oppure campionare tra le alternative, regolandone l'ampiezza tramite parametri come la *temperature.* Il campionamento serve a evitare che il testo generato risulti ripetitivo[13] ma, allo stesso tempo, comporta una conseguenza rilevante ai fini di questo lavoro: introduce una componente stocastica che fa sì che, a parità di modello e input, esecuzioni ripetute producano risultati diversi

Il problema si estende oltre il campionamento: anche azzerando *la temperature*, l'inferenza non risulta comunque riproducibile nella pratica. In un esperimento

[13] Holtzman, A., Buys, J., Du, L., Forbes, M., Choi, Y. (2020). *The Curious Case of Neural Text Degeneration*. International Conference on Learning Representations (ICLR). arXiv:1904.09751

condotto da He e Thinking Machines Lab (2025), mille generazioni ottenute a temperature zero dallo stesso prompt hanno prodotto ottanta risultati distinti[14].

Ne segue che l'esito di una singola esecuzione non costituisce, di per sé, una misura affidabile della qualità di una metodologia basata su LLM. Per questo motivo è stata scelta un'impostazione sperimentale fondata sull'esecuzione ripetuta e sull'aggregazione dei risultati, che verrà discussa nei capitoli successivi.

I.2.3 Modelli open-weights ed esecuzione locale

I m*odelli open-weights* sono modelli i cui parametri vengono resi pubblici e scaricabili sotto una licenza che ne consente l'uso, la modifica e la ridistribuzione. Essi si distinguono dai modelli proprietari (come GPT-4 o Claude), accessibili attraverso interfacce remote (API). Il modello impiegato qui, gpt-oss:20b, appartiene alla prima categoria ed è stato rilasciato con licenza Apache 2.0[15].

La disponibilità dei parametri non è però di per sé sufficiente all'utilizzo: occorre un ambiente di esecuzione che ne gestisca il caricamento in memoria e ne esponga le funzionalità attraverso un'interfaccia programmatica. Nel presente lavoro tale ruolo è svolto da Ollama.

[14] He, H. e Thinking Machines Lab (2025). Defeating Nondeterminism in LLM Inference. Thinking Machines Lab: Connectionism. DOI: 10.64434/tml.20250910.

[15] OpenAI (2025). *gpt-oss-120b & gpt-oss-20b Model Card*. arXiv:2508.10925

Rispetto all'accesso remoto, l'esecuzione locale modifica il rapporto tra utilizzatore e modello sotto tre aspetti rilevanti per la metodologia. In primo luogo, la versione del modello è fissata: i pesi non vengono aggiornati se non per scelta esplicita, il che mantiene stabile nel tempo l'ambiente di esecuzione e ne sottrae così una possibile fonte di variabilità (a differenza dei servizi remoti, dove il modello può essere aggiornato senza preavviso). In secondo luogo, i parametri dell'inferenza (strategia di decodifica, temperatura, ampiezza del contesto, seme di generazione) sono direttamente accessibili e documentabili, mentre nei servizi remoti sono in parte imposti dal fornitore.

La contropartita è costituita dalle risorse disponibili. Un modello eseguibile su una singola macchina ha necessariamente dimensioni inferiori rispetto ai modelli erogati su infrastrutture dedicate, con conseguenze sulla capacità di ragionamento e sull'ampiezza del contesto gestibile in una singola richiesta.

I.3 Tecniche di prompting per task strutturati

Poiché il comportamento di un LLM è guidato dalle istruzioni che riceve, il modo in cui esse vengono formulate incide direttamente sulla qualità dell’output. La progettazione di queste istruzioni, nota come *prompt engineering*, è particolarmente importante quando all'LLM si chiede non una risposta libera, ma un output strutturato ottenuto attraverso un ragionamento articolato. Questo paragrafo

presenta le tecniche rilevanti per il presente lavoro, riprendendo quelle su cui si fonda la metodologia di riferimento *Diamantini et al.* e introducendo quelle che ne motivano l'estensione.

I.3.1 Prompting per il ragionamento

Il *chain-of-thought prompting* consiste nell'indurre il modello a produrre una serie di passaggi di ragionamento intermedi, al fine di perfezionare il suo processo e ottenere un risultato più coerente e, come effetto collaterale utile, rende il ragionamento ispezionabile[16]. Nel nostro contesto questo significa non solo ottenere una dipendenza A→B, ma fare una serie di step che portano ad essa.

Una seconda tecnica complementare è lo *step-back prompting*: prima di affrontare un caso specifico, si chiede al modello di risalire e astrarre i principi generali e teorici, da usare poi come guida per la risposta[17]. Ciò aiuta a evitare conclusioni errate o allucinazioni. Infine, un terzo approccio riguarda l'organizzazione del compito. Con il *pairwise prompting* il problema viene scomposto su singole coppie di elementi, nel nostro caso: «il corso A è prerequisito del corso B?».

---

[16] Wei, J. et al. (2022). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. Advances in Neural Information Processing Systems 35 (NeurIPS). arXiv:2201.11903.

[17] Zheng, H.S. et al. (2024). *Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models*. International Conference on Learning Representations (ICLR). arXiv:2310.06117.

I.4 Stato dell'arte

Gli LLM si sono dimostrati capaci di riconoscere relazioni di dipendenza contenute in un testo, esplicite ed implicite, con prestazioni pari o superiori a quelle dei metodi specializzati sviluppati per lo stesso scopo[18]. Il presupposto degli algoritmi classici di causal discovery è però la disponibilità di dati osservativi sufficienti e rappresentativi del fenomeno, di cui studiarne le regolarità statistiche per dedurre quali variabili influenzino le altre. Un LLM procede in modo differente: sfrutta la conoscenza acquisita dai testi durante l'addestramento.

Come rilevano gli stessi autori dello studio citato sopra, il modello «ignora i dati» e riconosce le dipendenze sulla base di ciò che i testi affermano o lasciano intendere, proprio come nel caso qui affrontato. Non esistendo dati osservativi che catturino la dipendenza concettuale tra corsi, l'informazione necessaria è contenuta nei syllabus sotto forma di testo: è questa la conoscenza su cui un LLM opera.

L'estrazione automatica di relazioni di propedeuticità in ambito educativo non è un ambito nuovo, costituisce anzi una linea di ricerca ben consolidata. La maggior parte dei lavori opera a livello di singoli concetti o argomenti, stabilendo ad

[18] Kıcıman, E., Ness, R., Sharma, A., Tan, C. (2023). *Causal Reasoning and Large Language Models: Opening a New Frontier for Causality*. Transactions on Machine Learning Research. arXiv:2305.00050. (La citazione «il modello ignora i dati» traduce *"LLMs ignore the actual data"* dell'abstract.)

esempio, quale nozione vada appresa prima e ricava tali relazioni da libri di testo, materiali didattici o piattaforme MOOC, con metodi che spaziano da misure non supervisionate a modelli di apprendimento automatico[19]. Più recentemente, anche gli LLM sono stati impiegati per estrarre concetti e relazioni educative da testi non strutturati come i programmi dei corsi[20].

Rispetto a questo panorama, la linea in cui si inserisce il presente lavoro è più circoscritta. In questo caso, la scoperta delle propedeuticità viene trattata come un problema di *elicitation* di un modello di processo, il cui esito è un grafo di dipendenze fra interi insegnamenti anziché fra singoli concetti. Inoltre, il presente studio adotta un LLM open-weights eseguito in locale e riporta risultati basati su molteplici esecuzioni.

[19] Si vedano, ad esempio, Liang, C., Wu, Z., Huang, W., Giles, C.L. (2015). *Measuring Prerequisite Relations Among Concepts*. Proceedings of EMNLP 2015, pp. 1668–1674 (metrica non supervisionata RefD); e Pan, L., Li, C., Li, J., Tang, J. (2017). *Prerequisite Relation Learning for Concepts in MOOCs*. Proceedings of ACL 2017, pp. 1447–1456.

[20] Yang, T., Ren, B., Gu, C., He, T., Ma, B., Konomi, S. (2025). *Leveraging LLMs for Automated Extraction and Structuring of Educational Concepts and Relationships*. Machine Learning and Knowledge Extraction, 7(3), 103.

I.5 Baseline Metodologica

Il presente lavoro si appoggia e prende le mosse da due contributi: l'articolo *"Leveraging LLMs to Discover Causal Dependencies: A Case Study on a University Program"* di Claudia Diamantini, Chiara Gobbi, Alessandro Mele, Domenico Potena e Cristina Rossetti, docenti e dottorandi del Dipartimento di Ingegneria dell'Informazione dell'UNIVPM di Ancona, e la Tesi di Marika Salvatori *"Definizione di un grafo di dipendenze tramite LLM, con applicazione a corsi universitari",* che sviluppa e approfondisce il metodo.

Questo capitolo li descrive e ne mette a fuoco i concetti portanti, così da rendere chiaro che cosa è stato ripreso e che cosa è stato cambiato.

I.5.1 L'articolo di riferimento

L'articolo si pone come base di partenza per esplorare e sviluppare una metodologia in grado di ricostruire, a partire dai syllabus dei vari corsi, le dipendenze di propedeuticità tra insegnamenti e rappresentarle come un Directly-Follows Graph. È da qui che si riprendono le metodologie di step-back prompting e chain-of-thought, nonché l'approccio pairwise, introdotte in precedenza.

Il modello qui impiegato è di tipo proprietario e cloud (gpt-o1-preview, accessibile tramite le API di OpenAI).

La metodologia si articola in tre fasi principali.

Nella prima, chiamata *Syllabus Summarization*, si assegna a ogni corso un codice neutro, che serve a impedire al modello di farsi guidare dal nome dell'insegnamento anziché dal suo contenuto, e si chiede di riassumere i concetti principali di ciascun corso, costruendo una base di conoscenza a cui attingere in seguito. Successivamente, nella *Dependencies Extraction*, viene fornita la definizione di causal dependency tra corsi: *"A causal dependency between courses A and B (A → B) means that in order to attend B, some topics covered in course A are required."* e su questa base vengono richiesti tre sotto-step: derivare un insieme iniziale di dipendenze, motivarne la scelta e rivedere le scelte ed eventualmente modificarle, ottenendo così il set finale.

Infine, nella terza ed ultima fase, il risultato viene trasformato in un DFG, al quale si aggiungono due nodi artificiali, START ed END, per collegare anche i corsi privi di dipendenze in ingresso o in uscita e ottenere così un grafo connesso.

Il grafo prodotto viene poi confrontato con un grafo di riferimento che funge da base di verità (*Ground Truth*[21]): è la rappresentazione ufficiale delle propedeuticità, ottenuta chiedendo a ciascun docente quali insegnamenti andrebbero seguiti prima del proprio. È rispetto ad esso che si misura la qualità del grafo del modello.

[21]L'insieme di informazioni reali, verificate e considerate corrette che serve come termine di paragone per addestrare e valutare i modelli di intelligenza artificiale

Tuttavia, occorre sottolineare che non si tratta di una verità infallibile poiché nasce da un giudizio umano e soggettivo: ogni docente conosce bene il proprio corso e solo in parte gli altri, omettendo a volte legami per semplicità.

Il risultato principale ottenuto, riporta l'articolo, è che il modello riesce a individuare le dipendenze reali con buona affidabilità, ma ne trascura molte. Da questa conclusione, prende le mosse il precedente lavoro di tesi.

I.5.2 L'estensione della tesi precedente

La tesi di Salvatori parte dallo stesso problema e dallo stesso caso di studio, ma rielabora la pipeline in una forma più controllata e interpretabile, con l'obiettivo di migliorare la qualità del grafo finale.

La pipeline si articola in 4 step:

Il primo passaggio vede la definizione del ruolo dell'LLM («educational curriculum expert») e del suo obiettivo. Gli si forniscono anche il formato di input e la lista completa dei codici dei corsi esistenti, così che non possa inventarne di nuovi.

La pipeline vera e propria si compone poi di quattro step.

STEP 0: il modello elenca, per ogni corso, gli argomenti e le competenze che quell'insegnamento fornisce, espliciti ed impliciti;

STEP 1: fa lo stesso con la conoscenza richiesta, cioè i prerequisiti necessari per seguirlo; STEP 2, avviene l'inferenza vera e propria: il modello stabilisce una

dipendenza A→B quando ciò che A fornisce copre, anche solo in parte, ciò che B richiede. A ogni relazione associa inoltre un peso $p \in [0,1]$, dove 0 significa che non c'è nessuna dipendenza concettuale e 1 invece è massima, più una breve motivazione. Infine, nello STEP 3, gli si chiede una revisione complessiva per correggere eventuali incoerenze.

A questa fase di generazione segue la valutazione. Qui ogni dipendenza segue una serie di filtri a cascata: per primo l'applicazione di una soglia su p, per scartare i legami deboli; poi la rimozione delle simmetrie, cioè il caso particolare di una coppia con archi in entrambe le direzioni, di cui si mantiene solo il verso con peso maggiore. Con lo stesso criterio, infine, si spezzano i cicli, cioè le situazioni in cui, seguendo la direzione degli archi, si potrebbe partire da un corso e tornarvi (come in A→B→C→A), il che implicherebbe che un corso sia prerequisito di sé stesso; per ogni ciclo si rimuove l'arco di peso minore. Al termine di questi passaggi si ottiene un grafo diretto aciclico su cui si misurano le metriche.

I.5.3 Valutazione: metriche e struttura del grafo

Entrambi i lavori valutano la qualità del risultato confrontando il grafo prodotto dall'LLM con il grafo dei docenti. Il confronto avviene tra archi orientati, cioè A->B è diverso da B->A, e le dipendenze si distinguono in tre gruppi principali.

I veri positivi (*True Positive,* TP) sono gli archi prodotti dal modello che compaiono anche nel grafo dei docenti. Sono le dipendenze individuate correttamente e, a parità di errori, più sono migliore è la ricostruzione del modello.

I falsi positivi (*False Positive*, FP) sono gli archi prodotti dal modello ma assenti nel riferimento. Non sempre però un falso positivo è un errore: può infatti essere dovuto ad una *over-generation*, quando il modello produce dipendenze in eccesso, inventando e quindi sbagliando. Può però trattarsi di una relazione reale, che i docenti non avevano individuato o formalizzato. Per questo una volta ottenuto il grafo conviene sempre effettuare un'analisi qualitativa delle dipendenze prodotte.

Infine, nei falsi negativi (False Negatives, FN), rientrano le dipendenze del grafo dei docenti non trovate dal modello, le quali rappresentano una mancanza.

Da questi tre conteggi si ricavano le metriche di qualità: la *Precision* rappresenta la quota di archi prodotti che sono corretti e misura l'affidabilità del modello. Usando le notazioni prima elencate si calcola come: $P = \frac{TP}{TP+FP}$; il *Recall* indica quante delle dipendenze vere sono state trovate e si misura come: $Re = \frac{TP}{TP+FN}$.

Le due metriche tendono però a essere in tensione, alzare l'una spesso abbassa l'altra. Per evitare quindi di dare erroneamente più peso a una o all'altra, si sintetizzano nella loro *media armonica, l'F1*, che le riassume in un unico numero bilanciandole. In formula: $F1 = 2\ x\ \frac{Precision \times Recall}{Precision+Recall}$.

Il grafo dei docenti è il riferimento più autorevole a disposizione, ma va interpretato con cautela: come già detto non è un ground truth assoluto e su questo entrambi i lavori insistono, in quanto un arco «sbagliato» rispetto al riferimento non è necessariamente un errore.

Da qui la scelta di inserire una categoria particolare: le dipendenze *indirette*. Si ha una dipendenza indiretta quando il modello produce un arco A→C che nel riferimento non compare come collegamento diretto ma che vi si ritrova sotto forma di cammino: si arriva da A a C passando per uno o più corsi intermedi (come, ad esempio, A→B→C). In tali casi la dipendenza esiste davvero, solo in forma mediata e contarla come errore sarebbe ingiusto. Per questo tali archi si contano a parte, come corretti anziché come falsi positivi. Sul piano delle metriche la scelta alza la precision, perché quegli archi non pesano più come errori, e lascia il recall invariato, dato che non sono dipendenze del riferimento da recuperare. Ne risulta una lettura più equa, che non classifica come sbagliate relazioni che sono in realtà scorciatoie coerenti e legittime.

Sul piano strutturale, due proprietà del grafo vengono prese in considerazione prima della valutazione: l'aciclicità e la ridondanza transitiva.

La prima richiede che il grafo finale sia un DAG, cioè un grafo privo di cicli.

La seconda riguarda invece gli archi indiretti, ma con una sfumatura diversa. Se valgono A→B e B→C, l'arco A→C è già implicito nella catena e non aggiunge

informazione. La riduzione transitiva è l'operazione che rimuove questi archi ridondanti, lasciando il grafo minimo che conserva le stesse relazioni di raggiungibilità[22].

Conviene qui tenere distinte due nozioni che ruotano attorno alla transitività ma agiscono in senso opposto. La dipendenza indiretta è un arco del modello giustificato da un cammino nel grafo dei docenti e quindi va mantenuto; l'arco transitivo ridondante è invece giustificato da un cammino interno al modello e per questo si può rimuovere. Se poi questa potatura, nel contesto locale, aiuti davvero o faccia perdere informazione è una delle domande riprese nella parte sperimentale.

[22]Aho, A.V., Garey, M.R., Ullman, J.D. (1972). The Transitive Reduction of a Directed Graph. SIAM Journal on Computing, 1(2), 131–137. DOI: 10.1137/0201008

CAPITOLO II

METODOLOGIA PROPOSTA

Questo capitolo presenta la metodologia proposta nel presente lavoro, ripresa dall'approccio di riferimento e adattata al contesto locale. Nel passaggio a un LLM locale emergono limiti specifici, per ciascuno dei quali si descrive l'intervento adottato per contenerli. In particolare, si interviene sulla forma dell'input, sulla struttura della pipeline e sul prompt, e si introduce una strategia di esecuzione ripetuta e aggregazione pensata per contenere la variabilità tipica di un modello locale. In tale sede si definiscono il metodo e il perché di certe scelte, mentre la loro verifica empirica, con i risultati che le giustificano, è rimandata ai capitoli IV e V.

II.1 Preparazione dell'input: la segmentazione dei syllabus

I programmi dei corsi hanno la caratteristica di essere fortemente eterogenei per lunghezza, formato di scrittura e livello di dettaglio, al punto che alcuni si presentano come un elenco essenziale di argomenti e altri come una descrizione discorsiva che intreccia contenuti, obiettivi e modalità didattiche. In un contesto locale questa disomogeneità pesa più di quanto farebbe su un modello di frontiera in cloud, il quale dispone di una capacità sufficiente ad assorbire da sé le irregolarità di formato. Un modello di piccole dimensioni è invece meno robusto rispetto alla

forma superficiale dell'input e fatica a riconoscere che un blocco di prosa discorsiva e un elenco puntato veicolano lo stesso tipo di informazione. È una dinamica generale dei modelli di piccola taglia: la capacità di gestire in autonomia un compito - qui, normalizzare internamente un input disomogeneo - cresce con la dimensione del modello[23], per cui ciò che un modello grande risolve da sé conviene fornirlo già risolto a uno più piccolo.

Da qui l'intervento adottato, che indicheremo con il termine *segmentazione*[24]: trasformare ogni programma da prosa continua a una lista esplicita di argomenti, scritti sotto forma di elenco puntato.

Tale scelta trova riscontro nelle recenti pipeline di analisi dei curricula, che non danno in pasto al modello i syllabus grezzi ma li riconducono a uno schema comune proprio perché i dati curricolari sono in larga parte non strutturati, con relazioni di

[23] Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., Fedus, W. (2022). *Emergent Abilities of Large Language Models*. Transactions on Machine Learning Research. arXiv:2206.07682.

[24] Nel *natural language processing*, per *segmentazione* si intende la suddivisione di un testo continuo in unità significative (parole, frasi o blocchi tematici), sulla base di elementi come la punteggiatura; è uno dei passaggi del più generale processo di *normalizzazione del testo,* cioè ricondurre un testo a una forma più conveniente e standard, di cui fanno parte anche la normalizzazione lessicale (lemmatizzazione, *stemming*) e la segmentazione delle frasi. Cfr. Jurafsky, D., Martin, J. H. (2024). *Speech and Language Processing 3ª ed. (bozza), cap. 2.*

propedeuticità spesso non documentate o definite in modo incoerente tra corsi e istituzioni[25]. Rispetto a quelle pipeline, che riorganizzano ogni corso in una scheda a campi predefiniti, l'intervento qui adottato è volutamente leggero e conservativo: non si costruisce alcuna struttura nuova ma ci si limita a rendere esplicita la suddivisione in argomenti che il testo del programma già contiene in forma implicita, spezzando unicamente sui separatori già presenti (il punto, il punto e virgola e il trattino usato come marcatore di elenco).

Due accorgimenti garantiscono che la segmentazione sia pulita:

1) i punti decimali interni a espressioni come «(0.5 ects)», vengono riconosciuti e tenuti fuori dalla divisione, così da non essere scambiati per un punto di fine frase.

2) i trattini interni a un termine tecnico (*skew-symmetric*, *2-port*) vengono preservati, così da non spezzare la parola e rischiare di perderne il significato.

L'elemento essenziale è che l'intervento conserva integralmente il contenuto: rende esplicita la struttura senza rimuovere né riformulare alcun termine.

A titolo di esempio, si riporta il programma di Analisi Matematica 2 (BAC), originariamente in forma discorsiva: "*functions of several variables. curves, line*

[25] Awasthi, R., Shukla, A., Agrawal, A. K., Dubey, P., Ramasamy, R. K. (2026). *A Hybrid Transformer–Graph Framework for Curriculum Sequencing and Prerequisite Optimization in Computer Science Education with Explainable AI*. Algorithms, 19(4), 308. DOI: 10.3390/a19040308.

integrals, vector fields, differential forms. multiple integrals with applications. ordinary differential equations. laplace transform on r."

Viene trasformato nel seguente elenco, che ne conserva integralmente i contenuti:

- *functions of several variables*
- *curves, line integrals, vector fields, differential forms*
- *multiple integrals with applications*
- *ordinary differential equations*
- *laplace transform on r*

L'elenco così ottenuto viene inserito verbatim nel prompt.

Sull'intero dataset la segmentazione aggiunge appena l'1,4% di caratteri, risultando quindi praticamente gratuita in termini di lunghezza dell'input. Ne risultano circa cinquecento righe complessive, in media 16 per corso, da un minimo di una (per i pochi programmi scritti come unico blocco privo di separatori) a oltre cinquanta.

II.2 La pipeline adattata: rimozione dello Step di revisione

La pipeline di riferimento, descritta nel dettaglio nel I.5.2, prevede come ultimo passaggio una revisione complessiva finale, pensata per sanare le incoerenze residue (STEP 3). Nel contesto locale, quest'ultima fase si rivela controproducente: non corregge le incoerenze che dovrebbe risolvere e tende anzi a rimuovere dipendenze corrette, oltre a introdurre instabilità.

Del resto, tale comportamento trova riscontro nella letteratura sull'auto-correzione degli LLM[26], secondo cui, in assenza di un riscontro esterno che segnali quali parti della risposta siano errate, la revisione autonoma che un modello opera sul proprio output tende a non migliorarne e talvolta a peggiorarne, la qualità.

Per questa ragione la metodologia proposta rimuove lo STEP 3 e l'output finale della pipeline coincide con la lista di dipendenze prodotta dallo STEP 2.

La diagnosi puntuale e quantificata che motiva tale scelta è ripresa nel Capitolo IV.

II.3 La progettazione del prompt dello STEP 2

Semplificare la pipeline rende il comportamento del modello più affidabile ma non agisce su un altro problema emerso dal contesto locale: l'over-generation.

La tendenza del modello a proporre molte più dipendenze di quante il riferimento ne preveda si concentra nello STEP 2, dove il modello elenca le coppie A → B. È dunque sul prompt di questo passaggio che interviene il prossimo accorgimento.

L'idea è di guidare il modello inserendo alcuni esempi direttamente nell'istruzione. Si tratta di una modalità consolidata nei modelli linguistici, conosciuta come "l'apprendimento in contesto" (*in-context learning*), secondo cui un modello sufficientemente capace può apprendere il comportamento atteso da poche

[26] Huang et al. (2023), *Large Language Models Cannot Self-Correct Reasoning Yet*, ICLR 2024, arXiv:2310.01798.

dimostrazioni collocate nel prompt, senza alcun ri-addestramento[27]. Di solito questi esempi sono di tipo positivo, cioè casi che illustrano la risposta desiderata. Qui però il modello tende a individuare legami anche dove non esistono e gli si deve perciò insegnare a riconoscere quando due corsi non sono legati da alcuna propedeuticità, così da non generare in quel caso alcun arco.

A tale scopo il prompt dello STEP 2 è stato arricchito con tre indicazioni e una coppia di esempi negativi.

La prima è una regola di omissione, che vieta di produrre un arco A → B quando l'unico legame è un argomento o una parola chiave in comune o uno stesso sfondo generico, in assenza di una conoscenza specifica che A fornisca e B richieda.

Segue un chiarimento di segno opposto, indispensabile per non pagare in recall il guadagno in precision: una dipendenza resta legittima quando A fornisce contenuti sostanziali su cui B si fonda, anche se i due corsi appartengono alla stessa area. Senza questa precisazione la regola di omissione rischierebbe di essere interpretata in modo troppo aggressivo, portando il modello a scartare anche prerequisiti reali.

La terza indicazione autorizza esplicitamente l'insieme vuoto: se un corso non ha alcun vero prerequisito tra gli altri, il modello non deve restituire nulla per quel

[27] Brown, T. B., Mann, B., Ryder, N., et al. (2020). *Language Models are Few-Shot Learners*. NeurIPS 2020. arXiv:2005.14165.

corso, poiché altrimenti tende a «riempire», forzando almeno un arco per ciascuno. Chiudono il blocco due casi da non produrre, entrambi costruiti su codici fittizi e su materie generiche, in quanto impiegare come esempio una coppia reale del dataset significherebbe iniettare nel prompt un'informazione derivata dal grafo di riferimento che il modello dovrebbe invece ricostruire da sé.

Il prompt che ne risulta, adottato nella configurazione della pipeline, è riportato integralmente di seguito (in lingua originale inglese). La pipeline è strutturata come una conversazione multi-turno: un messaggio iniziale che definisce ruolo, obiettivo e formato di input, seguito dai tre passaggi STEP 0, STEP 1 e STEP 2:

You are an educational curriculum expert.
Your task is to analyze a Bachelor's Degree in Information and Automation Engineering and extract both explicit and implicit knowledge dependencies among its courses, based on their program descriptions.

Input format:
<subject>
code: [COURSE CODE]
program: [Course description]
</subject>

The valid course codes are: [lista dei codici validi].

STEP 0 – Identify Knowledge Provided

For each course, list the knowledge it provides (e.g., theory, laws, formulas, tools).

Example of output format for the course "Mathematical Analysis II" with code XYZ:
"XYZ provides:functions of several variables, partial derivatives, gradient and Jacobian, multiple integrals, line and surface integrals, vector fields, differential forms, ordinary differential equations, Laplace transform."

STEP 1 – Identify Required Knowledge

For each course, list the knowledge required to understand it, both explicit and implicit prerequisite knowledge.

Example of output format for the course "Mathematical Analysis II" with code XYZ: "XYZ requires:knowledge of limits, derivatives, continuity, basic integral calculus, and introductory linear algebra (vectors, matrices, determinants)"

STEP 2 – Discover Dependencies (A -> B)
For each course B, identify the knowledge dependencies from other courses to B. A knowledge dependency A -> B exists when course A provides knowledge required to understand course B.

Assign a dependency weight p ∈ [0, 1], where:
p = 0 → no conceptual dependency
p = 1 → full conceptual dependency

Give your answer as a plain list with one dependency per line, in exactly this format:
AAA -> BBB |p=0.NN|
Example: JKL -> XYZ |p=0.85|
Use the arrow "->" and write the weight as |p=0.NN|. No tables, no Markdown, no preamble or closing remarks.

Important - avoid spurious dependencies:
Only output A -> B when A provides knowledge that is genuinely needed to understand B. Do NOT create a dependency when the ONLY connection is a single shared topic, a shared keyword, or the same generic background (e.g. both using basic mathematics), without A providing specific knowledge required by B. A dependency IS warranted when A provides substantive content that B builds upon, even if the two courses belong to the same area.
If a course B has no genuine prerequisites among the others, output nothing for B.

Negative examples (these must NOT be output):
- PQR and STU both use basic mathematics, but neither provides knowledge specifically required by the other -> no dependency.
- VWX and YZA share the single keyword "geometry", but one shared keyword alone is not a conceptual prerequisite -> no dependency.

Fig. 1: Estratto del prompt impiegato nella pipeline

II.4 Il peso p e il filtraggio delle dipendenze

Dalla pipeline di riferimento la metodologia eredita il peso $p \in [0,1]$ che il modello associa a ciascuna dipendenza. Si tratta di una confidenza che gradua la forza di ciascun legame: valori vicini a 1 indicano una forte corrispondenza tra conoscenza fornita e richiesta; valori intermedi indicano una dipendenza che contribuisce alla comprensione ma non è indispensabile ed infine valori bassi segnalano relazioni probabilmente basate su semplici somiglianze terminologiche.

Su di esso si fondano alcuni dei filtri applicati in fase di valutazione. Primo fra tutti, una soglia su p che scarta i legami più deboli, seguita dalla rimozione delle coppie simmetriche e dalla rottura dei cicli, per le quali si conserva di volta in volta il verso di peso maggiore. In linea di principio, poiché l'output di un modello locale è rumoroso, un indice di confidenza consente di trattenere le dipendenze più solide e scartare quelle spurie.

Tuttavia, conviene anticipare, che il funzionamento di questo filtro in un modello debole non è scontato. Il punteggio gradua la forza del legame nella direzione che il modello dichiara, ma nulla garantisce che quella direzione sia corretta. In alcuni casi ricorrenti, come si vedrà, il modello coglie la relazione giusta ma la orienta al contrario. Se e quanto p rifletta il verso di una dipendenza, e non solo la sua intensità, è una delle domande a cui la parte sperimentale darà risposta.

II.5 La valutazione singola e multipla

L'ultimo elemento, che più profondamente ha orientato l'impianto sperimentale, è l'inaffidabilità del singolo run. A parità di input, esecuzioni diverse producono grafi diversi, talvolta anche di molto. A causa di questa variabilità l'esito di una singola esecuzione non costituisce una misura affidabile della qualità del metodo e, per questo, la valutazione non si fonda mai su un run isolato, ma sull'esecuzione ripetuta e sull'aggregazione dei risultati.

Distinguiamo due modalità di valutazione del grafo prodotto rispetto al grafo dei docenti (le cui metriche sono definite nel I.5.3):

1) *Valutazione per singola esecuzione*: ogni run produce un proprio grafo, che viene confrontato singolarmente con il riferimento. Dovendo però eseguire più run, la prestazione riferita a un insieme di esecuzioni è la sintesi statistica (media e deviazione standard) dei valori ottenuti su ciascun grafo.
2) *Valutazione per ensemble*: i grafi delle diverse esecuzioni vengono prima aggregati in un unico grafo, il quale viene poi confrontato con il riferimento, ottenendo così un'unica misura di prestazione.

È questa seconda modalità a costituire il cuore del metodo: aggregare molte esecuzioni permette di ricomporre un'informazione che nessun singolo run possiede per intero. Le tecniche con cui i grafi vengono aggregati e le prove che ne misurano l'efficacia sono sviluppate nel Capitolo V.

CAPITOLO III

CASO DI STUDIO E SETUP SPERIMENTALE

III.1 Il corso di laurea in Ingegneria Informatica e dell'Automazione

Il caso di studio oggetto del presente lavoro è il Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione dell'Università Politecnica delle Marche (UNIVPM)[28]. Il corso è distribuito su tre anni accademici, secondo una progressione che muove dalle discipline di base verso quelle specialistiche.

Al primo anno si trovano prevalentemente insegnamenti fondamentali quali matematica, fisica ed informatica, che forniscono i concetti necessari ad affrontare i corsi successivi; il secondo anno introduce le materie caratterizzanti nelle aree dell'automazione, dell'elettronica e dell'elettrotecnica; il terzo anno, infine, è dedicato alla specializzazione, con corsi avanzati e un'ampia quota di discipline a scelta. In tutto il curriculum si compone di 39 insegnamenti, tra corsi obbligatori, caratterizzanti e a scelta dello studente.

[28] Si tratta dello stesso corso adottato nel paper di Diamantini et al. (2025) e del precedente lavoro di tesi di Salvatori.

Di seguito si riporta la struttura completa del piano di studi, con l'indicazione di anno, semestre e crediti di ciascun insegnamento:

SSD	Ciclo	Insegnamento	CFU
Anno 1 (TOT 54 CFU)			
FIS/01	I	Fisica Generale 1	9
MAT/03	I	Algebra Lineare e Geometria	6
MAT/05	I	Analisi Matematica 1	9
–	I	Lingua Straniera	3
FIS/01	II	Fisica Generale 2	6
ING-INF/05	II	Fondamenti di Informatica	9
MAT/05	II	Analisi Matematica 2	6
SECS-P/06	II	Economia dell'Impresa	6
Anno 2 (TOT 66 CFU)			
–		Corso/i a scelta	9
ING-INF/04	I	Fondamenti di Automatica	9
ING-INF/05	I	Programmazione ad Oggetti	9
ING-IND/31	I	Elettrotecnica	6
ING-INF/01	II	Elementi di Elettronica	9
ING-INF/02	II	Elettromagnetismo per Trasmissione Informazione	9
ING-INF/04	II	Controlli Automatici	9
–		1 insegnamento a scelta per un totale di 6 CFU: MAT/06 – Calcolo probabilità e statistica matematica MAT/02 – Algebra e Logica MAT/07 – Meccanica Razionale MAT/08 – Analisi Numerica	6
Anno 3 (TOT 60 CFU)			
–		Corso/i a scelta	—
–		Prova Finale	—
–		Tirocinio	—

ING-INF/04		Tecnologie per l'Automazione e la Robotica	9
MAT/09		Ricerca Operativa	9
–		1 insegnamento a scelta per un totale di 9 CFU: ING-INF/05 – Basi di Dati ING-INF/05 – Sistemi Operativi	9
–		2 insegnamenti a scelta per un totale di 18 CFU: ING-INF/04 – Automazione Industriale ING-INF/04 – Laboratorio di Automazione ING-INF/04 – Metodi e Tecniche per l'Automazione ING-INF/04 – Modellistica e Identificazione dei Processi Dinamici ING-INF/04 – Progettazione Assistita da Calcolatore dei Sistemi di Controllo ING-INF/05 – Architetture dei Calcolatori e Cloud Computing ING-INF/05 – Programmazione Mobile ING-INF/05 – Tecnologie Web	18

Offerta a scelta libera dello studente (OL) per i corsi a scelta

SSD	Ciclo	Insegnamento	CFU	Anno
ING-INF/03	I	Tecnologie per le Telecomunicazioni	9	3
ING-INF/04	II	Modellistica e Identificazione dei Processi Dinamici	9	3
ING-INF/05	I	Basi di Dati	9	3
ING-INF/05	II	Ingegneria del Software	9	2
ING-INF/05	II	Sistemi Operativi	9	3
MAT/05	I	Metodi Matematici per Ingegneria dell'Automazione	9	2

Tab. 1: Piano formativo del Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione dell'Università Politecnica delle Marche (UNIVPM), Anno accademico 2023-2024.

III.2 Dataset dei syllabus e formato di input

Per ciascun insegnamento il dato utilizzato come input è il programma dei contenuti, cioè la descrizione degli argomenti trattati nel corso, riportata in lingua inglese. Per questo motivo dal dataset di analisi sono state escluse le attività formative il cui syllabus non riporta il programma degli argomenti trattati (la prova finale, il tirocinio, la lingua straniera e a scelta generici).

L'insieme così ottenuto si compone di 32 insegnamenti, a ciascuno dei quali è assegnato un codice neutro (da BAA a BBH) che ne sostituisce il nome: come già osservato nel Capitolo I, questo accorgimento impedisce al modello di lasciarsi guidare dal titolo dell'insegnamento anziché dal suo contenuto effettivo.

I programmi dei corsi sono stati reperiti dalla guida ufficiale agli insegnamenti dell'Ateneo[29]. Di ciascuna scheda è stata conservata la sola descrizione dei contenuti, con gli argomenti e i moduli tematici trattati, tralasciando gli elementi di natura amministrativa, come l'anno di erogazione, per evitare che il modello ricostruisse le dipendenze sulla base dell'organizzazione del piano di studi anziché delle effettive relazioni tra i contenuti dei corsi.

[29] https://guide.univpm.it/guide.php?lang=langita&fac=ingegneria&aa=2026&cds=IT04&cerca=cerca

Ogni voce del dataset è così costituita da tre campi: il nome dell'insegnamento (*course*), il codice neutro assegnato (*code*) e il programma (*program*), secondo il formato «*course* | *code* | *program*», illustrato in Figura 1:

course	code	program
linear algebra and geometry	BAA	the space of the mxn matrices: sum and product by scalars. the transpose. square, symmetric, skew-symmetric matrices. product of matrices. invertible matrices. the determinant and its properties. binet's theorem. [...] eigenvectors and eigenvalues. characteristic polynomial. [...] plane and space geometry.

Fig. 2: Esempio di formato input

Una caratteristica rilevante ai fini di questo lavoro è la marcata eterogeneità dei programmi in termini di lunghezza, formato di scrittura e livello di dettaglio dei contenuti. La lunghezza dei testi varia, infatti, da un minimo di 21 a un massimo di 283 parole: alcuni programmi si riducono a un elenco essenziale di argomenti, mentre altri assumono la forma di una descrizione discorsiva che include anche le modalità didattiche e il monte ore. A titolo di esempio, il programma di Fisica Generale 2 (BAG) si esaurisce in un'unica riga: "*electric and magnetic phenomena in vacuum and matter, stationary case and non-stationary case. geometric optics and basic principles of wave optics*", mentre quello di Laboratorio di Automazione (BAU) si sviluppa in un testo articolato che descrive gli obiettivi del corso, i blocchi funzionali affrontati e le attività di laboratorio.

III.3 Il grafo dei docenti

La valutazione dei risultati assume come termine di paragone il grafo dei docenti. In questa sede se ne descrive l'istanza concreta relativa al caso di studio, rimandando al Capitolo I per la sua natura e per le cautele con cui va interpretato.

Il grafo in questione mette in relazione i 32 insegnamenti considerati e, nella forma in cui è stato originariamente raccolto, comprende 50 dipendenze dirette. Questa forma però non è direttamente utilizzabile come riferimento, poiché contiene dei cicli: tre insegnamenti fondamentali del primo anno - algebra lineare e geometria, analisi matematica 1 e fisica generale 1 - risultano legati da dipendenze reciproche, ciascuno indicato come prerequisito degli altri due. Questa reciprocità riflette la stretta interconnessione di tre discipline di base, erogate in parallelo nel primo semestre e con ampie porzioni di conoscenze condivise, tra le quali non è possibile stabilire un ordine univoco di precedenza. Poiché però il grafo deve risultare aciclico (cfr. Capitolo I), i sei archi che intercorrono tra i tre insegnamenti vengono eliminati, ottenendo così un grafo diretto aciclico composto da 44 dipendenze. È questa la versione del grafo dei docenti adottata come riferimento in tutte le valutazioni presentate nei capitoli successivi.

A titolo illustrativo si riporta l'immagine in Figura 2, mentre l'elenco specifico completo delle 44 dipendenze è riportato in Appendice A.

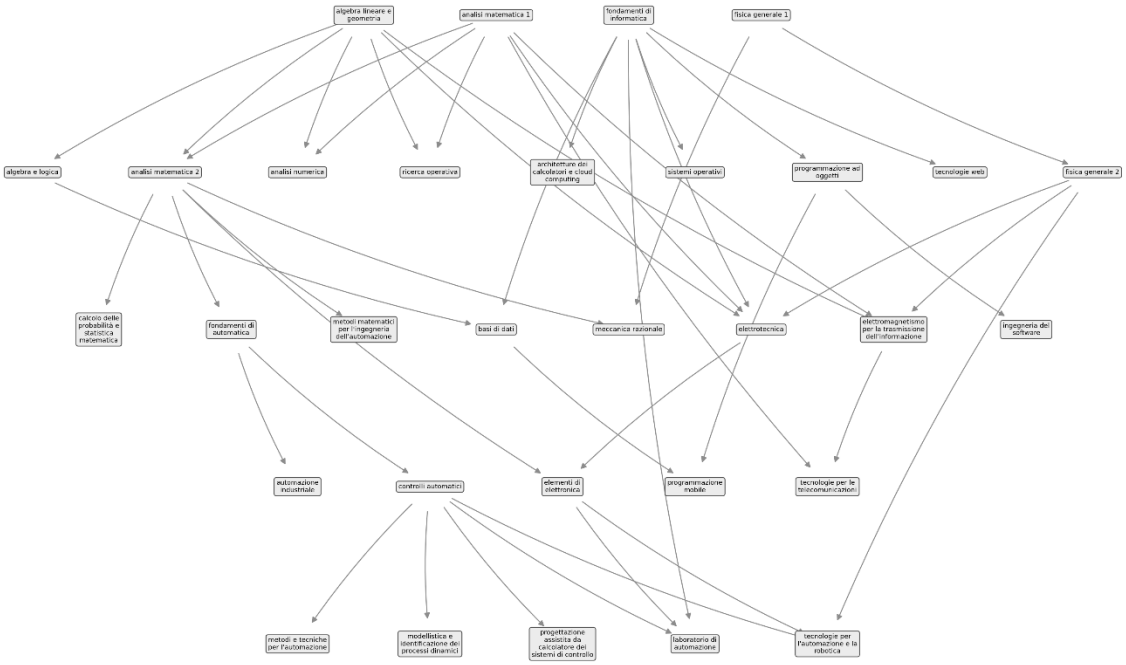


Fig. 3: Grafo dei docenti aciclico - 31 corsi[30]*,44 archi.*

[30] Uno dei 32 insegnamenti, Economia dell'Impresa, non è né prerequisito né dipendente da altri corsi. Costituisce un nodo isolato e per questo la Figura 2 ne rappresenta 31. Il corso resta comunque tra i 32 valutati: non avendo archi nel riferimento, non contribuisce con veri né falsi negativi, mentre eventuali dipendenze che il modello dovesse attribuirgli verrebbero conteggiate come falsi positivi.

III.4 Infrastruttura e configurazione del modello

Gli esperimenti sono stati condotti su una macchina dedicata del Dipartimento di Ingegneria dell'Informazione, dotata di due GPU NVIDIA GeForce GTX 1080.

III.4.1 Il modello gpt-oss:20b

Il modello impiegato è gpt-oss:20b, un modello *open-weights* rilasciato da OpenAI nel 2025 con licenza Apache 2.0. Appartiene alla famiglia gpt-oss, la quale è composta da due varianti: una da 120 e una da 20 miliardi di parametri, di cui si è scelta quest'ultima, progettata per l'esecuzione locale e scenari a bassa latenza[31].

Dal punto di vista architetturale, si tratta di un modello *Mixture-of-Experts* (MoE). In un modello di questo tipo la parte della rete che elabora ciascuna parola (token) non è un blocco unico, ma è suddivisa in più sottoreti specializzate, dette "esperti": per generare ogni token il modello non le attiva tutte, bensì solo un piccolo sottoinsieme. In questo modo, pur disponendo di circa 21 miliardi di parametri complessivi, ne impiega soltanto circa 3,6 miliardi per ciascun token. Il costo di calcolo risulta così molto inferiore a quello di un modello di pari dimensioni in cui, invece, tutti i parametri concorrono alla generazione di ogni token.

[31] OpenAI, *gpt-oss-120b & gpt-oss-20b Model Card*, arXiv:2508.10925 (2025); scheda del modello su huggingface.co/openai/gpt-oss-20b.

A questa efficienza si aggiunge quella data dalla *quantizzazione*, ossia la memorizzazione dei parametri con una precisione numerica ridotta. I pesi degli esperti, che da soli costituiscono oltre il 90% dei parametri del modello, sono salvati nel formato MXFP4: circa 4,25 bit ciascuno, anziché i consueti 16 o 32. Questa compressione fa scendere l'occupazione del modello sotto i 16 GB di memoria, così da consentirne il caricamento sulle due schede grafiche a disposizione, dove occupa complessivamente circa 14 GB.

Una seconda caratteristica rilevante è che gpt-oss:20b è un *reasoning model*, cioè un modello orientato al ragionamento. Prima di fornire la risposta finale, elabora una serie di passaggi intermedi (*chain-of-thought*) e solo al termine formula l'output vero e proprio. È possibile regolare quanto a fondo il modello ragiona scegliendo tra tre livelli: basso, medio, alto, dove un livello più alto produce un ragionamento più approfondito, a costo di tempi di elaborazione maggiori.

Il modello dispone anche di capacità agentiche (chiamata di funzioni, navigazione web, esecuzione di codice), che tuttavia non vengono utilizzate in questa sede.

III.4.2 Ambiente di esecuzione e interazione multi-turn

Il modello viene utilizzato in locale tramite Ollama, l'ambiente di esecuzione che ne gestisce il caricamento in memoria e ne espone le funzionalità attraverso un'interfaccia programmatica. L'interazione avviene tramite l'endpoint /api/chat di

Ollama[32]. Tale scelta risponde a due esigenze: la prima, di natura tecnica, è che gpt-oss è stato addestrato su un formato di conversazione specifico, chiamato *harmony response format*, in assenza del quale non produce risultati corretti[33]. L'endpoint /api/chat riceve i messaggi già distinti per ruolo (sistema, utente, assistente) e li converte automaticamente in quel formato attraverso il *modello di conversazione* associato a gpt-oss[34], evitando così di doverlo comporre a mano a ogni chiamata.

La seconda esigenza è legata alla natura sequenziale della pipeline. I suoi passaggi non sono indipendenti, ciascuno si fonda sugli output prodotti nei precedenti, per cui il modello deve poter ricordare ciò che ha già elaborato.

Questa memoria conversazionale, però, non è di per sé garantita: un LLM per sua natura non conserva le interazioni precedenti. Ogni chiamata viene elaborata in modo indipendente e il modello vede unicamente il testo che riceve in quel momento. Una volta generata la risposta, non ne mantiene traccia.

---

[32] Un *endpoint* è un punto di accesso di un'interfaccia programmatica (API), cioè un indirizzo a cui inviare richieste per ottenere una specifica funzionalità. Ollama espone /api/chat per l'interazione conversazionale e /api/generate per il completamento di un singolo testo.

[33] Lo *harmony response format* è il formato strutturato con cui gpt-oss organizza la conversazione: distingue i ruoli dei partecipanti e separa, mediante appositi token speciali, i passaggi di ragionamento dalla risposta finale. Il modello lo richiede perché è stato addestrato su di esso. Rilasciato da OpenAI (github.com/openai/harmony).

[34] Il *modello di conversazione* (*chat template*) è lo schema che trasforma la lista dei messaggi nel testo formattato atteso dal modello, inserendo i token e la struttura previsti dal formato harmony.

L'apparente continuità della conversazione non è quindi ricostruita e garantita dal modello, ma dal software che effettua le chiamate, nel presente caso rappresentato dallo script che esegue la pipeline. È quest'ultimo che, a ogni passaggio, riaccoda al messaggio corrente l'intera storia precedente (l'istruzione di sistema, tutte le domande e le corrispondenti risposte) e la reinvia per intero, così che il modello possa "rileggere" ciò che è stato detto prima.

Da qui deriva uno degli aspetti più delicati da gestire: la lunghezza del contesto. Questa cresce rapidamente a ogni passaggio e se il totale dei token accumulati supera l'ampiezza della finestra di contesto, i token più vecchi vengono scartati e il modello può quindi perdere parte dei passaggi iniziali, con conseguente degrado o incoerenza dell'output. È questa la ragione per cui, come descritto nel paragrafo seguente, la finestra di contesto è stata dimensionata con ampio margine e il suo riempimento viene monitorato a ogni chiamata.

III.4.3 Parametri di inferenza

I parametri dell'inferenza sono stati fissati e documentati esplicitamente.
Per comodità di lettura sono stati riassunti in Tabella 2, che ne riepiloga i valori e le definizioni ufficiali.

Parametro	Valore	Descrizione
Modello	gpt-oss:20b	Modello open-weights, servito in locale da Ollama.
Runtime	Ollama 0.20.7	Ambiente di esecuzione; versione fissata (aggiornamento automatico disabilitato).
Endpoint	/api/chat	Interfaccia conversazionale multi-turn: a ogni chiamata si invia l'intera cronologia dei messaggi.
Temperature	1.0	Regola la casualità del campionamento; valore raccomandato per gpt-oss.[35] (default Ollama: 0.8)
Seed	42 (variato)	Seme del generatore pseudo-casuale: a parità di prompt fissa l'output; variato per fare l'ensemble.[36]
num_ctx	32768	Ampiezza della finestra di contesto in token; il default (2048) è insufficiente per la pipeline.
num_predict	default	Numero massimo di token generati; lasciato al default (−1, generazione illimitata).
Think	default	Sforzo di ragionamento (reasoning effort) di gpt-oss; lasciato al valore predefinito.

Tab. 2: Configurazione del modello e parametri di inferenza

Di seguito se ne motivano le principali scelte.

La *temperatura*: è il parametro che regola la casualità del campionamento, valori bassi rendono le risposte più prevedibili e ripetitive, valori alti più varie. In questo contesto è impostata a 1.0, il valore raccomandato per gpt-oss. Non si è scelto di abbassarla, idealmente fino a 0, perché non garantirebbe comunque un output deterministico e un valore più basso penalizzerebbe la qualità del ragionamento.

[35] *OpenAI, scheda del modello gpt-oss-20b (huggingface.co/openai/gpt-oss-20b)*

[36] *Ollama, Modelfile reference (docs.ollama.com/modelfile) e API reference (docs.ollama.com/api).*

Il *seed*: La ripetibilità è perseguita attraverso il seme di generazione, il numero che inizializza il generatore pseudo-casuale da cui dipende il campionamento. Fissandolo, a parità di prompt, il modello produce lo stesso output, mentre farlo variare produce esecuzioni diverse. Nel presente lavoro il seed è fissato a 42 come valore di riferimento per le prove ripetibili e variato per le esecuzioni multiple.

L'ampiezza del contesto (*num_ctx*): portata a 32768 token, in quanto il valore predefinito di Ollama (2048) è risultato insufficiente a contenere la cronologia accumulata dalla pipeline. Il tetto sul numero di token generati (*num_predict*) e il livello di ragionamento (*think*) sono lasciati ai valori predefiniti, così da non troncare le catene di ragionamento e alterare il comportamento nativo del modello.

Un ultimo accorgimento ha riguardato la versione di Ollama, mantenuta fissa alla 0.20.7 con l'aggiornamento automatico disabilitato. Le versioni successive si sono infatti rivelate incompatibili con l'hardware in uso, causando errori CUDA che interrompono l'inferenza.

CAPITOLO IV

ADATTAMENTO AL MODELLO LOCALE

Questo capitolo apre la parte sperimentale, fornendo il fondamento empirico dietro ogni scelta che ha poi portato alla definizione della metodologia proposta.

Che cosa accade quando quella pipeline, nata per un modello proprietario in cloud, viene portata su un modello a pesi aperti eseguito in locale?[37]

IV.1 Il comportamento del modello sulla pipeline originale

Il punto di partenza consiste nell'eseguire la pipeline nella sua forma originale, quindi lo stesso prompt e la stessa sequenza di passaggi della tesi di riferimento (Cap. I.5.2). L'obiettivo è osservare come il modello locale si comporta e quali difficoltà emergono, confrontando i risultati con quelli del modello cloud.

Poiché l'esito di una singola esecuzione è, come si vedrà, fortemente variabile, la pipeline è stata lanciata venti volte, facendo variare il seed. Le prestazioni riportate in seguito sono quindi la media su 20 run con la relativa deviazione standard, secondo il protocollo *per singola esecuzione* descritto nel II.5.

[37] Le prestazioni sono misurate rispetto al grafo dei docenti secondo le metriche del I.5.3 e, data la variabilità tra esecuzioni, secondo le modalità di valutazione introdotte nel II.5; l'infrastruttura di esecuzione è quella descritta nel III.4.

IV.1.1 Primi risultati e criticità

Il primo risultato è che la pipeline risulta pienamente eseguibile in locale: ogni esecuzione si completa, in media, in 5–6 minuti. Tuttavia, emergono due criticità.

La prima è l'inaffidabilità del singolo run. Delle venti esecuzioni, quattro non producono un output utilizzabile: pur completando l'intera pipeline, restituiscono un grafo vuoto, privo di qualsiasi dipendenza.

Le sedici run valide restituiscono un F1 medio pari al 25,99%, con una deviazione standard di circa quindici punti e valori che spaziano dall'8,40% al 54,35%.

Un risultato nettamente inferiore a quello ottenuto nel precedente lavoro di tesi, che sul proprio grafo finale - una singola esecuzione con GPT-o3 in cloud - riporta un F1 dell'80,43%[38]. Ma soprattutto, a parità di prompt, il solo variare del seme fa oscillare il risultato tra valori anche molto distanti tra loro. Prendere per buono un run isolato significherebbe riportare un numero non riproducibile ed è questa constatazione a giustificare l'intera impostazione basata sull'esecuzione ripetuta.

[38] Occorre precisare con maggior rigore che i confronti con il lavoro di Salvatori richiamati in questo e nel prossimo capitolo non sono immediati. I suoi risultati sono valutati sulla prestazione di un unico grafo finale, selezionato e ripulito, ottenuto in cloud con un modello proprietario di fascia ben superiore (GPT-o3); i valori del presente lavoro sono invece calcolati su più esecuzioni di un modello locale da venti miliardi di parametri.

La seconda criticità emersa è l'over-generation, cioè la tendenza del modello a produrre molte più dipendenze di quante il riferimento ne contenga. Il numero di archi generati varia sensibilmente da un seme all'altro e con esso la qualità del grafo. Quando il modello è parsimonioso la *precision* regge, mentre quando la produzione esplode affonda e trascina con sé l'F1, in quanto la grande maggioranza degli archi in eccesso si rivelano falsi positivi. Il caso limite osservato è un seme che recupera oltre il 60% delle dipendenze del riferimento ma con una precision inferiore al 10%, avendo prodotto diverse centinaia di archi a fronte dei 44 attesi. Questa sovra-generazione è il principale fattore che comprime la precision ed è il vero collo di bottiglia del metodo, che si è cercato di contenere con gli interventi sul prompt

IV.1.2 Dove nascono gli errori: confronto step per step

Analizzando gli output di ciascuno step e confrontandoli con il vecchio lavoro emergono le origini di queste criticità.

Lo STEP 0, in cui il modello elenca le conoscenze fornite da ciascun corso, si trasferisce in locale senza particolari problemi. Lo STEP 1, che elenca i prerequisiti di ogni corso, tende invece a essere sovrabbondante, attribuendo a ogni insegnamento più conoscenze richieste di quante ne servano realmente. Questo potrebbe rappresentare una delle fonti da cui si origina l'over-generation, poiché ogni prerequisito in più diventa una potenziale dipendenza allo step successivo.

Lo STEP 2, dove avviene l'inferenza vera e propria delle dipendenze A→B, cattura correttamente molti legami nel verso giusto, ma introduce tre patologie ricorrenti: i *self-loop*, cioè archi di un corso verso sé stesso (ad esempio BAH → BAH); le *coppie reciproche*, in cui compaiono entrambi i versi tra due corsi; e, più in generale, un numero non trascurabile di *archi orientati al contrario.*

Un'analisi dettagliata su una singola esecuzione rappresentativa, che produce 105 dipendenze, mostra che i corsi fondamentali del primo anno hanno un grado di dipendenze in entrata molto alto. Ad esempio, Analisi Matematica 1 compare come destinazione di diciannove dipendenze e come sorgente di sole sette, quando invece, trattandosi di un corso base, ci si aspetterebbe l'opposto. Il modello, in sostanza, confonde con una certa frequenza le due direzioni, cioè il «fornisce» e il «richiede». Nella stessa esecuzione si contano quindici coppie reciproche e in molte di esse le due direzioni ricevono lo stesso peso: per Algebra Lineare (BAA) ed Elettrotecnica (BAQ), ad esempio, il modello genera contemporaneamente entrambe le direzioni e con la stessa confidenza:

BAA -> BAQ |p=0.9| (algebra lineare → elettrotecnica, corretto)

BAQ -> BAA |p=0.9| (elettrotecnica → algebra lineare, invertito)

Il filtro che in fase di valutazione dovrebbe conservare, tra due versi opposti, quello di peso maggiore diventa così inapplicabile e la scelta ricade sulla direzione elencata per prima: un tie-break di fatto arbitrario, indipendente dall'effettivo verso.

Il compito di ricomporre queste incoerenze spetterebbe allo STEP 3. Tuttavia, i dati mostrano che tale passaggio da un lato, non corregge: le quindici coppie reciproche prodotte allo STEP 2 sopravvivono integralmente alla revisione; dall'altro, in un caso su cinque (le quattro esecuzioni nulle descritte in precedenza), restituisce una risposta vuota. Lo STEP 3 si rivela quindi inefficace e instabile in questo contesto, un limite che ne motiverà la rimozione, quantificata nel prossimo paragrafo.

IV.2 Rimozione step di revisione

La diagnosi condotta porta come conseguenza operativa la rimozione dello step 3. Per isolare e quantificare l'effetto si è ricostruita, per ogni esecuzione, la lista prodotta dallo STEP 2 e la si è valutata con lo stesso protocollo dell'output finale, confrontando così il grafo «prima» (STEP 2) e «dopo» (STEP 3) a parità di seme.
Il confronto restituisce tre evidenze convergenti. La prima è che lo STEP 3 non corregge le incoerenze che dovrebbe risolvere: le coppie reciproche, le inversioni e i self-loop prodotti dallo STEP 2 sopravvivono pressoché immutati alla revisione. La seconda è che non porta alcun guadagno di prestazioni: sulle esecuzioni in cui restituisce un output, l'F1 medio risulta di circa un punto e mezzo inferiore a quello che si otterrebbe fermandosi allo STEP 2. Quando interviene, rimaneggia diverse decine di pesi per esecuzione e tende a cancellare archi, abbassando il *recall*.

La terza motivazione, infine, è che è il passaggio più instabile: in un caso su cinque collassa in una risposta vuota, distruggendo un output dello STEP 2 che era invece regolare. A ciò si aggiunge il costo in termini di tempo: risulta essere lo step più lento e assorbe da solo circa il 30–40% del tempo di elaborazione.

Un punto su cui serve maggiore chiarezza riguarda in che cosa consiste il collasso a vuoto citato in precedenza, perché è istruttivo sul tipo di modello con cui si lavora. Nelle quattro esecuzioni nulle lo STEP 3 non si interrompe per un errore: la pipeline lo esegue per intero, impegna il modello per alcuni minuti ma il canale di risposta da cui si ricava il grafo torna vuoto. Gpt-oss:20b è un modello di tipo *reasoning*, che distingue un canale di ragionamento interno (*analysis*), in cui elabora la sua catena di ragionamento, da un canale di risposta finale (*final*)[39], che è ciò che la pipeline legge. Il prompt dello STEP 3 richiede esplicitamente di restituire la lista completa come risposta finale: «*give your FINAL ANSWER as the complete list of dependencies […]; Include all dependencies (the full final list, not just the changes)*» ma questo non è sufficiente a impedire il fenomeno. In un modello il cui comportamento non è pienamente riproducibile, un fallimento intermittente di questo tipo non è prevedibile a priori né eliminabile intervenendo ulteriormente sul prompt, come dimostra il fatto che persiste pur in presenza di un'istruzione esplicita.

[39] OpenAI, *OpenAI Harmony Response Format*, OpenAI Cookbook

Il risultato conferma sul piano empirico quanto anticipato nel II.2: la revisione autonoma, priva di un riscontro esterno, non aggiunge valore. L'output della pipeline coincide dunque, da qui in avanti, con la lista prodotta dallo STEP 2, risultata regolare in tutte le esecuzioni

IV.3 Interventi su prompt e input

Individuata nell'over-generation la causa principale che comprime la precision, restano da misurare gli interventi pensati per contenerla. Ciascuno è stato valutato con un'ablazione controllata: la stessa pipeline, gli stessi venti semi e la stessa valutazione, cambiando un solo elemento per volta, così da isolarne l'effetto.

IV.3.1 Segmentazione dei syllabus

La versione segmentata e l'input originale sono stati dati in pasto alla stessa pipeline, così che l'unica differenza sia la forma dell'input.

Mediando sui venti semi, tutti gli indicatori si muovono nella stessa direzione favorevole: l'F1 sale dal 24,62% al 30,08%, la *precision* dal 25,65% al 34,27% e il *recall* dal 24,66% al 29,66%. I falsi positivi medi per esecuzione scendono da 40,2 a 30,4 e i veri positivi salgono da 10,8 a 13,1. Gli archi del riferimento mai individuati in nessuna delle venti esecuzioni calano da 10 a 7 su 44 e la quota di archi prodotti nel verso sbagliato dal 40,2% al 31,5%.

La concordanza di indicatori tanto diversi costituisce un segnale coerente a favore della segmentazione. Tuttavia, questo miglioramento va letto con una precisazione. È una media su venti esecuzioni caratterizzate da una variabilità molto elevata, con deviazione standard dell'F1 sui tredici–sedici punti. Per questo il guadagno medio, pur concorde su ogni metrica, resta entro la fascia di rumore della singola esecuzione: un confronto appaiato seme per seme mostra che su venti semi dodici migliorano e otto peggiorano. Ciò non segnala un limite dell'intervento, ma riflette quanto già osservato: il singolo run è uno stimatore troppo instabile perché l'effetto di una scelta vi si dimostri in modo netto. È proprio questo a motivare il ricorso all'ensemble (capitolo V), dove il guadagno diventa ampio e stabile.

IV.3.2 Normalizzazione del testo: un controllo negativo

La segmentazione cambia la struttura del testo senza toccarne il contenuto.

Visti i buoni risultati, è lecito chiedersi se non convenga andare oltre e ripulire attivamente il testo. Tale scelta ricade tra le tecniche classiche di pre-processing del text mining, che prevedono interventi di tokenizzazione, *lowercasing*, rimozione di *markup*, (…), ed è da tempo considerata il passaggio standard che precede la maggior parte delle analisi testuali. Il suo razionale è chiaro: rimuovere le parole frequenti e poco informative abbatte rumore e ridondanza, migliorando così le prestazioni dei classificatori.

Tale motivazione però, si è indebolita con l'arrivo dei modelli basati sul contesto. Le indicazioni recenti raccomandano una pulizia leggera, che preservi il contesto, anziché una normalizzazione aggressiva. Per un modello che sfrutta l'intero contesto testuale e l'ordine delle parole, rimuovere ciò che appare rumore rischia di sottrarre segnale utile[40]. Se dunque per i metodi classici la pulizia era un guadagno quasi certo, per un modello come quello impiegato qui l'esito non è scontato ed è ciò che questo controllo intende verificare.

L'intervento testato applica, sopra l'input già segmentato, una pulizia volutamente conservativa: rimuove numerazioni ed etichette di sezione («unit 1–6», «part i:», «theory:», «exercises:»), le parentesi amministrative («(2 ects)», «(lectures, 72 hours)») e una lista, curata e verificata a mano, di frasi-cornice e logistiche («*the course is divided into two parts*», «*the various topics covered are as follows:*»); si uniforma inoltre la tipografia. I termini tecnici «0-1», «2-port», «p-n» e le lettere accentate non vengono toccate. Così, ad esempio, in Elettrotecnica spariscono i marcatori «part i:» e «part ii:», in Sistemi Operativi i marcatori «unit N», in Ingegneria del Software le annotazioni «(… ects)». A differenza della segmentazione, che conservava ogni parola, questa pulizia modifica i token: rimuove una sessantina tra marcatori e voci amministrative e una dozzina di frasi.

[40] NVIDIA (2024), *Mastering LLM Techniques: Text Data Processing*, NVIDIA Technical Blog.

Sottoposta alla stessa ablazione della segmentazione, emerge che la pulizia non porta alcun miglioramento; anzi, in media ogni indicatore arretra. L'F1 medio scende dal 30,08% della sola segmentazione al 26,37%, la *precision* dal 34,27% al 29,53% e il *recall* dal 29,66% al 26,02%; inoltre, l'over-generation non si riduce affatto, dato che i falsi positivi medi per esecuzione, anziché calare, aumentano (da 30,4 a 33,8). Come per la segmentazione, il problema dell'elevata variabilità tra esecuzioni si ripresenta anche qui, ma mentre nel primo caso tutti gli indicatori si muovevano concordi verso il miglioramento, qui no[41].

La ragione si trova osservando che cosa viene effettivamente rimosso: il confine tra «rumore» e «contenuto» non è netto. Nei syllabus più discorsivi, alcune frasi che si presentano come introduzione o logistica nominano in realtà materia concreta. In Laboratorio di Automazione, per esempio, tra le frasi eliminate figurano «*students will learn how to analyse data-sheets of interface devices with the micro controller…*» e «*these topics provide the link to real applications and to the state of the art in embedded robotics*»: enunciati di forma discorsiva che però menzionano contenuti tecnici specifici su cui il modello potrebbe far leva per inferire una dipendenza. Rimuoverli, anche con una lista compilata a mano, significa gettare via il segnale insieme al rumore.

[41] Le esecuzioni con testo ripulito, inoltre, sono state generate in una giornata diversa da quelle segmentate; poiché tuttavia la pulizia non produce alcun miglioramento su nessuna metrica, la conclusione non dipende dal confronto tra giornate.

Il controllo conferma così il punto operativo individuato: sull'input conviene rendere esplicita la struttura conservando integralmente il contenuto, non potarlo.

IV.3.3 Esempi negativi nel prompt

L'ultimo intervento agisce direttamente sul prompt dello STEP 2, con gli esempi negativi introdotti nel II.3, e mira in modo mirato all'over-generation.

L'ablazione ne misura l'effetto confrontando il prompt con e senza il blocco aggiunto, appaiando il confronto sugli stessi semi e nella stessa giornata.

La precision sale dal 28,56% al 40,69%; l'F1 media passa dal 25,85% al 31,58% e il recall cresce lievemente, dal 25,83% al 27,73% (in tutte le condizioni la deviazione standard dell'F1 resta attorno ai tredici punti)[42]. Il fatto che anche il recall migliori, sia pure di poco, conferma che il chiarimento sui prerequisiti reali tra corsi affini ha effettivamente protetto le dipendenze legittime: gli esempi negativi hanno ridotto gli archi spuri senza intaccare quelli veri.

[42] Il valore della base senza esempi negativi (25,85%) differisce da quello riportato per la segmentazione nel paragrafo precedente (30,08%) perché ottenuto in una diversa giornata. Il confronto con e senza esempi negativi resta però valido, perché appaiato sugli stessi seed e giornata.

CAPITOLO V

INSTABILITA' DEL SINGOLO RUN, ENSEMBLE E RANDOMIZZAZIONE

Nelle analisi svolte fino ad ora, uno degli aspetti più importanti emerso è il fatto che, a parità di input, gli output di più esecuzioni sono diversi tra loro, talvolta anche di molto. Nel presente capitolo si indagano le cause sottostanti a tale fenomeno, i rimedi proposti e un'analisi qualitativa degli errori ricorrenti.

V.1 Il comportamento non deterministico del modello locale

È stato osservato che eseguire più volte la stessa pipeline, con lo stesso input, non produce il medesimo grafo. Questo comportamento coinvolge due aspetti diversi e occorre quindi porsi due domande distinte: *quanto* cambia il risultato al variare di un elemento della generazione, come ad esempio il seed o l'ordine degli input? Il risultato resta identico quando *non* si cambia nulla, a seme e configurazione fissi?

V.1.1 Variabilità tra esecuzioni

La prima domanda riguarda la variabilità tra esecuzioni.

Mantenendo fissa l'intera configurazione e facendo variare soltanto il seme di campionamento su un insieme di trenta esecuzioni, il numero di dipendenze estratte e la loro identità cambiano in modo marcato da un'esecuzione all'altra.

Come si evince dal grafico sottostante, i valori di F1 delle singole run oscillano notevolmente, con una media del 31,6% e una deviazione standard di circa 13 punti:

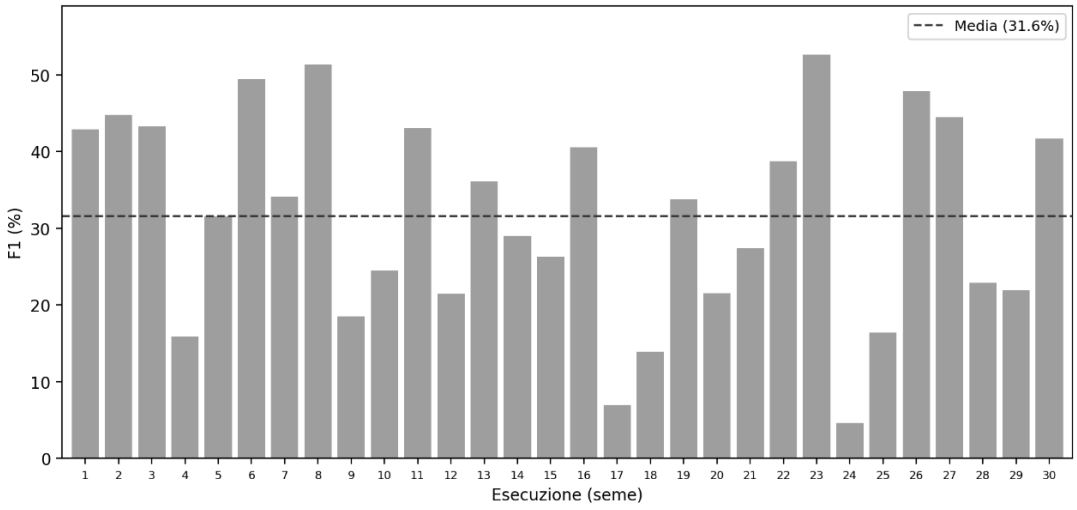


Fig. 4: F1 delle 30 esecuzioni a solo seme (ordine fisso), nell'ordine di esecuzione

È come se, quale grafo si ottenga da una singola esecuzione, sia una questione di fortuna. Prendere quindi per buono un run isolato significa riportare un risultato non riproducibile e potenzialmente molto lontano dal comportamento tipico del modello. Di conseguenza, una singola esecuzione della pipeline è uno stimatore debole e instabile del grafo di propedeuticità. Il rimedio naturale sarebbe quello di filtrare le dipendenze meno affidabili attraverso l'uso di una soglia. A ciascuna dipendenza il modello associa un peso $p \in [0,1]$ e ci si aspetterebbe che alzandone la soglia si eliminino gli archi spuri e la precisione migliori, tuttavia ciò non accade. La valutazione al variare della soglia è sostanzialmente piatta: passando da nessun filtro a una soglia di 0.4 l'F1 resta praticamente invariato.

Le ragioni sono due e portano alla stessa conclusione.

Da un lato, la maggior parte degli archi riceve un peso alto, per cui una soglia bassa non taglia quasi nulla e una soglia alta svuota il grafo di colpo, senza una fascia intermedia utile. Dall'altro, il peso p è spesso assegnato con valori alti anche a dipendenze sbagliate, oppure con confidenze simili nei casi in cui vengono generati, per una stessa coppia di corsi, archi in entrambe le direzioni.

Questo è il primo indizio concreto di un modello debole: non che sbaglia in modo casuale, ma che non dispone di un segnale interno affidabile con cui separare ciò che ha estratto bene da ciò che ha estratto male.

C'è però un secondo aspetto, che cambia la lettura della variabilità: aggregando le diverse run, la copertura delle dipendenze reali trovate aumenta.

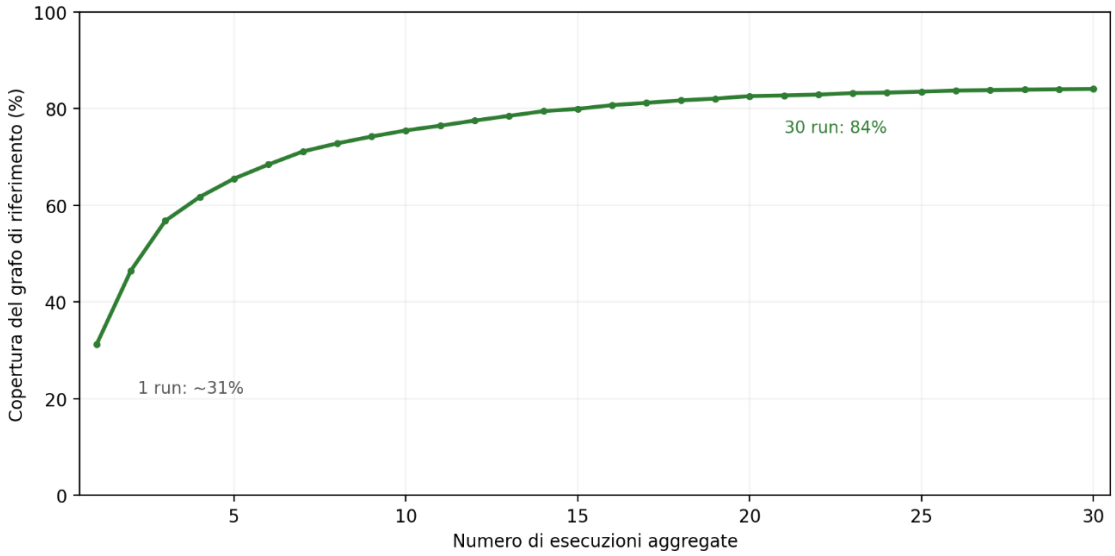


Fig. 5: Copertura del grafo di riferimento al crescere del numero di esecuzioni aggregate.

I run non sbagliano tutti allo stesso modo: gli insiemi di dipendenze corrette si sovrappongono poco tra loro e gli archi "ruotano" da un'esecuzione all'altra. L'informazione cioè, è presente nel modello, è solo distribuita su molte esecuzioni e nessun singolo run la cattura per intero.

V.1.2 Riproducibilità a configurazione fissa

Resta aperta la seconda domanda iniziale: se non si cambia nulla, l'output è riproducibile? Per rispondere sono state condotte una serie di prove controllate, in cui a variare sono stati alcuni parametri interni al modello.

La documentazione ufficiale di Ollama elenca, tra i parametri di generazione che possono essere impostati e passati direttamente via codice, il *seed*, che imposta il seme del generatore di numeri casuali e, nelle parole della documentazione, *"fissandolo a un valore specifico il modello genera lo stesso testo a parità di prompt"*. Nella pratica però questa garanzia si rivela fragile e, nel caso qui presentato, è utile distinguere due livelli: cosa accade entro una stessa sessione e cosa accade tra giornate diverse.

Entro una stessa sessione, ripetendo la stessa richiesta, l'output non è costante ma nemmeno casuale: alterna tra due esiti, ciascuno deterministico e ripetibile, ad esempio un grafo da 53 dipendenze e uno da 117, che si ripresentano identici a ogni ricorrenza. Si tratta di un comportamento segnalato anche da altri utenti sul

repository ufficiale di Ollama. In particolare, in una di queste segnalazioni l'autore osserva che, inviato più volte lo stesso prompt con lo stesso seme, le prime risposte risultano identiche mentre a partire da una certa chiamata iniziano a divergere.
L'ipotesi formulata è che a cambiare sia uno stato interno del server tra una chiamata e la successiva (la cache del contesto, o KV-cache[43]). Per isolare la causa è stato modificato il parametro *keep_alive*, il quale controlla per quanto tempo il modello resta caricato in memoria dopo la richiesta (per default impostato a cinque minuti). Imponendo lo scaricamento del modello dalla memoria a ogni chiamata (keep_alive = 0), l'alternanza è scomparsa del tutto e l'output è tornato deterministico e ripetibile. Entro la singola sessione, quindi, la riproducibilità è recuperabile, al prezzo di ricaricare il modello a ogni chiamata.

Tra giornate diverse, invece, nemmeno l'output "a freddo" (quello della prima chiamata) è una costante assoluta. Il motivo è che l'output di questi modelli non dipende soltanto dall'input e dal seme, ma anche dalla forma numerica del calcolo, che a sua volta dipende da come il modello viene eseguito sull'hardware: è una proprietà nota in letteratura come mancata invarianza al batch (cfr. nota 13).

[43] La KV-cache (key-value cache) è la memoria in cui il modello conserva i risultati intermedi del meccanismo di attenzione calcolati sui token già elaborati, per non doverli ricalcolare a ogni passo. Se questa memoria viene riutilizzata tra una chiamata e la successiva, lo stato di partenza del calcolo non è identico a quello di una chiamata "a freddo", e questo può bastare a modificare l'output.

Se ne è avuta conferma sui dati di questa tesi, variando un parametro di esecuzione, la dimensione del batch con cui il prompt viene elaborato (num_batch[44]). A valori diversi del parametro corrispondono esiti diversi, ciascuno deterministico (per esempio 53, 116 e 39 dipendenze). Il parametro num_batch resta però fisso lungo tutta la pipeline e non è quindi la variabile che cambia da un giorno all'altro. Serve a illustrare il tipo di fenomeno: quando qualcosa altera la forma del calcolo, l'output può cambiare pur restando identici l'input e il seme.

Come ultime prove, sono stati verificati uno ad uno alcuni elementi (un riavvio, un aggiornamento del driver, una diversa suddivisione del carico tra le due GPU). Nessuno di essi risultava cambiato tra le giornate e non è stato quindi possibile ricondurre la differenza a un fattore controllabile a livello di pipeline.
In sintesi: entro una singola sessione il modello è controllabile e riproducibile, forzando lo scaricamento a ogni chiamata; tra sessioni e tra giorni invece, la riproducibilità non è garantita, e non a causa del seme o della configurazione, che restano fissi, ma per il modo in cui il modello viene eseguito.

[44] num_batch corrisponde al parametro n_batch di llama.cpp, la dimensione logica del batch con cui i token del prompt vengono elaborati nella fase di *prefill*, cioè la fase in cui il modello elabora tutto il prompt in ingresso prima di iniziare a generare la risposta. Cfr. la documentazione di llama.cpp (ggml-org) sul batch processing.

V.2 L’ensemble come rimedio

Le due osservazioni del paragrafo precedente (l'instabilità del singolo run e il fatto che ogni esecuzione coglie una porzione diversa), indicano con chiarezza cosa fare: non affidarsi a una sola esecuzione, ma aggregarne molte in un unico grafo.

Questa strategia corrisponde a una famiglia di tecniche consolidate e ben note nell'apprendimento automatico, conosciute come metodi ensemble[45].

L'idea alla base è che la combinazione di molti predittori individualmente deboli possa dare luogo ad uno complessivo più accurato e stabile. Affinché questo accada, devono sussistere due condizioni: che i singoli predittori siano ragionevolmente competenti, cioè migliori del caso; e che siano sufficientemente diversi tra loro, cioè che non sbaglino tutti allo stesso modo. Entrambe sono soddisfatte.

Il singolo run infatti, per quanto debole, estrae comunque dipendenze in prevalenza corrette e run diversi recuperano insiemi di dipendenze che si sovrappongono solo in parte. La diversità, invece, nasce dall'introdurre elementi di variazione tra i diversi componenti, qui rappresentati non da modelli distinti, ma dalle singole esecuzioni dello stesso modello. La forma di variazione più semplice viene dal

[45] I metodi ensemble combinano le uscite di più modelli o esecuzioni per ottenere un risultato più accurato e stabile del singolo componente; l'idea funziona a condizione che i singoli predittori siano ragionevolmente competenti e commettano errori poco correlati tra loro. Cfr. Dietterich, T. G. (2000). *Ensemble Methods in Machine Learning*. pp. 1–15.

comportamento stocastico della stessa generazione, poiché basta ripetere l'esecuzione variando il seme perché il risultato cambi. Questa diversità si può poi accrescere perturbando anche l'input: è l'idea del bagging[46], in cui i componenti sono costruiti su versioni perturbate dello stesso input, per esempio randomizzandone l'ordine e la composizione, come si vedrà nel paragrafo V.3.
Nel contesto specifico dei LLM questo principio ha una declinazione particolarmente naturale, resa possibile proprio dal comportamento non deterministico, che diventa la sorgente di diversità di cui l'ensemble ha bisogno. L'aggregazione, inoltre, è tanto più indicata quanto più il modello di partenza è debole, esattamente la situazione di un modello locale da venti miliardi di parametri. A ciò si aggiunge una considerazione pratica: un modello eseguito in locale non comporta costi per chiamata, per cui ripetere la generazione decine di volte è pienamente sostenibile, e la numerosità delle esecuzioni, che è ciò che rende affidabile l'aggregazione, non costituisce un vincolo.

C'è però una differenza rispetto al caso classico. Di solito ogni esecuzione produce una singola risposta, tipicamente un numero o una breve stringa. Votare in quel caso significa contare quale risposta ricorre più spesso oppure farne la media.

[46] Bagging (*bootstrap aggregating*): tecnica ensemble in cui ogni componente è costruito su un campione perturbato dei dati di partenza e le uscite dei componenti vengono poi combinate per voto o per media. Cfr. Breiman, L. (1996). *Bagging Predictors*. Machine Learning, 24(2), 123–140.

Qui invece, ogni esecuzione produce un intero grafo, cioè un insieme di dipendenze orientate, e non avrebbe senso aggregare «il grafo di maggioranza» come se fosse un'unica risposta. L'aggregazione va portata al livello del singolo arco, chiedendosi per ogni possibile dipendenza in quante e in quali esecuzioni essa compaia e con quale peso. Questo passaggio si può realizzare in due modi ed entrambi sono stati implementati. Il primo è basato sul voto e verrà discusso nel dettaglio nel paragrafo V.2.1; il secondo sfrutta la media dei pesi p ed è discusso nel V.2.2

V.2.1 Il voto di maggioranza

Il primo metodo si basa sulla stessa idea che nell'ambito dei modelli linguistici è nota come *self-consistency:* si campionano più risposte per uno stesso prompt e si sceglie quella prodotta dalla maggioranza delle esecuzioni, con un miglioramento sistematico rispetto alla singola generazione[47]. Il meccanismo è il seguente.

Ogni esecuzione produce un insieme di dipendenze orientate. Per ciascuna coppia di corsi {u, v} si conta in quante esecuzioni quella coppia compare, in un verso qualsiasi, ottenendo un valore di supporto. La coppia entra nel grafo aggregato se il suo supporto raggiunge almeno una frazione *k* del numero totale *N* di esecuzioni,

[47] Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., Zhou, D. (2023). *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. International Conference on Learning Representations (ICLR). arXiv:2203.11171.

cioè se compare in almeno $k \cdot N$ run: k è quindi la quota minima di consenso richiesta perché una dipendenza sia ritenuta attendibile. Stabilito che la coppia va tenuta, se ne fissa il verso con un secondo voto, scegliendo l'orientamento (u→v oppure v→u) prodotto dalla maggioranza delle esecuzioni in cui la coppia compare; in caso di parità esatta tra i due versi l'arco viene scartato, perché il modello non esprime una direzione prevalente e sceglierne una sarebbe arbitrario. Infine, se il grafo così ottenuto contiene cicli, questi vengono rotti rimuovendo l'arco di peso minore in ciascun ciclo, per restituire un grafo orientato e aciclico.

La logica sottostante è che la ricorrenza di una dipendenza in molte run sia un indicatore di affidabilità ben più solido della confidenza dichiarata dal modello su una singola esecuzione. Ad esempio, una dipendenza che compare in venti run su trenta, pur con pesi diversi, è verosimilmente un segnale reale, mentre una che appare una volta sola è probabilmente un rumore.

Il parametro k governa in modo esplicito il compromesso tra *precision* e *recall*: abbassarlo fa sì che una coppia possa entrare nel grafo anche se compare poco, di conseguenza se ne tengono di più. Tra queste però, alcune sono dipendenze corrette che prima restavano fuori e quindi il *recall* cresce, ma molte altre sono archi occasionali e sbagliati, che fanno scendere la *precision*. Viceversa, alzando k, si tengono solo le coppie su cui concorda un'ampia maggioranza di esecuzioni: quasi tutti gli archi sbagliati vengono scartati e la *precision* sale, ma nella potatura cadono anche dipendenze corrette che il modello estrae di rado, e quindi il *recall* cala.

Applicando il metodo alle trenta esecuzioni di cui si erano riportati i valori delle singole run, il risultato è nettamente positivo: nel suo punto migliore l'F1 passa da circa il 31% della singola esecuzione a oltre il 57%.

Di seguito si riporta l'esito del voto al variare di *k* con passo 0.05.

k	Supporto min (run)	Precision	Recall	F1	Archi finali
0,05	2	28,57	72,73	41,03	151
0,10	3	33,68	72,73	46,04	130
0,15	5	41,67	56,82	48,08	90
0,20	6	41,38	54,55	47,06	87
0,25	8	50,00	54,55	52,17	72
0,30	9	61,11	50,00	55,00	58
0,35	11	61,11	50,00	55,00	58
0,40	**12**	**66,67**	**50,00**	**57,14**	**53**
0,45	14	67,86	43,18	52,78	44
0,50	15	76,00	43,18	55,07	40
0,55	16	72,73	36,36	48,48	36
0,60	18	68,75	25,00	36,67	25
0,65	20	71,43	22,73	34,48	23

Tab. 3: Voto di maggioranza su 30 esecuzioni (variazione solo seme): precision, recall e F1 (%) al variare della quota minima di consenso k. In grassetto la riga di F1 massimo

Il quadro conferma il compromesso atteso. Per valori bassi di *k* il grafo è ampio e sbilanciato verso il recall (a $k = 0.05$ il recall tocca il 72.7%, ma la precision è appena del 28.6%); crescendo *k* la precision sale, mentre il recall cala, finché una

soglia troppo alta ($k > 0{,}65$) svuota il grafo. Il miglior equilibrio si ottiene a $k = 0.40$, dove l'F1 raggiunge il 57,14% (con 22 dipendenze corrette su 44).

Il confronto con la singola esecuzione è il risultato principale ed è soprattutto sul recall che si concentra il guadagno. Il metodo raccoglie, in un unico grafo, le dipendenze corrette che le singole esecuzioni trovavano solo in parte e ogni volta in parte diversa (la cosiddetta «rotazione» degli archi tra run). La variabilità, che sul singolo run penalizza, diventa qui il meccanismo con cui si ritrova l'informazione dispersa.

V.2.2 La media dei pesi

Il voto di maggioranza, per come è stato costruito, tiene conto soltanto della ricorrenza di una dipendenza e ignora il peso p con cui il modello la propone. Il secondo metodo nasce dalla domanda lasciata aperta dal paragrafo precedente: tenere conto anche di quel peso, invece di scartarlo, cambia il risultato?

L'idea è di sostituire al conteggio una media. Per ogni possibile dipendenza, si calcola il peso medio che essa riceve nelle diverse esecuzioni e si costruisce così un unico grafo, i cui archi portano, al posto di un numero di voti, un valore continuo che riassume quanto spesso e con quanta forza quella dipendenza è stata estratta.

Il meccanismo si compone dei seguenti passaggi principali:

La *normalizzazione per esecuzione.* Prima di poter mediare i pesi occorre renderli confrontabili tra un'esecuzione e l'altra. Al modello si chiede di accompagnare ogni dipendenza con un punteggio p compreso tra 0 e 1, ma non si ha alcun controllo su come esso effettivamente utilizzi quella scala. Per questo, prima di aggregare, i pesi di ogni esecuzione vengono riscalati con una normalizzazione min-max, che porta a 0 il peso più basso prodotto in quel run e a 1 il più alto, distribuendo linearmente gli altri nell'intervallo[48]. In questo modo tutte le esecuzioni vengono ricondotte alla stessa scala e la media che segue mette a confronto quantità omogenee.

La *media sulle esecuzioni.* Per ciascuna coppia orientata A→B si calcola poi la media dei pesi normalizzati su tutte le N esecuzioni. Importante è far notare che, quando una run non contiene quella dipendenza, essa contribuisce alla media con un valore nullo. Il denominatore della media è sempre il numero totale di esecuzioni, non il numero di quelle in cui l'arco effettivamente compare. Ne segue che una dipendenza estratta di rado, per quanto alto sia il suo peso nelle poche run in cui appare, riceve comunque una media bassa perché trascinata verso il basso dagli zeri delle esecuzioni in cui è assente. Il valore medio così ottenuto combina

[48] La normalizzazione min-max riscala un insieme di valori nell'intervallo [0, 1] con la trasformazione (p - min)/(max - min), dove min e max sono il peso minimo e massimo prodotti dalla run considerata. Ne consegue che il peso più basso di ogni run viene portato a 0, confondendosi con l'assenza dell'arco; l'effetto è però trascurabile, perché riguarda quasi solo dipendenze estratte in una o due esecuzioni, che resterebbero comunque sotto qualsiasi soglia operativa.

quindi, in un'unica quantità, due informazioni: la frequenza con cui la dipendenza ricorre e la confidenza che il modello le assegna quando la produce. Il risultato è un grafo completo in cui a ogni arco è associato un peso medio compreso tra 0 e 1.

Soglia, orientamento e aciclicità. Successivamente, su questo grafo medio, si applica una soglia t, la quale rappresenta il valore del peso medio che l'arco deve raggiungere per essere tenuto. Si risolvono poi le coppie reciproche, mantenendo il verso con la media maggiore e scartando entrambi i versi in caso di parità e si rompono gli eventuali cicli togliendo l'arco di peso minore.

La soglia t svolge, sul peso medio, lo stesso ruolo che k svolgeva sul numero di voti: è la leva che regola il compromesso tra *precision* e *recall*. Abbassandola entrano nel grafo anche archi dal peso medio modesto, quindi qualche dipendenza corretta in più, ma soprattutto molte coppie spurie. Alzandola sopravvivono invece, solo le dipendenze con media elevata, cioè quelle al tempo stesso ricorrenti e ad alta confidenza. La differenza rispetto al voto è concettuale: mentre il voto guarda solo a quante volte una coppia compare, la media pondera quel conteggio con l'intensità dichiarata e, in linea di principio, potrebbe premiare una dipendenza un po' meno frequente ma sempre estratta con grande sicurezza, oppure penalizzarne una frequente ma debole. Se questa informazione si traduca davvero in un grafo diverso, è la domanda a cui si risponde confrontando i risultati dei due metodi.

Di seguito si riportano i risultati sulle stesse 30 run del voto al variare della soglia:

Soglia t	Precision	Recall	F1
0,00	23,24	75,00	35,48
0,05	37,50	68,18	48,39
0,10	50,00	54,55	52,17
0,15	**62,16**	**52,27**	**56,79**
0,20	67,74	47,73	56,00
0,25	75,00	40,91	52,94
0,30	75,00	34,09	46,87
0,35	72,22	29,55	41,94
0,40	66,67	13,64	22,64
0,45	80,00	9,09	16,33
0,50	100,00	6,82	12,77
0,55	100,00	2,27	4,44

Tab. 4: Media dei pesi su 30 esecuzioni (variazione del solo seme): precision, recall e F1 (%) al variare della soglia t sul peso medio. In grassetto la riga di F1 massimo.

L'esito è del tutto analogo: nel suo punto migliore l'F1 sale da circa il 31% della singola esecuzione al 56.79%. Per soglie basse il grafo medio è ampio e sbilanciato verso il recall; crescendo la soglia la precision sale, fino a toccare il 100% da t = 0,50 in su, mentre il recall si riduce, finché oltre t = 0,55 il grafo si svuota. Il miglior equilibrio è a t = 0,15, dove l'F1 raggiunge il 56,79% (con 23 dipendenze corrette su 44). Come per il voto, il guadagno rispetto alla singola esecuzione è netto e si concentra sul recall, che passa da circa il 28% a oltre il 52%.

V.2.3 I due metodi a confronto

Uno degli aspetti peculiari che emerge dal confronto è che i due metodi di aggregazione, pur così diversi nell'impianto, arrivano allo stesso risultato: nel loro punto migliore il voto di maggioranza raggiunge un F1 del 57,14% e la media dei pesi del 56,79%, con un equilibrio del tutto analogo tra precision e recall. La coincidenza non è affatto scontata, perché la media utilizza un'informazione che il voto ignora del tutto, eppure quell'informazione in più non si traduce in un grafo migliore. Vale allora la pena chiedersi il perché.

Confrontando più nel dettaglio i due grafi aggregati, si nota che essi recuperano in larga parte le stesse dipendenze corrette e commettono gli stessi errori: la medesima sovra-generazione a partire dagli stessi corsi-nodo e le stesse inversioni sistematiche del verso, di cui si dirà in seguito nell'analisi degli errori.

La ragione si comprende scomponendo che cosa sia, in concreto, il peso medio di un arco. Per costruzione esso è il prodotto di due fattori: il peso tipico che quell'arco riceve nelle esecuzioni in cui compare, moltiplicato per la frazione di esecuzioni in cui effettivamente compare. Questo secondo fattore non è altro che la frequenza dell'arco, ovvero esattamente la quantità che il voto di maggioranza conta. La domanda diventa allora quale dei due fattori guidi la media.

L'analisi dei run, sintetizzata visivamente nella Fig. 5, mostra che a dominare è la frequenza, per due motivi convergenti. Il primo è di scala: la frequenza spazia sull'intero intervallo dato dal numero di run eseguiti, mentre il peso tipico resta confinato in una banda medio-alta e ristretta, poiché il modello tende ad assegnare punteggi elevati. Il secondo motivo è che il peso tipico è di fatto scorrelato dalla frequenza e invece di aggiungere un ordinamento si comporta come rumore. Ne segue che ordinare gli archi per peso medio equivale in pratica a ordinarli per frequenza, che è ciò che fa il voto. Questo non significa che ogni arco riceva sempre lo stesso peso, il peso infatti varia, ma in una banda troppo stretta e in modo troppo slegato dalla frequenza per incidere sull'ordinamento, così che nella media finisce sopraffatto dal fattore frequenza.

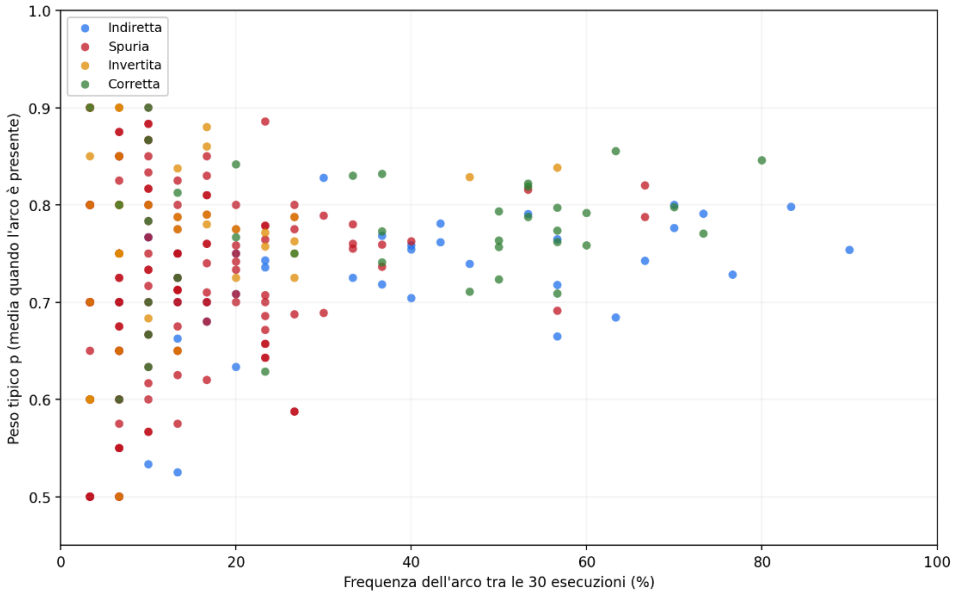


Fig. 6: Per ogni arco candidato, frequenza tra le 30 esecuzioni (asse x) e peso tipico assegnato dal modello quando l'arco è presente (asse y), distinti per tipo.

La vera differenza tra i due metodi si nota confrontando le tabelle 3 e 4.

Il voto presenta un intervallo di valori di k relativamente ampio in cui l'F1 resta elevato, mentre la media raggiunge il suo massimo in un intorno più stretto. Inoltre, nella media, normalizzare i pesi e mediarli ridistribuisce gli archi sull'intero intervallo [0, 1], così che la coda di archi spuri finisce in basso e una soglia riesce finalmente a separarli. Filtrare il peso grezzo invece, schiacciato verso l'alto, non separava nulla e produceva una curva piatta. È in questo senso che l'aggregare-poi-filtrare fa emergere una soglia capace di discriminare, dove prima non ce n'era una.

In conclusione, la scelta tra i due metodi di aggregazione non cambia la sostanza del risultato, poiché ciò che conta è aggregare molte esecuzioni e non il modo in cui lo si fa. Resta però un limite comune a entrambi, già leggibile in questi numeri. Nel punto di massimo il recall si ferma poco oltre la metà del grafo di riferimento, perché una parte delle dipendenze non viene quasi mai estratta. Su queste nessuna soglia e nessun metodo di aggregazione può intervenire. È il punto da cui muove il paragrafo successivo, dove si agisce sull’ordine dell'input.

V.3 Le prove di randomizzazione

Il capitolo precedente ha mostrato che l’aggregazione è la metodologia corretta da applicare, portando l'F1 da circa il 31% del singolo run a circa il 57% dell'ensemble.

Le esecuzioni aggregate presentate finora condividono però una scelta rimasta implicita e fissa, cioè l'ordine con cui i syllabus vengono presentati al modello.

Tale scelta non è neutra. La letteratura sui Transformer documenta un bias di posizione, per cui le prestazioni risultano più alte quando l'informazione rilevante si trova all'inizio o alla fine del contesto e degradano nella parte centrale[49]. Se questo valesse anche nel nostro caso, la sequenza con cui i 32 syllabus vengono presentati potrebbe influire su quali dipendenze il modello produce e su come le orienta. Randomizzare l'ordine di presentazione dei syllabus permetterebbe allora di togliere al risultato la dipendenza da un ordine particolare e, al tempo stesso, di aggiungere all'ensemble una seconda sorgente di diversità.

Gli esperimenti che seguono verificano se e in che modo questo incida sul risultato.

V.3.1 Randomizzazione dell'ordine dell'input

Il primo esperimento si concentra sulla randomizzazione del solo ordine dell'input, tenendo fisso tutto il resto, seed compreso (fissato al valore 42).

Ogni esecuzione è identificata da un numero progressivo, indicato con *r*, che svolge anche il ruolo di seme della permutazione. A partire da *r*, un generatore pseudo-casuale produce un riordino delle trentadue righe dell'elenco dei corsi e questa è la

[49] Fenomeno conosciuto come *lost in the middle*. Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P. (2024). *Lost in the Middle: How Language Models Use Long Contexts*. Transactions of the Association for Computational Linguistics (TACL), 12, 157–173.

sequenza con cui i corsi vengono forniti al modello in quella esecuzione. Poiché la permutazione dipende in modo deterministico da *r*, l'ordine di ciascuna esecuzione è al tempo stesso riproducibile e distinto da quello delle altre: l'esecuzione $r = 0$ riceve sempre la medesima permutazione, l'esecuzione $r = 1$ un'altra, e così via. Su ciascuna sequenza così ottenuta si esegue l'intera pipeline di estrazione, senza alcuna modifica al prompt, agli step o ai parametri.

Sono state lanciate 100 esecuzioni, con *r* da 0 a 99, in una prima giornata di esecuzione (luglio). La stessa procedura è stata replicata su altre cento permutazioni, con *r* da 100 a 199, in una diversa giornata (agosto)[50]. Infine, i due gruppi sono stati uniti e valutati su un unico grafo dato dalla loro unione.

Alcuni risultati particolari da segnalare sono casi in cui il modello è entrato in un ciclo di ripetizione e ha restituito un output vuoto o monocorde. Tali esecuzioni sono state escluse: nel gruppo di luglio nessuna, in quello di agosto la sola run_181, rimasta priva di dipendenze. L'insieme di agosto conta quindi novantanove esecuzioni valide e l'unione complessiva centonovantanove.

[50] Per chiarezza: usare indici nuovi garantisce ordini diversi da quelli di luglio, proprio perché la permutazione è determinata da *r*.

Su ciascuno dei tre insiemi sono stati applicati entrambi i metodi di aggregazione, facendo variare la soglia. I valori migliori di F1 sono riassunti di seguito.

Configurazione	Voto di maggioranza	Media dei pesi
Ordine, luglio (100 run)	65,93 (k = 0,15)	64,00 (0,10)
Ordine, agosto (99 run)	65,79 (k = 0,20)	62,16 (0,10)
Ordine, unione (199 run)	64,94 (k = 0,20)	64,86 (0,10)

Tab. 5: F1 migliore dei due metodi di aggregazione per ciascuna configurazione. Valori di F1 espressi in %. La soglia riportata tra parentesi è il valore (k per il voto di maggioranza, t per la media dei pesi) a cui si raggiunge l'F1 migliore.

Il confronto naturale è con l'ensemble costruito sulle esecuzioni non randomizzate: l'ensemble a ordine fisso si fermava a un F1 migliore di 57,14% con il voto e del 56,79% con la media. Randomizzando l'ordine dell'input, l'F1 migliore sale a circa il 65% con entrambi i metodi e su tutti gli insiemi.

Vale la pena chiarire che cosa, di preciso, la randomizzazione dell'ordine porti. Tale cambiamento non porta ad un ampliamento dell'insieme delle dipendenze recuperabili. Le poche dipendenze reali che nessuna esecuzione coglie restano infatti fuori anche randomizzando l'ordine, perché dipendono dai limiti del modello e non dalla posizione dei corsi nell'input. L'ordine è però di per sé una leva di diversità efficace e non secondaria. Tenendo il seme fisso, la sola permutazione dell'input porta la copertura dell'ensemble da circa un quarto delle dipendenze reali,

quante ne recupera in media un singolo run, a circa il 90% della loro unione. In questo senso l'ordine agisce come il seme, cioè come leve che alimentano l'aggregazione, cosa che l'esperimento successivo renderà esplicita.

Il vero contributo dell'ordine è il cambiamento di verso di alcune dipendenze: una sequenza di lettura fissa tende a orientare in modo sistematicamente errato certe coppie di corsi e variare l'ordine ne raddrizza una parte. È un effetto che verrà ripreso e analizzato nel paragrafo dedicato agli errori del modello.

Particolare attenzione merita poi, l'unione dei due gruppi. Passare da cento a centonovantanove esecuzioni non alza il tetto delle prestazioni, il voto di maggioranza resta a 64.94 e la media dei pesi a 64.86, in linea con i due gruppi presi singolarmente. Ciò che aumenta è la stabilità, in quanto più esecuzioni riducono la dipendenza dal singolo campione senza spostare il valore massimo raggiungibile. All'unione va riconosciuto anche un secondo pregio, quello di attenuare la variabilità tra giornate di esecuzione, un fattore capace altrove di far oscillare sensibilmente il risultato: aggregando permutazioni raccolte in giornate diverse l'effetto di giornata viene mediato anziché subito.

V.3.2 Randomizzazione di ordine e seme

Il secondo esperimento aggiunge un'ulteriore sorgente di variabilità: il seed.

In ogni esecuzione, oltre a variare l'ordine di input come in precedenza, si aggiunge la variazione del seme, estratto in modo casuale e registrato per ciascuna run.
Negli esperimenti precedenti, l'ensemble variava prima il solo seme del modello, tenendo fisso l'ordine dell'input; poi, variava il solo ordine, tenendo fisso il seme. Il passo naturale è quindi combinarle, lasciando che entrambe cambino a ogni esecuzione. La domanda a cui si vuole rispondere è se una seconda sorgente di diversità, indipendente dalla prima e aggiunta sopra alla randomizzazione dell'ordine, spinga più in alto il valore massimo delle prestazioni oppure se quello raggiunto finora rappresenti già il limite ottenibile con questo modello.

Per isolare l'effetto del solo seme aggiunto, gli ordini di input riprendono esattamente quelli di luglio, con *r* da 0 a 99. Come prima, anche qui un'unica esecuzione, la run_029, è stata esclusa perché monopolizzata da un solo corso sorgente, lasciando 99 esecuzioni valide. Combinandole e valutandole con entrambi i metodi di aggregazione, i valori migliori di F1 sono del 65,26% con il voto di maggioranza, alla soglia k = 0,15, e del 65,79% con la media dei pesi, alla soglia 0,10; nel voto il recall migliore arriva al 70,5%, con 31 dipendenze corrette su 44.
Confrontando i risultati dei due esperimenti si nota che i valori, sostanzialmente, coincidono tra loro: aggiungere la variazione del seme non alza il tetto delle prestazioni. Inoltre, i due metodi di aggregazione, che negli esperimenti precedenti mostravano differenze, sebbene contenute, qui quasi coincidono.

Il vero guadagno sta nell'aggiungere stabilità al risultato, rendendolo di fatto indipendente dalla tecnica di aggregazione. Si deduce quindi che il valore di circa il 65% non è l'effetto di una scelta particolare, un ordine fortunato o un metodo piuttosto che l'altro. L'aggregazione neutralizza la debolezza della singola esecuzione, che è essenzialmente variabilità; ciò che invece trattiene l'F1 intorno a quel valore è il comportamento sistematico del modello locale, che nessuna delle leve di randomizzazione riesce a modificare.

Di seguito vengono sintetizzati i risultati:

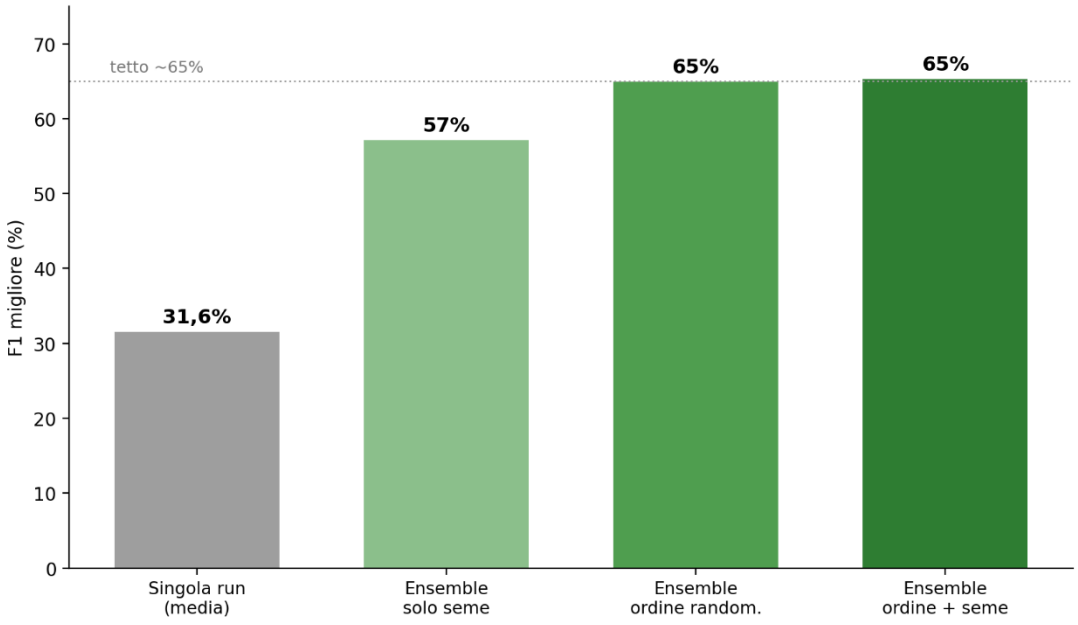


Fig. 7: F1 migliore lungo gli esperimenti - l'aggregazione produce il salto maggiore, le leve di randomizzazione portano al tetto, oltre il quale nessuna variazione spinge.

V.3.3 Sottocampionamento dell'input

Un ultimo esperimento ha esplorato una direzione diversa. Il punto di partenza è un'ipotesi sul perché la randomizzazione dell'ordine funzioni. L'idea è che il modello non sfrutti l'intero input, come se, superata una certa dimensione, una parte dei syllabus finisse per non essere più letta davvero. Se così fosse, fornire ogni volta tutti i corsi sarebbe in parte uno spreco e converrebbe dare al modello solo una porzione dell'input, cambiando di volta in volta quale. Da qui l'esperimento, con il duplice obiettivo di verificare se i tempi di esecuzione calino e se, nonostante il taglio, l'F1 dell'ensemble si mantenga.

I 32 syllabus vengono randomizzati come negli esperimenti precedenti, questa volta però ad ogni esecuzione se ne trattiene solo il 50%, corrispondente ai primi 16 corsi, gli altri vengono scartati per quella esecuzione. Al modello si dichiarano validi unicamente i codici dei corsi effettivamente forniti, così che una dipendenza possa emergere solo quando entrambi i corsi coinvolti sono presenti in input. Il procedimento viene ripetuto cento volte, un numero abbastanza grande affinché ogni coppia compaia insieme in un numero sufficiente di esecuzioni. Poiché a ogni run si mantiene una metà, due corsi specifici risultano entrambi presenti in circa un quarto delle run, in media intorno a ventiquattro volte su cento e mai meno di una decina, così che nessuna possibile dipendenza resti esclusa solo perché i corsi non sono mai capitati insieme. I 100 grafi ottenuti vengono aggregati con i due metodi

Il guadagno in termini di tempo c'è ma è modesto. Il tempo mediano per esecuzione scende da circa 269 a circa 178 secondi, una riduzione del 34% a fronte di un input dimezzato, segno che il costo dominante è la generazione delle dipendenze e non la lettura del prompt. Per quanto riguarda invece i risultati in termini di metriche, il confronto è nettamente negativo: riportando i valori sullo stesso denominatore usato con l'input completo, ovvero il numero totale di esecuzioni, il voto di maggioranza si ferma a un F1 migliore del 49,61% e la media dei pesi del 37,93%, entrambi nettamente sotto il circa 65% e 64% ottenuti con l'input completo. Questo esito dà una risposta netta alla domanda da cui l'esperimento nasce. Il calo dell'F1 è infatti controintuitivo rispetto all'ipotesi del taglio: se il modello, oltre una certa dimensione, ignorasse comunque parte dell'input, fornirgliene direttamente solo metà non dovrebbe penalizzarlo. La perdita osservata indica invece che l'estrazione sfrutta più della metà dei corsi disponibili. L'esperimento fornisce quindi una conferma indiretta che in questo caso l'input non viene troncato, circoscrivendo l'interpretazione del perché la randomizzazione dell'ordine funzioni, escludendo che il guadagno dipenda semplicemente da una porzione di input mai letta[51].

[51] Un test pienamente equo richiederebbe un numero di esecuzioni molto più elevato. Con il campione ridotto ogni coppia di corsi è valutata solo in circa un quarto dei run, e servirebbero perciò molte più ripetizioni per accumulare, coppia per coppia, la stessa evidenza dell'input completo; la riduzione del tempo per esecuzione renderebbe praticabile questo aumento. L'esperimento è stato momentaneamente archiviato, lasciando la verifica a un lavoro successivo.

V.3.4 Robustezza rispetto al modello

Gli esperimenti condotti finora sono stati eseguiti con lo stesso modello, gpt-oss:20b. A questo punto bisogna chiedersi se i risultati raggiunti siano una proprietà del *metodo* o dipendano dalle capacità del *modello* specifico. Per rispondere, l'intera serie di esperimenti sulla randomizzazione (V.3.1–V.3.3), è stata eseguita con un modello più potente e performante, gemma4:26b.

Di seguito si riportano i valori migliori di F1 dei tre metodi per ciascun esperimento, affiancando i due modelli:

Configurazione	**Classica (gemma / gpt-oss)**	**Media dei pesi (gemma / gpt-oss)**	**Voto di maggioranza (gemma / gpt-oss)**
Ordine (0-99)	47,25 / 28,26	56,10 / 64,00	60,00 / 65,93
Ordine (100-199)	45,79 / 28,65	53,66 / 62,16	60,95 / 65,79
Ordine + seme	46,38 / 28,24	56,10 / 65,79	59,32 / 65,26
Sottocampionamento 50%	20,42 / 14,25	44,78 / 37,93	60,00 / 49,61

Tab. 6: Picchi di F1 (%) a confronto tra i due modelli, gemma4:26b / gpt-oss:20b, per configurazione e metodo di aggregazione.

Sulla singola esecuzione (valutazione classica) gemma è nettamente superiore: l'F1 medio passa da circa 28 a circa 47 punti sugli esperimenti a input completo, con una variabilità tra run più bassa (deviazione standard di 8 punti contro gli 11–12 di gpt-oss). Ci si aspetterebbe che questo vantaggio si trasferisca all'aggregazione ma

invece, sull'ensemble a input completo, è gpt-oss a restare davanti, sia con il voto di maggioranza (60 contro 66) sia con la media dei pesi (54–56 contro 62–66).

Il perché di questi risultati risiede nel meccanismo dell'ensemble: l'aggregazione rende di più soprattutto quando le esecuzioni sono diverse e gli errori indipendenti. Combinando molte esecuzioni rumorose ma scorrelate, gli archi corretti che ricorrono spesso si rafforzano, mentre gli errori casuali, sempre diversi da una run all'altra, vengono eliminati. Gemma, essendo un modello più performante, produce esecuzioni più stabili e simili tra loro (l'unione degli archi distinti prodotti sulle 100 run è di circa 210, contro i circa 500 di gpt-oss). C'è quindi meno diversità da sfruttare e infatti il guadagno dell'aggregazione rispetto alla singola run risulta minore (circa +13 punti per gemma contro +38 di gpt-oss).

Nell'esperimento sul sottocampionamento invece, è gemma a prevalere anche in ensemble. Ciò non è dovuto però ad un miglioramento del modello bensì ad un crollo di gpt-oss. Quando l'input si riduce a 16 corsi su 32, una dipendenza può emergere solo nelle run in cui entrambi sono stati campionati. Di conseguenza, il supporto massimo raggiungibile di ogni arco si abbassa e viene meno l'alto supporto degli archi corretti, cioè ciò su cui si fondava il vantaggio. In tale situazione, ogni esecuzione è un sotto-problema più piccolo e autonomo e a contare maggiormente in questi casi è l'accuratezza della singola run, dove gemma è più forte.

L’esecuzione della pipeline su un modello più capace conferma l’ensemble come strumento centrale del metodo. Tuttavia, mostra anche che una singola run più stabile e precisa non si traduce in un ensemble migliore. La qualità della singola esecuzione e il guadagno ottenibile dall'aggregazione sono governati da fattori distinti: la prima dipende dall'accuratezza del modello, il secondo dalla diversità e dall'indipendenza degli errori tra le esecuzioni. L'ensemble è una leva di riduzione della varianza, che rende meglio su un modello debole e diversificato piuttosto che su uno forte e coerente. Per questo un modello dalla singola run più forte può produrre un ensemble dal tetto lievemente più basso, pur partendo da una copertura di archi veri superiore (42/44 per gemma vs 40/44 per gpt-oss).

V.4 Analisi degli errori

Il livello più alto raggiunto di F1 è pari a circa il 65% (ottenuto nell’esperimento sulla randomizzazione dell’input). Si tratta di un valore che si è rivelato stabile rispetto al numero di esecuzioni (unione delle 199 run), alle variazioni (randomizzazione del seme) e alla tecnica di aggregazione (voto o media). Tale soglia può essere ricondotta ad un comportamento sistematico del modello, che si presenta con tre manifestazioni particolari: un insieme di dipendenze che il modello non riesce mai a trovare; un piccolo numero di archi orientati in modo sistematicamente errato ed infine, la tendenza a generare archi in eccesso.

Nel grafo di riferimento in Fig. 7, ogni dipendenza è colorata secondo il comportamento del modello:

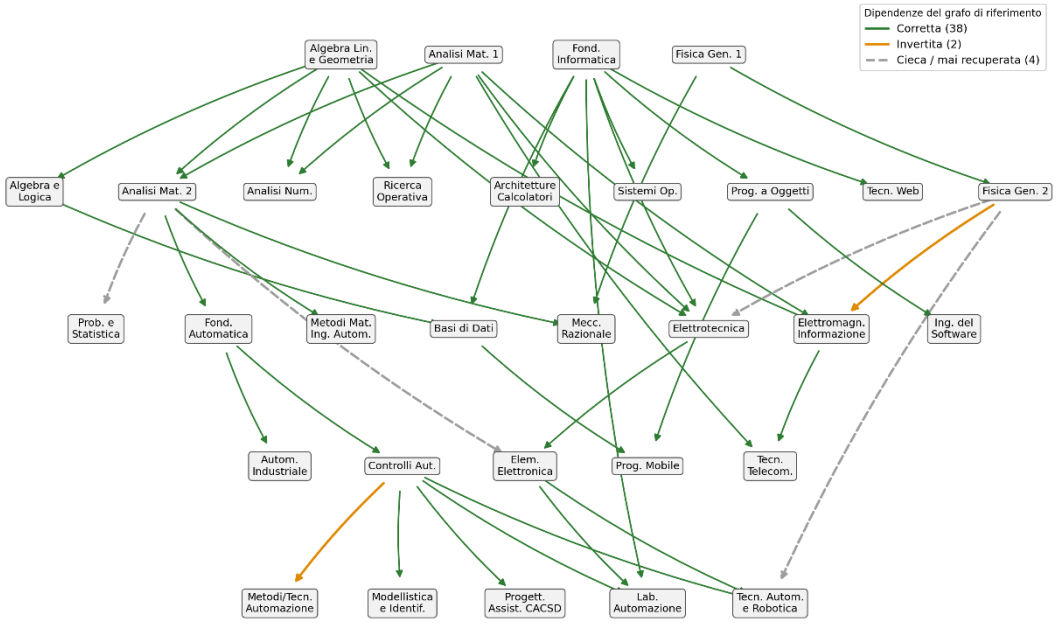


Fig. 8: Grafo dei docenti secondo i risultati del modello

V.4.1 Le dipendenze mai recuperate

Il primo limite riguarda un piccolo residuo di dipendenze reali che nessuna esecuzione, in nessun ordine e con nessun seme, riesce a produrre. Su quarantaquattro dipendenze del grafo di riferimento, quattro non compaiono mai (evidenziate in grigio nella Fig. 7 e indicate per esteso nella Tab. 7).

Per risalire alla causa delle omissioni, la prima analisi ha riguardato la posizione dei corsi nell'elenco fornito in input, sul presupposto che quelli collocati nella parte centrale, penalizzata dal bias di posizione, venissero trascurati. La copertura corso

per corso però smentisce: i corsi coinvolti non si trovano in coda e la frequenza con cui compaiono non decresce con la posizione nell'elenco.

La verifica si è allora spostata sul contenuto dei programmi, dove emerge lo stesso tratto: prerequisito reale sul piano concettuale, privo di riscontro esplicito nel testo.

Dipendenza cieca	Estratto dal syllabus	Osservazione
Analisi matematica 2 → Elementi di elettronica	**Analisi 2:** *functions of several variables; (...) ordinary differential equations; Laplace transform.* **Elettronica:** *number systems and codes; boolean algebra; combinatorial and sequential networks; (...) fundamentals of solid-state electronics: (...)*	Il prerequisito è reale (l'analisi dei circuiti e dei loro transitori poggia su equazioni differenziali e trasformata di Laplace) ma il programma di Elementi, non le nomina: nessun termine in comune.
Analisi matematica 2 → Probabilità e statistica	**Probabilità:** *probability spaces; discrete and continuous random variables; convergence: law of large numbers and central limit theory; mathematical statistics: (...)*	Le variabili aleatorie continue e i teoremi di convergenza presuppongono gli integrali multipli e le funzioni di più variabili, ma la parola «integrale» non compare: legame matematico genuino, privo di traccia testuale.
Fisica generale 2 → Tecnologie per l'automazione e la robotica	***Fisica 2:*** *electric and magnetic phenomena in vacuum, stationary and non-stationary case; geometric optics and basic wave optics.* ***Tecnologie****: introduction to process control;control system architecture; static and dynamic analysis; power semiconductor devices; (...).*	Gli azionamenti elettrici, i motori e i convertitori di potenza delle Tecnologie presuppongono i fenomeni elettromagnetici di Fisica 2, ma il ponte tra i due programmi è concettuale e non lessicale: il modello non lo attraversa.
Fisica generale 2 → Elettrotecnica	***Elettrotecnica****: Part I: elementary bipoles, , time-invariant,memoryless circuits (...); circuit analysis based on mesh currents and on node*	La teoria dei circuiti dell'elettrotecnica presuppone i fenomeni elettromagnetici di Fisica 2, ma il programma resta

	potentials; Thévenin's and Norton's theorems; Part II: non-linear dynamical circuits	interamente sul piano circuitale e non li nomina mai: il legame, reale, rimane implicito.

Tab. 7: Le quattro dipendenze cieche, con l'estratto rilevante del syllabus

A parte va segnalata anche fondamenti di informatica → elettrotecnica, una quinta dipendenza che il modello non manca del tutto ma produce solo in pochissime esecuzioni. Portata all'attenzione di un docente, è risultata discutibile come voce del grafo di riferimento: l'unico punto di contatto tra i due programmi è un accenno marginale alla programmazione algoritmica in MATLAB, troppo tenue per sostenere da solo una propedeuticità. È l'unico caso in cui alla possibile mancanza del modello si somma un'incertezza del riferimento stesso.

Da questa analisi si traggono due conclusioni. La prima è che il recall non è il problema principale del metodo: ha margine, e l'aggregazione lo sfrutta. La seconda è che le quattro dipendenze cieche sono tutte propedeuticità corrette, la cui assenza è una mancanza del modello e non un difetto del riferimento.

Sul perché il modello le manchi si può dire meno. Con certezza si può escludere la posizione nell'input, mentre il tratto comune sembra essere la natura concettuale, e non lessicale, del legame. Questa però resta un'osservazione e non una spiegazione dimostrata, in quanto in altri casi il modello ha mancato, almeno a ordine fisso, anche legami con ampio vocabolario condiviso. Un’analisi più solida richiederebbe un confronto sul contenuto semantico dei programmi, qui non condotto

V.4.2 Le inversioni di verso

Il secondo limite è di natura diversa: il modello individua la coppia di corsi giusta ma la orienta al contrario. Poiché la valutazione è direzionale, cioè un arco conta come corretto solo se riproduce il verso del grafo di riferimento, un'inversione pesa doppio perché è sia una dipendenza corretta mancata, sia un arco spurio generato.

Il fenomeno è circoscritto a solo due dipendenze che il modello sbaglia in modo sistematico: *Fisica generale 2 → Elettromagnetismo per la trasmissione dell'informazione* e *Fondamenti di automatica → controlli automatici*, entrambe formate da un corso di base e dalla sua prosecuzione avanzata. Nelle esecuzioni con l'ordine di input fisso il verso sbagliato è un errore ricorrente: la prima coppia viene prodotta girata nel 42% delle esecuzioni contro l'11% nel verso giusto, la seconda addirittura nel 62% contro il 16%.

La causa più immediata riguarda l'ordine con cui i corsi vengono letti in input. Il modello sembra orientare la freccia dal corso che incontra prima a quello che incontra dopo e le due coppie hanno in effetti la particolarità che, nella sequenza in cui i corsi vengono presentati, il prerequisito vero compare dopo il corso da cui dovrebbe dipendere: elettromagnetismo si trova in 4° posizione e fisica in 6°, controlli automatici 13° e fondamenti di automatica 17°. In entrambi i casi, quindi, seguire l'ordine di lettura porterebbe alla freccia rovesciata. Quanto però l'ordine pesi davvero lo verifica l'esperimento di randomizzazione.

Mescolando l'input, ogni dipendenza viene letta in entrambe le sequenze lungo le cento esecuzioni. Per fondamenti di automatica e controlli automatici, l'ipotesi dell'ordine trova conferma: il verso corretto torna a essere quello prevalente (39 esecuzioni contro 16), segno che l'inversione era effettivamente indotta dalla sola sequenza di lettura. Per fisica generale 2 ed elettromagnetismo, invece, il verso sbagliato resta prevalente anche mescolando l'input (36 contro 18) e perfino restringendo l'analisi alle sole esecuzioni in cui fisica 2 viene letta per prima. In questo caso, quindi, la causa non è l'ordine, ma un bias interno del modello, che nessuna permutazione dell'input riesce a correggere. Nel grafo aggregato finale (Fig. 7) restano così in arancione due inversioni: Fisica 2 → Elettromagnetismo, di natura intrinseca, e Controlli automatici → Metodi e tecniche, un caso quasi in parità che il voto risolve nel verso sbagliato.

Resta da spiegare perché a queste inversioni non si possa porre rimedio con il peso. La ragione, già emersa altrove, è che il peso p è pressoché identico nei due orientamenti di una stessa coppia: gli archi girati arrivano con una confidenza intorno a 0.77, indistinguibile da quella, tra 0.75 e 0.80, dei corretti. Il peso misura la forza dell'associazione tra i due corsi, che è simmetrica, non la correttezza della direzione, e nessuna soglia può quindi separare il verso giusto dal suo rovescio. È per questo che l'unico strumento efficace sulle inversioni è quello strutturale, come la randomizzazione dell'ordine unita al voto sulla direzione, che però, come si è visto, agisce solo su quelle indotte dall'ordine e lascia intatte quelle intrinseche.

V.4.3 L'over-generation

L'ultima osservazione da fare rimane quella sulla sovra-generazione, che impatta soprattutto sulla *precision*. Il modello produce in media una quarantina di dipendenze per esecuzione, ma solo poco più del 40% di esse è corretto (la precision del singolo run): la maggior parte sono archi spuri, in larga parte fra corsi che nel grafo di riferimento non hanno alcuna relazione, né diretta né transitiva.

Esaminando più da vicino il fenomeno, si nota che l'over-generation tende a concentrarsi su pochi corsi. Questi hanno la caratteristica di essere quelli che occupano una posizione centrale nel percorso di studi, con un programma ampio e trasversale, fatto di argomenti di base che ricorrono in molti altri insegnamenti. Rientrano in questa categoria i fondamentali di matematica (analisi matematica 1, algebra lineare e geometria), ma il caso più marcato è programmazione ad oggetti. Ciò che questi corsi hanno in comune è di trattare temi vicini a quelli di molti altri insegnamenti, ed è verosimilmente questa affinità tematica diffusa a spingere il modello a generare dipendenze in eccesso proprio a partire da essi.

Prendiamo come esempio programmazione ad oggetti.

Il suo programma si articola in due parti: la programmazione a oggetti vera e propria e un blocco di strutture dati e algoritmi. È soprattutto questa seconda parte a poterne spiegare il comportamento: si tratta di contenuti generali dell'informatica, che fanno da base a un gran numero di insegnamenti dell'area. Una lettura plausibile è allora

che il modello, davanti a un programma così vicino per tema a mezza area informatica, colga questa vicinanza e la scambi per una dipendenza, tracciando un arco verso ciascuno di quei corsi anche là dove una propedeuticità non esiste. Il meccanismo preciso, però, non è stato isolato e quanto detto è perciò una lettura plausibile più che una causa dimostrata.

Di seguito si riportano gli archi spuri prodotti da programmazione ad oggetti e dagli altri corsi hub[52]:

Sorgente	Destinazione	Freq.
Programmazione ad oggetti	Sistemi operativi	46/100
Algebra lineare e geometria	Fisica generale 1	37/100
Analisi matematica 1	Fisica generale 1	36/100
Programmazione ad oggetti	Laboratorio di automazione	28/100
Programmazione ad oggetti	Tecnologie web	25/100
Programmazione ad oggetti	Automazione industriale	21/100
Analisi matematica 2	Elettrotecnica	21/100
Metodi per l'automazione	Progettazione assistita	18/100
Elettromagnetismo	Tecn. per automaz. e robotica	18/100
Analisi matematica 1	Algebra e logica	16/100
Algebra lineare e geometria	Analisi matematica 1	11/100

Tab. 8: Archi spuri del grafo aggregato sulle 0-99 esecuzioni (voto, k = 0.20[53])

[52] In teoria dei grafi un *hub* è un nodo con un numero di connessioni molto superiore alla media. Qui indica un corso da cui, nel grafo prodotto, si diparte un numero di archi (dipendenze uscenti) nettamente più alto che dagli altri.

[53] La soglia *k* agisce sul supporto di coppia (le esecuzioni in cui i due corsi compaiono insieme, in un verso qualsiasi). La colonna «Freq.» è invece la frequenza del solo verso indicato, per questo alcuni valori sono inferiori al supporto minimo (0,20 x 100 = 20)

Sul comportamento di questi archi in fase di aggregazione serve essere precisi, perché è qui che si capisce cosa l'ensemble corregge e cosa non tocca. Molti archi spuri sono legati a un particolare ordine di lettura: a input fisso ricorrono con alta frequenza, ma randomizzando la loro frequenza cala e si disperdono, così che il consenso li lascia fuori. Gli archi dei corsi hub, invece, sono indipendenti dall'ordine: compaiono in una frazione ampia e stabile qualunque sia la sequenza dell'input, con una frequenza paragonabile a quella delle dipendenze autentiche. Per questo non si disperdono e superano la soglia, arrivando al grafo finale. È il motivo per cui non si possono togliere alzando semplicemente la soglia: lo stesso vicolo cieco già incontrato con il peso, che a questi archi spuri assegna una confidenza alta esattamente come a quelli corretti. Sul grafo aggregato finale, rimangono 25 dipendenze corrette e 28 indirette, mentre gli archi errati sono soltanto 13 (le due inversioni già discusse e undici di over-generation pura, quattro delle quali ancora da programmazione ad oggetti). La precision effettiva sale così a circa il 66%.

A margine va segnalato un altro fenomeno raro ma caratteristico, che con l'over-generation condivide la radice. Circa una volta su cento, una run degenera in un ciclo centrato proprio su un corso di area matematica molto connesso, di solito analisi matematica 1 o algebra lineare: il modello si auto-condiziona e produce un lungo blocco di righe tutte con lo stesso corso come sorgente. Queste esecuzioni gonfierebbero le uscite di quel corso e per questo vengono riconosciute ed escluse.

CONCLUSIONI

Giungendo al termine del lavoro, il risultato più immediato è un numero: l'F1 passa dal 28% di una singola esecuzione al 65% dell'ensemble. È il dato che riassume il guadagno dell'aggregazione ed è la prova che una pipeline nata per un modello commerciale in cloud può funzionare anche su un LLM open-weights locale. Ridurre l'analisi a questo numero, però, significherebbe limitare ciò che invece ha davvero messo in luce: come cambia il problema quando il modello con cui si lavora è piccolo e debole, e che cosa, in quel caso, si può ragionevolmente pretendere.

Cosa mostrano i risultati

Il problema riscontrato fin da subito è la forte variabilità del modello locale: esecuzioni con lo stesso input restituiscono grafi diversi e l'esito di un singolo run è in buona parte una questione di fortuna. Lo stesso tratto, però, è la condizione che rende possibile l'applicazione dell'ensemble. Poiché ogni esecuzione sbaglia a modo suo e coglie una porzione diversa delle dipendenze corrette, aggregarne molte ricompone un'informazione che nessuna possiede per intero: da circa un quarto del grafo recuperato in media da un run, all'unione che ne copre circa il 90%.

Questo "vantaggio" ha però un confine preciso. L'aggregazione corregge l'errore accidentale, quello che cambia da un run all'altro e mediando si annulla, ma non tocca l'errore sistematico, quello che tutte le esecuzioni condividono.

Per questo, esaurita la variabilità, il risultato si arresta intorno al 65% e nessuna leva (più esecuzioni, un secondo seme, un diverso metodo di aggregazione) lo sposta. Quel tetto è una proprietà del modello e conoscerne l'esistenza è importante: dice fin dove l'aggregazione può arrivare e dove comincia ciò che essa non risolve.
Proprio quel residuo sistematico è interessante da analizzare. Il modello non trova mai le stesse propedeuticità e inverte spesso le medesime coppie. Anche la sovra-generazione parte da alcuni soliti corsi centrali. Il segnale più chiaro di questa regolarità è il peso p, che il modello dichiara come confidenza ma che si comporta come una misura di associazione: risulta alto anche quando gli archi sono sbagliati oppure quasi identico in entrambi i versi di una stessa coppia. La conseguenza pratica positiva è che se gli errori hanno una certa ricorrenza, quella forma si può prevedere, rendendo lo strumento “leggibile” anche quando sbaglia.

Va infine ricordato il senso della scelta locale, che i numeri da soli tendono a nascondere. Rispetto all'80,43% riportato dal lavoro sul modello in cloud, il 65% ottenuto qui può sembrare un passo indietro, ma i due valori non misurano la stessa cosa. Uno è l'esito di una singola esecuzione, l'altro un risultato aggregato e stabile, robusto rispetto al seme, all'ordine e alla tecnica di aggregazione. Soprattutto, paragonare i due esiti sulla sola base di una metrica sarebbe fuorviante: un modello in cloud conta miliardi di parametri e una capacità di calcolo non paragonabile a quella di un modello eseguito su una singola macchina, e attendersi dal secondo lo

stesso valore del primo significherebbe ignorare questa distanza. L'obiettivo, in questa sede, non è raggiungere quel numero, ma comprendere e caratterizzare il comportamento di un modello più debole. È un punto di partenza, da cui lavori futuri possono partire per rendere l'approccio locale sempre più competitivo.

Limiti del lavoro

Bisogna riconoscere alcuni limiti del lavoro presentato.

Primo fra tutti, le soglie di aggregazione sono scelte a posteriori, prendendo sul grafo di riferimento il valore che restituisce l'F1 migliore. È un criterio che richiede di conoscere già le dipendenze corrette e che quindi non sarebbe applicabile a un nuovo corso di laurea, ancora privo di un grafo di riferimento con cui confrontarsi. Il caso di studio è un solo corso di laurea, un grafo di quarantaquattro archi, un modello; e lo stesso riferimento non è infallibile (una delle sue voci si è rivelata discutibile all'esame di un docente), per cui le metriche di valutazione si misurano rispetto a un termine di paragone a sua volta costruito. Infine, la spiegazione data per le dipendenze mai trovate resta un'ipotesi: distinguere un legame concettuale dalla semplice assenza di parole condivise richiederebbe un confronto semantico tra i programmi, che qui non è stato svolto.

Prospettive e applicazioni

Alcuni possibili sviluppi del lavoro nascono proprio dai limiti appena descritti.

Innanzitutto, bisognerà trovare un modo per guidare il modello a scegliere le dipendenze basandosi anche sul verso, il segnale direzionale e concettuale che attualmente manca al peso *p*.

A questo si affianca la possibilità, che l'esecuzione locale apre e che questo lavoro non ha toccato, di arricchire l'input con materiale riservato, come dispense, slide o estratti di libri di testo, così da fornire quel contesto che ai soli syllabus talvolta manca.

Sul piano applicativo, questi risultati suggeriscono un ruolo preciso per il metodo. Allo stato attuale, più che restituire un grafo finito da correggere, esso può ordinare le dipendenze candidate per affidabilità e portare all'attenzione di un docente solo quelle incerte (ad esempio le coppie di verso ambiguo), riducendo la revisione manuale a una frazione dei casi.

Più in generale, un grafo di propedeuticità affidabile è la struttura su cui si costruiscono i sistemi di raccomandazione dei percorsi di studio e gli ambienti di apprendimento adattivo, che di una simile mappa hanno bisogno. Poterla generare in locale, a costo nullo, sui dati riservati dell'istituzione e senza dipendere da un fornitore esterno, rende l'approccio particolarmente adatto ai contesti in cui l'offerta formativa cambia con frequenza e una revisione manuale a ogni aggiornamento sarebbe impraticabile.

BIBLIOGRAFIA

Aho, A. V., Garey, M. R., Ullman, J. D. (1972). *The Transitive Reduction of a Directed Graph*. SIAM Journal on Computing, 131–137. DOI: 10.1137/0201008.

Awasthi, R., Shukla, A., Agrawal, A. K., Dubey, P., Ramasamy, R. K. (2026). *A Hybrid Transformer–Graph Framework for Curriculum Sequencing and Prerequisite Optimization in Computer Science Education with Explainable AI*. Algorithms, 19(4), 308. DOI: 10.3390/a19040308.

Bogarín, A., Cerezo, R., Romero, C. (2018). *A survey on educational process mining*. WIREs Data Mining and Knowledge Discovery, 8(1), e1230. DOI: 10.1002/widm.1230.

Breiman, L. (1996). *Bagging Predictors*. Machine Learning, 24(2), 123–140. DOI: 10.1007/BF00058655.

Brown, T. B., Mann, B., Ryder, N., et al. (2020). *Language Models are Few-Shot Learners*. Advances in Neural Information Processing Systems 33 (NeurIPS 2020). arXiv:2005.14165.

Calegari, D., Delgado, A. (2025). *A Systematic Review on Process Mining for Curricular Analysis*. WIREs Data Mining and Knowledge Discovery, 15. DOI: 10.1002/widm.70005.

Diamantini, C., Gobbi, C., Mele, A., Potena, D., Rossetti, C. (2025). *Leveraging LLMs to Discover Causal Dependencies: A Case Study on a University Program*.

In: J. Grabis, Y. Wautelet (Eds.), CAiSE 2025 Workshops, Lecture Notes in Business Information Processing (LNBIP), vol. 556, pp. 237–248. Springer. DOI: 10.1007/978-3-031-94931-9_19.

Dietterich, T. G. (2000). *Ensemble Methods in Machine Learning*. In: Kittler, J., Roli, F. (Eds.), Multiple Classifier Systems (MCS 2000), Lecture Notes in Computer Science, vol. 1857, pp. 1–15. Springer. DOI: 10.1007/3-540-45014-9_1.

Holtzman, A., Buys, J., Du, L., Forbes, M., Choi, Y. (2020). *The Curious Case of Neural Text Degeneration*. International Conference on Learning Representations (ICLR). arXiv:1904.09751.

Huang, J., et al. (2024). *Large Language Models Cannot Self-Correct Reasoning Yet*. International Conference on Learning Representations (ICLR 2024). arXiv:2310.01798.

Jurafsky, D., Martin, J. H. (2024). *Speech and Language Processing*, 3ª ed. (bozza).

Kıcıman, E., Ness, R., Sharma, A., Tan, C. (2023). *Causal Reasoning and Large Language Models: Opening a New Frontier for Causality*. Transactions on Machine Learning Research. arXiv:2305.00050.

Liang, C., Wu, Z., Huang, W., Giles, C. L. (2015). *Measuring Prerequisite Relations Among Concepts*. Proceedings of EMNLP 2015, pp. 1668–1674.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P. (2024). *Lost in the Middle: How Language Models Use Long Contexts*. Transactions of the Association for Computational Linguistics, 12, 157–173.

OpenAI (2025). *gpt-oss-120b & gpt-oss-20b Model Card*. arXiv:2508.10925.

Pan, L., Li, C., Li, J., Tang, J. (2017). *Prerequisite Relation Learning for Concepts in MOOCs*. Proceedings of ACL 2017, pp. 1447–1456.

Salvatori, M. (Tesi di Laurea Magistrale, UNIVPM - 2024–2025). *Definizione di un grafo di dipendenze tramite LLM, con applicazione a corsi universitari*.

Van der Aalst, W. M. P. (2016). *Process Mining: Data Science in Action*, 2ª ed.

Van der Aalst, W. M. P. (2019). *A practitioner's guide to process mining: Limitations of the directly-follows graph*. Procedia Computer Science, 164, 321–328. DOI: 10.1016/j.procs.2019.12.189.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. (2017). *Attention Is All You Need*. Advances in Neural Information Processing Systems 30 (NeurIPS). arXiv:1706.03762.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., Zhou, D. (2023). *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. ICLR. arXiv:2203.11171.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. Advances in Neural Information Processing Systems 35 (NeurIPS). arXiv:2201.11903.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang,

P., Dean, J., Fedus, W. (2022). *Emergent Abilities of Large Language Models*. Transactions on Machine Learning Research. arXiv:2206.07682.

Yang, T., Ren, B., Gu, C., He, T., Ma, B., Konomi, S. (2025). *Leveraging LLMs for Automated Extraction and Structuring of Educational Concepts and Relationships*. Machine Learning and Knowledge Extraction, 7(3), 103. DOI: 10.3390/make7030103.

Zheng, H. S., Mishra, S., Chen, X., Cheng, H. T., Chi, E. H., Le, Q. V., Zhou, D. (2024). *Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models*. ICLR. arXiv:2310.06117.

SITOGRAFIA

Ollama, *Ollama* — https://ollama.com

Ollama, *gpt-oss* (libreria dei modelli), https://ollama.com/library/gpt-oss

Ollama, *Modelfile Reference*, https://docs.ollama.com/modelfile

Ollama, *API Reference*, https://docs.ollama.com/api

OpenAI, *gpt-oss-20b* — https://huggingface.co/openai/gpt-oss-20b

OpenAI, *harmony — Renderer for the harmony response format to be used with gpt-oss* (repository ufficiale), https://github.com/openai/harmony

OpenAI, *OpenAI Harmony Response Format*, OpenAI Cookbook, https://cookbook.openai.com/articles/openai-harmony

ggml-org, *llama.cpp* (repository ufficiale), https://github.com/ggml-org/llama.cpp

NVIDIA, *Mastering LLM Techniques: Text Data Processing*, 2024, https://developer.nvidia.com/blog/mastering-llm-techniques-data-preprocessing/

He, H. & Thinking Machines Lab, *Defeating Nondeterminism in LLM Inference*, https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/

International Educational Data Mining Society, *Educational Data Mining* (definizione della disciplina), https://educationaldatamining.org

Università Politecnica delle Marche, *Guida dello studente - Corso di Laurea in Ingegneria Informatica e dell'Automazione (IT04), a.a. 2023/2024*, https://guide.univpm.it/guide.php?lang=langita&fac=ingegneria&aa=2023&cds=IT04&cerca=cerca

APPENDICE A

Si riporta la versione aciclica del grafo dei docenti adottata come riferimento in tutte le valutazioni (44 dipendenze dirette tra i 32 insegnamenti del corso di laurea). Mentre la Figura 2 mostrata nel III.3 ne fornisce una rappresentazione illustrativa, la seguente tabella ne elenca il contenuto per esteso: per ciascun insegnamento sono indicati i suoi prerequisiti diretti, così da rendere verificabile e riproducibile il calcolo delle metriche.

Insegnamento	Prerequisiti diretti
Algebra e logica	Algebra lineare e geometria
Analisi matematica 2	Algebra lineare e geometria; Analisi matematica 1
Analisi numerica	Algebra lineare e geometria; Analisi matematica 1
Architetture dei calcolatori e cloud computing	Fondamenti di informatica
Automazione industriale	Fondamenti di automatica
Basi di dati	Algebra e logica; Fondamenti di informatica
Calcolo delle probabilità e statistica matematica	Analisi matematica 2
Controlli automatici	Fondamenti di automatica
Elementi di elettronica	Analisi matematica 2; Elettrotecnica
Elettromagnetismo per la trasmissione dell'informazione	Algebra lineare e geometria; Analisi matematica 1; Fisica generale 2
Elettrotecnica	Algebra lineare e geometria; Analisi matematica 1; Fisica generale 2; Fondamenti di informatica
Fisica generale 2	Fisica generale 1
Fondamenti di automatica	Analisi matematica 2

Ingegneria del software	Programmazione ad oggetti
Laboratorio di automazione	Controlli automatici; Elementi di elettronica; Fondamenti di informatica
Meccanica razionale	Analisi matematica 2; Fisica generale 1
Metodi e tecniche per l'automazione	Controlli automatici
Metodi matematici per l'ingegneria dell'automazione	Analisi matematica 2
Modellistica e identificazione dei processi dinamici	Controlli automatici
Progettazione assistita da calcolatore dei sistemi di controllo	Controlli automatici
Programmazione ad oggetti	Fondamenti di informatica
Programmazione mobile	Basi di dati; Programmazione ad oggetti
Ricerca operativa	Algebra lineare e geometria; Analisi matematica 1
Sistemi operativi	Fondamenti di informatica
Tecnologie per l'automazione e la robotica	Controlli automatici; Elementi di elettronica; Fisica generale 2
Tecnologie per le telecomunicazioni	Analisi matematica 1; Elettromagnetismo per la trasmissione dell'informazione
Tecnologie web	Fondamenti di informatica