UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

TESI DI LAUREA

Riproducibilità, analisi comparativa e stabilità dei modelli neurali per la generazione automatica di messaggi di commit

RELATORE

Prof. Filomena Ferrucci

Università degli Studi di Salerno

CANDIDATA

Valentina Annunziata

Matricola: 0522501687

Anno Accademico 2024-2025

Questa tesi è stata realizzata nel

Dedica o citazione

Abstract

La generazione automatica di messaggi di commit è un compito chiave nella documentazione del codice, ma è spesso trascurato dagli sviluppatori per motivi di tempo o disattenzione. Questa tesi analizza in modo comparativo tre modelli rappresentativi di questo task: CoDiSum, RACE e KADEL, selezionati per coprire l'evoluzione architetturale fino ad oggi. Il lavoro si concentra su tre aspetti fondamentali: la riproducibilità dei risultati, il confronto sperimentale tra approcci moderni e la stabilità delle prestazioni tra diverse esecuzioni. Gli esperimenti sono stati condotti su un dataset condiviso di commit reali, con valutazione basata su metriche automatiche e osservazioni qualitative.

I risultati confermano che gli approcci più recenti, basati su tecniche di retrieval e apprendimento robusto, superano i modelli sequenziali tradizionali. Tuttavia, emerge anche la fragilità di alcuni esperimenti originali e il limite delle metriche automatiche nel riflettere la qualità semantica dei messaggi generati. Il lavoro evidenzia quindi la necessità di benchmark riproducibili, valutazioni più affidabili e modelli più attenti al contenuto informativo. Le analisi svolte forniscono una base solida per futuri sviluppi nel campo della documentazione automatica del software.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto e tema del lavoro

Nel moderno sviluppo software, i *commit message* costituiscono un elemento fondamentale del processo di versionamento e collaborazione. Ogni modifica apportata al codice in sistemi di controllo versione distribuiti (come Git) è accompagnata da un messaggio descrittivo che riassume *cosa* è stato cambiato e *perché* il cambiamento è stato effettuato [3]. Questi messaggi rappresentano una forma di documentazione naturale strettamente legata all'evoluzione del codice sorgente e vengono utilizzati per facilitare attività di manutenzione, revisione e comprensione del software [4].

Numerosi studi hanno dimostrato il valore informativo dei commit message in vari compiti di analisi del software. Essi sono stati usati, ad esempio, per classificare la tipologia di manutenzione [5], migliorare la predizione di difetti [6], e identificare automaticamente operazioni di refactoring documentate dagli sviluppatori [7]. L'uso congiunto di messaggi testuali e informazioni strutturali del codice consente inoltre di migliorare la tracciabilità dei cambiamenti [8].

Con l'avanzare delle tecniche di Natural Language Processing (NLP) e Deep Learning, la generazione automatica dei commit message è diventata un tema centrale.

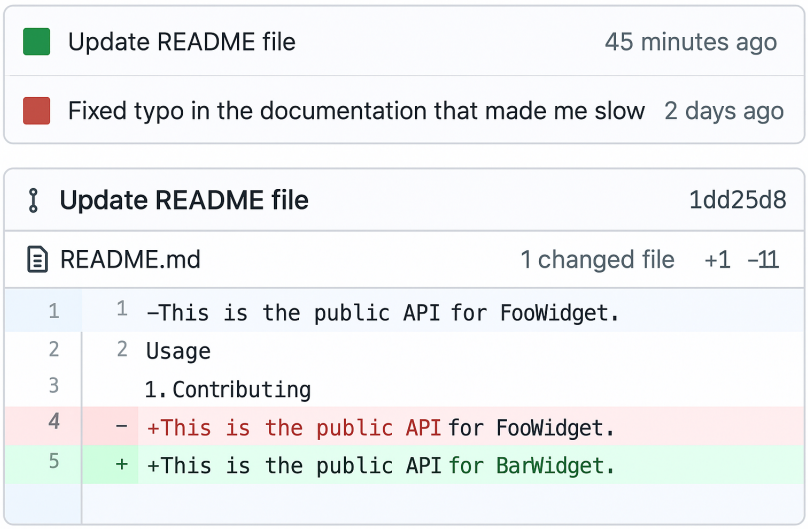

Figura 1.1: Esempio di differenza (diff) di codice e confronto tra commit message più o meno informativi. Un messaggio ben scritto comunica chiaramente sia *cosa* è stato fatto sia *perché*, facilitando la comprensione del contesto.

Oltre alle linee guida introdotte dalle comunità di sviluppo, diversi lavori scientifici hanno trattato la generazione automatica dei commit message come un problema di traduzione automatica dal codice al linguaggio naturale [9, 1, 2]. Questa prospettiva ha dato origine a modelli sempre più sofisticati che combinano rappresentazioni sintattiche e semantiche del codice per produrre descrizioni accurate e utili delle modifiche. Il campo della Commit Message Generation (CMG) è oggi un'area di ricerca attiva, al crocevia tra software maintenance, program comprehension e natural language generation.) è quindi oggi un'area di ricerca attiva, al crocevia tra software maintenance, program comprehension e natural language generation.

1.2 Motivazioni e problema

Nonostante la loro importanza, nella pratica quotidiana la qualità dei messaggi di commit risulta spesso inadeguata o quantomeno molto variabile. Molti messaggi di commit sono incompleti, vaghi o privi del giusto livello di dettaglio, il che ne riduce l'utilità come documentazione dei cambiamenti. Studi empirici hanno quantificato il fenomeno: una larga frazione dei commit message (circa il 44%) manca di informazioni essenziali, come la motivazione o perfino la descrizione accurata di *cosa* è stato

modificato [7].
Un'analisi su decine di migliaia di progetti open source ha inoltre rilevato che il 14% dei commit è privo di un messaggio significativo, mentre circa due terzi consistono in poche parole generiche [9]. Un altro studio ha osservato che la qualità complessiva dei commit message tende addirittura a peggiorare nel tempo, mentre la percezione degli sviluppatori è opposta [3]. In altre parole, commit message scritti frettolosamente o superficialmente possono generare debito di documentazione e ostacolare le future attività manutentive.

Tra i problemi più comuni si riscontra la mancanza della motivazione (*perché* si è effettuata la modifica) a corredo della descrizione tecnica di *cosa* è stato modificato. Messaggi generici come "Fixed bug" o "Updated file" forniscono scarso supporto nelle attività di debug, code review o analisi storica [4]. In altri casi, il contenuto del messaggio può risultare incoerente rispetto al diff associato, descrivendo solo parzialmente le modifiche effettuate o tralasciando cambiamenti significativi.
Queste carenze rendono più difficile tracciare l'evoluzione del software, complicano l'onboarding di nuovi contributori e possono aumentare i costi cognitivi nelle attività di manutenzione correttiva ed evolutiva.

Alla luce di queste problematiche, sono emerse due principali direzioni per affrontare il problema dei commit message di bassa qualità:

- definire standard e linee guida per favorire messaggi più informativi, sensibilizzando gli sviluppatori sull'inclusione di *cosa* e *perché* in ogni commit;
- sviluppare metodi automatici basati su NLP e deep learning per generare commit message coerenti e contestualizzati direttamente dal diff del codice [9, 8].

Nonostante i progressi, restano aperte importanti domande: quanto tali modelli siano in grado di catturare sia il *cosa* che il *perché* di una modifica, quanto siano affidabili nei contesti reali, e quanto siano effettivamente riproducibili i risultati riportati nei lavori originali. Queste lacune motivano la necessità di una valutazione empirica rigorosa dei moderni modelli di Commit Message Generation (CMG).

1.3 Obiettivi della ricerca

Il presente lavoro di tesi si colloca nel filone dell'automazione e mira a fornire una valutazione empirica rigorosa dello stato dell'arte nella generazione automatica dei messaggi di commit. In particolare, l'obiettivo è confrontare in modo sistematico le prestazioni di tre modelli avanzati proposti recentemente in letteratura per la Commit Message Generation (CMG): **CoDiSum** [1], **RACE** [2] e **KADEL** [10].
Questi modelli rappresentano approcci differenti e complementari allo stesso problema: si passa da un modello sequenziale puro (Neural Machine Translation applicata a diff grezzi), all'integrazione di meccanismi di *retrieval* semantico, fino a strategie di *denoising* basate sulla conoscenza delle buone pratiche di scrittura.

Attraverso un approccio sperimentale e comparativo, la tesi intende rispondere a domande di ricerca relative alla qualità, all'efficienza, alla robustezza e alla riproducibilità di tali tecniche. In sintesi, gli obiettivi specifici del lavoro sono:

- **Valutare la qualità e la fedeltà semantica** dei messaggi generati dai diversi modelli, verificando in che misura essi riescano a produrre commit message informativi, coerenti e aderenti alle modifiche del codice (includendo sia il *cosa* sia il *perché* del cambiamento).

- **Confrontare l'efficienza computazionale** di CoDiSum, RACE e KADEL, analizzando il rapporto tra qualità ottenuta e risorse impiegate (tempi di addestramento, dimensione dei batch, consumo di memoria GPU), così da identificare eventuali trade-off tra costo computazionale e performance.

- **Verificare la stabilità, riproducibilità e replicabilità** dei risultati, studiando la variazione delle prestazioni al variare di fattori non deterministici (ad esempio random seed e inizializzazioni), delle diverse strategie di suddivisione del dataset (ad es. per progetti o temporale) e valutando se i risultati riportati nei lavori originali possano essere confermati in un contesto indipendente.

- **Indagare l'impatto del contesto dei dati** sulle prestazioni dei modelli, confrontando risultati ottenuti con un dataset "classico" di commit Java (simile a quello utilizzato per l'addestramento di CoDiSum) rispetto a un dataset più recente

e ricco di informazioni contestuali, come il dataset MCMD introdotto da Tao et al. [11] o il dataset CodeCM, così da comprendere se dati più informativi possano realmente migliorare la generazione dei messaggi.

Per raggiungere tali obiettivi, è stato progettato un esperimento controllato in cui ciascun modello viene addestrato e valutato sugli stessi insiemi di dati, adottando configurazioni il più possibile allineate alle rispettive proposizioni originali.
Le prestazioni sono misurate mediante un insieme comune di metriche di valutazione (BLEU, ROUGE-L, METEOR, ecc.), integrate da metriche semantiche più avanzate come *BERTScore* e *CodeBLEU*. Le differenze osservate vengono poi analizzate tramite un protocollo statistico rigoroso, includendo test non parametrici, analisi delle varianze ed *effect size*.
Questo approccio consente di fornire un quadro chiaro e basato su evidenze empiriche dello stato attuale della CMG, delineando punti di forza e limiti degli approcci contemporanei, insieme a possibili direzioni future per far progredire ulteriormente il campo.

1.4 Struttura della tesi

La restante parte della tesi è organizzata come segue.

Il **Capitolo 2** presenta lo stato dell'arte, delineando il ruolo dei messaggi di commit nell'ingegneria del software, le principali problematiche emerse riguardo alla loro qualità e le soluzioni proposte in letteratura, con particolare attenzione agli approcci basati su metodi di intelligenza artificiale per la generazione automatica.

Il **Capitolo 3** descrive il design sperimentale e la metodologia adottata nello studio: vengono formalizzate le domande di ricerca e le relative ipotesi, introdotti nel dettaglio i modelli analizzati, specificati i dataset e le configurazioni sperimentali, e infine presentati i risultati empirici ottenuti nelle diverse fasi, con una prima analisi critica delle prestazioni.

Il **Capitolo 4** sviluppa la risposta alle domande di ricerca, organizzando l'analisi attorno ai tre temi centrali della tesi: riproducibilità dei modelli storici, confronto tra modelli moderni su dataset condiviso e stabilità delle prestazioni. Le evidenze quantitative e qualitative sono discusse in modo integrato, al fine di valutare il comportamento dei modelli in relazione alle ipotesi definite nel Capitolo 3.

Il **Capitolo 5** presenta le conclusioni complessive del lavoro, discutendo i contributi emersi, i limiti metodologici e sperimentali, e le possibili direzioni per ricerche future nel campo della generazione automatica di documentazione per il codice, con particolare riferimento ai commit message generati automaticamente.

CAPITOLO 2

Stato dell'arte

2.1 I messaggi di commit nel software engineering

I *commit message* svolgono un ruolo centrale nell'ingegneria del software moderna: rappresentano il collegamento diretto tra le modifiche effettuate al codice e le intenzioni progettuali degli sviluppatori. Ogni volta che uno sviluppatore registra una modifica in un sistema di versionamento (come `Git`), il messaggio di commit fornisce un breve resoconto di *cosa* è stato cambiato e, idealmente, *perché* il cambiamento è stato introdotto. Questa informazione contestuale costituisce una forma di documentazione naturale, strettamente intrecciata con l'evoluzione del codice sorgente, e viene utilizzata in numerose attività di manutenzione, revisione e comprensione del software [3, 4].

Diversi studi hanno evidenziato come i commit message siano una fonte informativa preziosa per molteplici task di analisi del software. Essi vengono impiegati, ad esempio, per classificare le tipologie di manutenzione e studiare i pattern evolutivi dei progetti [5], supportare analisi di qualità e predizione di difetti [6], nonché per identificare refactoring documentati esplicitamente dagli sviluppatori [7]. In generale, la combinazione di informazione testuale (il messaggio) e strutturale (il *diff* del

codice) consente di migliorare la tracciabilità e l'interpretabilità dei cambiamenti, fornendo un contesto più ricco rispetto all'analisi del solo codice sorgente [4, 8].

Nonostante questa centralità, la qualità dei commit message osservata nei progetti reali è estremamente variabile. Studi su larga scala mostrano che molti repository open source contengono messaggi generici, poco informativi o addirittura assenti [9]. Xu et al. [1], analizzando un ampio campione di progetti Java, riportano che circa il 14% dei commit non presenta un messaggio significativo, evidenziando una distanza rilevante tra le linee guida raccomandate e la pratica quotidiana degli sviluppatori.

Parallelamente, con la crescente diffusione di tecniche di *Natural Language Processing* (NLP) e di *Deep Learning*, i commit message sono stati reinterpretati come descrizioni in linguaggio naturale delle modifiche al codice, in analogia con il problema della *code summarization*. In questa prospettiva, diversi lavori hanno iniziato a trattare la generazione del messaggio di commit come un problema di traduzione automatica dal linguaggio del codice (o, più specificamente, dal *diff*) al linguaggio naturale [8, 1, 2]. L'uso di architetture neurali ha reso possibile combinare rappresentazioni sintattiche (ad esempio basate su *Abstract Syntax Tree*) e semantiche (embedding contestuali) del codice per produrre messaggi più coerenti e informativi.

In sintesi, i commit message si configurano come un elemento chiave del processo di sviluppo collaborativo: collegano le modifiche al codice con le decisioni e le motivazioni degli sviluppatori e costituiscono una risorsa fondamentale per la manutenzione e la comprensione dei sistemi software. La loro analisi, il miglioramento della qualità e la generazione automatica rappresentano oggi una linea di ricerca consolidata, al crocevia tra *software maintenance*, *program comprehension* e *natural language generation*.

2.2 Problemi legati alla qualità dei messaggi di commit

La qualità dei messaggi di commit rappresenta un aspetto cruciale nell'ingegneria del software collaborativo, poiché condiziona la comprensibilità, la tracciabilità e

la manutenzione del codice nel tempo. Nonostante il loro ruolo strategico — come ponte tra la modifica al codice e la documentazione del cambiamento — numerosi studi empirici mostrano che molti commit message risultano incompleti, generici o privi della motivazione sottostante (il "why") della modifica.
Studi recenti hanno quantificato tale problema in modo preciso. *What Makes a Good Commit Message?* ha analizzato circa 1.600 commit provenienti da cinque progetti open source e ha rilevato che circa il 44% dei messaggi potrebbe essere migliorato, mancando di informazioni essenziali come la motivazione o il contesto del cambiamento [7]. Un'analisi su larga scala condotta da Tian et al. [9] ha inoltre evidenziato che circa il 14% dei commit in progetti Java open source non contiene alcun messaggio significativo, mentre una quota consistente dei messaggi è composta da poche parole generiche (es. "update", "fix", "changes").
Uno studio longitudinale di Mukherjee e Robbes [3] ha mostrato inoltre che la qualità media dei commit message tende a peggiorare nel tempo, contrariamente alla percezione degli sviluppatori, e che messaggi poco informativi possono correlare con una maggiore incidenza di difetti post-rilascio e fallimenti di build. Questi risultati indicano l'esistenza di un vero e proprio *debito di documentazione*, che si accumula progressivamente e rende più costosa la manutenzione futura.

Categorie principali di problemi

Dalla letteratura emergono tre macro-categorie ricorrenti di problemi nei commit message:

- **Mancanza di completezza ("what" + "why")**. Molti commit descrivono solo *cosa* è stato cambiato (e.g., "refactor code", "fix bug") senza esplicitare *perché* la modifica è stata necessaria. L'assenza del "why" ostacola la comprensione dell'intento progettuale [5] e penalizza attività di manutenzione e debugging.

- **Generalità o vaghezza**. Messaggi eccessivamente generici o placeholder riducono drasticamente il valore documentale del commit. Eyolfson et al. [6] mostrano che commit poco descrittivi rendono difficoltoso correlare modifiche e difetti, ostacolando analisi successive.

- **Incoerenza tra messaggio e *diff***. Talvolta il messaggio non rappresenta fedelmente lo scope del commit: vengono omessi cambiamenti importanti, o la descrizione risulta non congruente rispetto alle linee modificate. Ciò può essere particolarmente problematico nei processi di code review [4].

Le conseguenze di tali problemi vanno ben oltre la leggibilità. Una documentazione di commit carente:

- aumenta il costo cognitivo nelle attività di revisione;
- riduce la tracciabilità delle modifiche nel tempo;
- complica l'onboarding di nuovi sviluppatori;
- può influenzare negativamente la qualità del software nelle fasi successive.

Come osservato da esperti del settore[1], *"un buon commit message può essere tanto prezioso quanto il codice stesso"*, poiché fornisce un contesto interpretativo indispensabile per comprendere il cambiamento.

Impatto sulla ricerca e sull'addestramento dei modelli

Dal punto di vista della ricerca, la presenza di commit message generici, incoerenti o del tutto mancanti introduce un ulteriore problema: aumenta il *rumore* nei dataset utilizzati per addestrare e valutare i modelli di Commit Message Generation (CMG). Dataset rumorosi compromettono la qualità del segnale informativo disponibile, rendendo più difficile:

- valutare correttamente le capacità dei modelli;
- comprendere se una bassa performance sia dovuta al modello o all'incoerenza dei dati;
- utilizzare metriche automatiche basate su n-gram in modo affidabile.

[1] Chris Beams, *How to Write a Git Commit Message*, disponibile su: `https://chris.beams.io/posts/git-commit/`.

Tali limiti hanno motivato la nascita di due principali direzioni di ricerca per affrontare il problema della bassa qualità dei messaggi:

1. **Standard e linee guida** per promuovere messaggi più completi e informativi (e.g. formati come *Conventional Commits*).

2. **Tecniche automatiche** basate su NLP e deep learning per generare commit message coerenti e contestualizzati direttamente dal *diff* o dalle *edit actions*. I modelli più recenti (es. RACE [2] e KADEL [10]) incorporano meccanismi di retrieval, filtering o denoising progettati esplicitamente per mitigare il rumore dei dataset.

Questi aspetti introducono in modo naturale la successiva sezione, dedicata alla panoramica dei principali modelli e approcci automatici per la generazione dei commit message.

2.3 Approcci automatici alla generazione dei messaggi di commit

La crescente consapevolezza dei limiti qualitativi dei commit message prodotti manualmente (incompletezza, vaghezza, assenza del "why", incoerenze con il diff) ha portato, negli ultimi dieci anni, allo sviluppo di una vasta gamma di approcci automatici volti a supportare o sostituire la scrittura manuale. Tali approcci si sono evoluti progressivamente, passando da semplici metodi basati su regole o *information retrieval* fino a modelli neurali complessi e sistemi *knowledge-aware*.
Questa sezione fornisce una panoramica strutturata dell'evoluzione dei principali filoni di ricerca, ponendo particolare attenzione agli approcci che hanno portato allo stato dell'arte attuale.

Approcci iniziali: rule-based e information retrieval

I primi tentativi di generare automaticamente commit message risalgono a lavori basati su regole e template linguistici. Approcci come quelli di Buse e Weimer [4] cercavano di estrarre informazioni chiave dal diff e sintetizzarle tramite formule

linguistiche predefinite. Sebbene trasparenti e controllabili, tali metodi risultavano poco flessibili e incapaci di generalizzare a modifiche complesse.
Parallelamente, alcuni sistemi hanno adottato tecniche di *information retrieval* (IR), generando il messaggio del commit più simile recuperando descrizioni da commit analoghi nel repository. Modelli come NNGen o Lucene-based retrieval costituivano una baseline importante, anche se limitata dal fatto che non producevano testo nuovo, ma si limitavano a riutilizzare messaggi preesistenti [12].
Questi approcci hanno fornito la base concettuale per l'uso del diff come rappresentazione testuale del cambiamento, ma non erano in grado di modellare fenomeni linguistici complessi né di inferire la motivazione del cambiamento.

Approcci di traduzione automatica (NMT)

Con l'introduzione dei modelli di *Neural Machine Translation* (NMT), la generazione dei messaggi di commit è stata reinterpretata come un problema di traduzione dal linguaggio del codice a quello naturale. L'idea, proposta inizialmente da Liu et al. [8], consiste nel trattare il diff come la sequenza sorgente e il commit message come la sequenza target, addestrando un modello encoder–decoder con attenzione.
Tali sistemi hanno rappresentato un avanzamento significativo rispetto ai metodi IR, producendo testi nuovi e flessibili. Tuttavia, mostravano ancora due limiti fondamentali:

- difficoltà nel trattare il problema delle parole fuori vocabolario (*OOV*), comuni nei nomi di variabili e metodi;
- incapacità di sfruttare la struttura sintattica del codice (e.g. AST).

Questi limiti sono stati affrontati da **CoDiSum** [1], che ha introdotto una rappresentazione congiunta della struttura del codice (via AST semplificati) e della sua semantica testuale, insieme a un meccanismo di copia ispirato alle pointer-generator networks.
CoDiSum ha rappresentato una prima svolta nello stato dell'arte, dimostrando che integrare struttura e testo è cruciale per generare messaggi più informativi.

Approcci ibridi e metodi retrieval+generazione

Con il crescente successo dei modelli pre-addestrati sul codice, come CodeBERT e CodeT5, la ricerca si è spostata verso approcci ibridi che combinano NMT e IR.
In questa categoria rientrano modelli come:

- **ATOM** (Liu et al., 2020) [13], che introduce una fase di retrieval per selezionare commit simili da utilizzare come guida;
- **CoRec** (Wang et al., 2021) [14], che unisce retrieval semantico e generazione neurale tramite meccanismi di fusione multivista.

Tali modelli hanno dimostrato che l'informazione proveniente da commit simili migliora notevolmente la qualità del messaggio generato, soprattutto nella scelta della terminologia tecnica.

RACE: la maturità degli approcci retrieval-augmented

Il passaggio successivo è stato segnato da **RACE** (Shi et al., 2022) [2], che propone un paradigma pienamente *retrieval-augmented*. RACE utilizza:

1. un encoder Transformer (basato su CodeT5) per mappare i diff in uno spazio semantico;
2. un modulo di retrieval per selezionare il commit più simile;
3. un *exemplar guider* che combina in modo pesato informazione dal diff corrente e dal diff recuperato;
4. un decoder Transformer che genera il messaggio finale.

Rispetto a CoDiSum e ai modelli NMT, RACE ottiene miglioramenti significativi su BLEU, METEOR, ROUGE e CIDEr, mostrando che la conoscenza esplicita del contesto storico (commit simili) è cruciale per la qualità del testo generato.

KADEL: apprendimento consapevole della qualità dei dati

Nonostante i progressi introdotti da RACE, rimanevano aperti problemi relativi alla qualità del dataset: i commit message reali includono rumore, vaghezza e incoerenze.

Questo limita la capacità dei modelli di apprendere buone pratiche linguistiche. **KADEL** (Tao et al., 2024) [10] affronta esplicitamente questo aspetto introducendo un paradigma di *Knowledge-Aware Denoising Learning*. Il modello:

- identifica commit "di buona pratica" e ne estrae una rappresentazione semantica di qualità (*commit knowledge*);
- applica un meccanismo di denoising dinamico che riduce il peso dei commit rumorosi durante il training;
- fonde conoscenza estratta e diff corrente tramite un meccanismo residuale.

KADEL rappresenta l'evoluzione più consapevole della qualità dei dati, ottenendo prestazioni state-of-the-art su MCMD.

Ruolo emergente dei Large Language Models

Parallelamente allo sviluppo dei modelli specializzati, la crescita dei *Large Language Models* (LLM) ha aperto possibilità nuove. Modelli general-purpose come GPT, StarCoder o CodeLlama sono in grado di generare commit message con coerenza e fluidità elevate, spesso includendo spontaneamente il "why". Tuttavia:

- la loro valutazione tramite metriche n-gram non è sempre affidabile;
- la riproducibilità dei risultati dipende fortemente dal prompt e dal contesto;
- la mancanza di controllo fine rende difficile utilizzarli come baseline nei confronti sperimentali classici.

In sintesi, l'evoluzione degli approcci automatici alla generazione dei commit message mostra una transizione chiara: dai metodi basati su regole, ai modelli NMT, agli approcci ibridi retrieval+generazione, fino ai moderni sistemi knowledge-aware e ai LLM. Questa traiettoria riflette la crescente complessità nel catturare non solo il *cosa* ma anche il *perché* della modifica, un obiettivo ancora aperto nella ricerca attuale.

2.4 Modelli di deep learning

L'evoluzione dei modelli di Commit Message Generation (CMG) negli ultimi anni riflette una transizione chiara: dai primi approcci di traduzione neurale con rappresentazioni limitate, fino a modelli retrieval-augmented e metodi *knowledge-aware* che integrano informazioni strutturali, semantiche e qualitative dei dati. In questa sezione vengono analizzati nel dettaglio i tre modelli più rappresentativi di questa evoluzione: **CoDiSum** (2019), **RACE** (2022) e **KADEL** (2024).
Ognuno di essi affronta limiti specifici emersi nei modelli precedenti, proponendo soluzioni via via più sofisticate. La loro analisi comparata costituisce il fondamento sperimentale di questo lavoro di tesi.

2.4.1 CoDiSum

CoDiSum (Xu et al., 2019) [1] rappresenta uno dei primi modelli neurali ad aver affrontato in modo sistematico due limiti fondamentali dei precedenti approcci NMT:

1. la difficoltà nel gestire il problema delle parole fuori vocabolario (*OOV*), molto frequenti nei nomi di variabili, classi e metodi;

2. l'incapacità dei modelli seq2seq tradizionali di catturare la struttura sintattica del codice oltre alla sua superficie testuale.

Idea principale

CoDiSum introduce una rappresentazione *congiunta* del diff del codice composta da:

- una vista strutturale (*code structure*), basata su identificatori sostituiti da placeholder per modellare la forma sintattica;

- una vista semantica (*code semantics*), basata sulla segmentazione degli identificatori in sottoparole.

Queste due viste vengono fuse token-per-token e forniscono una codifica più ricca e robusta del diff.

Architettura

L'architettura comprende:

- due encoder GRU bidirezionali, uno per la sequenza strutturale e uno per quella semantica;
- un meccanismo di attenzione che opera sulla rappresentazione fusa;
- un decoder GRU dotato di **copying mechanism**, mutuato dalle pointer-generator networks [15], che consente di copiare token dal diff.

L'uso del *copy mechanism* permette al modello di generare correttamente identificatori tecnici (OOV), migliorando in modo significativo il richiamo degli identificatori rispetto ai modelli NMT tradizionali [8].

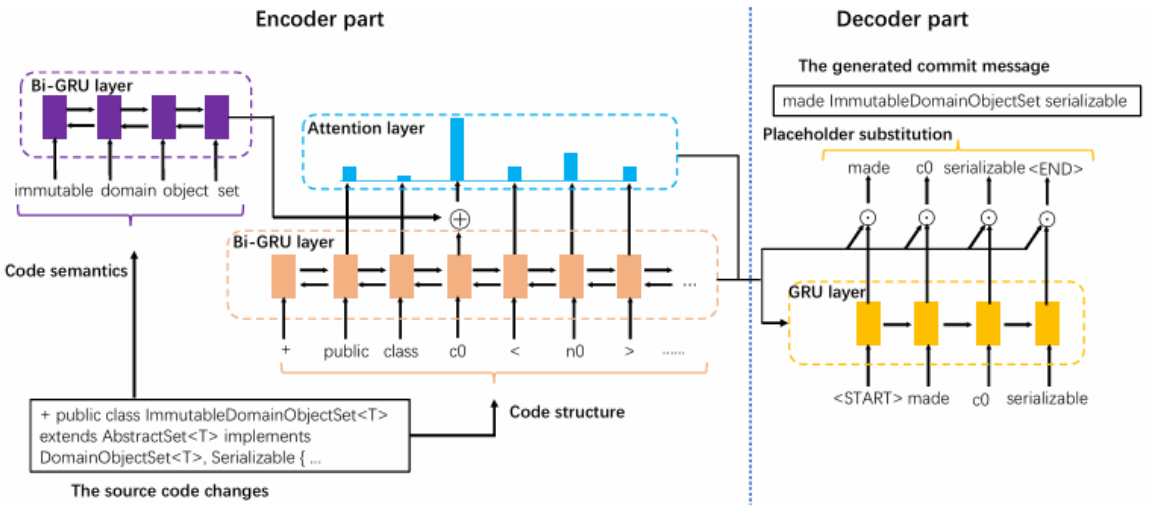

Figura 2.1: Architettura del modello CoDiSum [1], composta da due encoder Bi-GRU (strutturale e semantico), meccanismo di attenzione e decoder con copy mechanism.

Limiti principali

Nonostante il significativo progresso, CoDiSum presenta alcuni limiti:

- utilizza solo informazioni locali del diff e non sfrutta alcun contesto storico dei commit;
- non integra conoscenza esterna o esempi simili, producendo spesso messaggi troppo brevi;

- la sua implementazione originale (TensorFlow 1.x) è oggi considerata un artefatto *legacy*, di difficile riproducibilità.

2.4.2 RACE

RACE (Shi et al., 2022) [2] rappresenta un'evoluzione diretta rispetto ai modelli sequenziali come CoDiSum. Il modello parte da un'osservazione empirica: *molti commit presentano pattern ricorrenti nelle modifiche e nei messaggi*, e questi pattern possono essere sfruttati recuperando un commit simile come guida.

Idea principale: retrieval-augmented generation

RACE integra due fasi:

1. **Retrieval**: ricerca del commit semanticamente più simile rispetto a quello in input;
2. **Generation**: generazione del nuovo messaggio guidata dai contenuti del commit recuperato.

Questa pipeline permette di superare limiti fondamentali dei modelli NMT:

- ridondanza e ripetitività dei messaggi generati;
- mancanza di contesto storico del repository;
- difficoltà nel catturare la motivazione del cambiamento.

Architettura

Il framework è composto da:

- un encoder Transformer (CodeT5 [16]) per mappare i diff in uno spazio semantico denso;
- un **retriever** che seleziona il commit più simile tramite similarità coseno;
- un **exemplar guider**, che combina il diff del commit corrente con quello recuperato tramite una fusione pesata;

- un decoder Transformer per la generazione del messaggio.

Questa architettura consente a RACE di produrre messaggi più informativi e con più dettagli tecnici rispetto ai modelli puramente generativi.

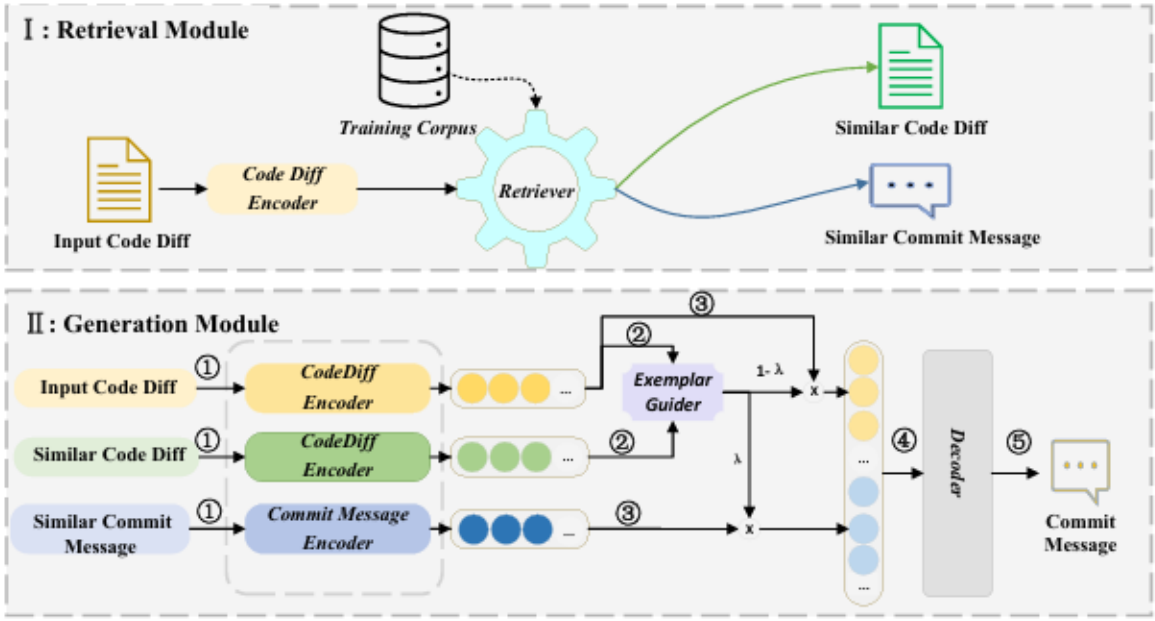

Figura 2.2: Architettura del modello RACE [2]. La pipeline si compone di due moduli: (I) *Retrieval Module*, che utilizza un encoder CodeT5 per recuperare dal corpus un commit semanticamente simile all'input; (II) *Generation Module*, che combina le rappresentazioni del diff originale, del diff recuperato e del relativo messaggio tramite l'*Exemplar Guider*, producendo infine il commit message tramite un decoder Transformer.

Limiti principali

Pur migliorando significativamente le performance, RACE presenta alcuni limiti:

- richiede un corpus molto grande per il retrieval, con costi computazionali elevati;
- comporta un tempo di addestramento notevolmente superiore a CoDiSum (fino a 35 ore);
- eredita il rumore contenuto nel dataset MCMD, da cui dipende la qualità del retrieval.

2.4.3 KADEL

KADEL (Tao et al., 2024) [10] nasce per affrontare un problema rimasto aperto anche dopo RACE: *il rumore semantico nei dataset*. Molti messaggi reali, infatti, sono incompleti o vaghi (es. “update”, “fix bug”), e questo compromette l’apprendimento dei modelli.

Idea principale: knowledge-aware denoising

KADEL introduce un paradigma innovativo:

- identifica commit di “buona pratica” e ne apprende una rappresentazione semantica (*commit knowledge*);
- applica un **dynamic denoising module** che riduce il peso dei sample rumorosi durante il training;
- fonde diff e conoscenza tramite un meccanismo residuale che guida il modello verso messaggi meglio strutturati.

Architettura

KADEL sfrutta:

- un encoder–decoder Transformer basato su CodeT5;
- un modulo di estrazione della conoscenza da commit “puliti”;
- un sistema di pesatura dinamica dei campioni: commit con perdita elevata contribuiscono meno all’addestramento;
- una loss pesata che enfatizza esempi ad alta qualità.

Limiti principali

Pur essendo lo stato dell’arte attuale, KADEL presenta alcuni limiti:

- dipendenza dalla disponibilità di commit “di buona pratica” per il knowledge extraction;

- elevato costo computazionale dovuto alla doppia fase di addestramento;
- difficoltà nel gestire diff molto lunghi (limitazione ereditata da CodeT5).

Sintesi comparativa

L'evoluzione dai modelli sequenziali ai metodi retrieval-augmented e knowledge-aware mostra una progressione coerente nel tentativo di affrontare limiti concreti:

- CoDiSum affronta OOV e struttura del codice;
- RACE affronta mancanza di contesto e ripetitività;
- KADEL affronta rumore e scarsa qualità dei dataset.

Queste tre soluzioni costituiscono quindi un percorso evolutivo logico e rappresentano tre tappe fondamentali della ricerca moderna sulla Commit Message Generation.

2.5 Dataset comunemente utilizzati

La qualità e la varietà dei dataset rappresentano un fattore determinante per lo sviluppo di modelli di generazione automatica dei messaggi di commit. A differenza di molti altri task NLP, i commit message reali presentano un'elevata variabilità stilistica, una frequente mancanza del "why", e un forte rumore semantico dovuto a messaggi generici o non informativi. Per questo motivo, i dataset utilizzati in letteratura differiscono significativamente in termini di dimensione, linguaggi di programmazione, qualità dei messaggi e pre-processing applicato.

In questa sezione vengono descritti i dataset più rilevanti adottati nella CMG, con particolare attenzione alle risorse utilizzate dai modelli analizzati in questa tesi (CoDiSum, RACE e KADEL).

GitHub Java Dataset (CoDiSum)

Il primo dataset di ampia scala per la CMG è quello introdotto da Xu et al. [1]. Esso è composto da circa **90 000** coppie *(diff, message)* estratte dai **1 000 repository Java più popolari** su GitHub.

Pre-processing. Il dataset è stato ottenuto tramite una pipeline di pulizia che:

- rimuove file non Java e commit non testuali;
- elimina messaggi troppo brevi (meno di tre parole);
- filtra commit duplicati, rollback e merge commit;
- normalizza i diff includendo solo modifiche al codice sorgente.

Caratteristiche. Il dataset è relativamente piccolo e monolingua, ma presenta un vantaggio: molti messaggi sono manuali e di qualità medio-alta, poiché provengono da progetti popolari. Questo spiega il buon comportamento di CoDiSum nelle metriche basate su n-gram.

Limiti. Le principali limitazioni sono:

- dimensione ridotta (90k), insufficiente per modelli Transformer di grandi dimensioni;
- mancanza di diversità linguistica (solo Java);
- assenza di informazioni contestuali aggiuntive (nome file, storia commit).

MCMD Dataset (RACE e KADEL)

Il dataset più utilizzato nei lavori recenti è **MCMD (Multi-lingual Commit Message Dataset)** introdotto da Tao et al. [11]. Contiene oltre **900 000** coppie *(diff, message)* provenienti da cinque linguaggi:

- Java,
- C#,

- C++,
- Python,
- JavaScript.

Rappresentazione dei diff. Una novità importante di MCMD è l'uso delle **edit actions** secondo Panthaplackel et al. [17], che rappresentano ogni modifica tramite token speciali:

`<insert>`, `<delete>`, `<replace>`

Questo schema permette di catturare operazioni semantiche non evidenti nei diff raw line-based, facilitando l'apprendimento da parte dei modelli Transformer.

Pre-processing. Il dataset applica filtri rigorosi per rimuovere:

- messaggi duplicati,
- commit di merge,
- commit non testuali o logistici (e.g., build scripts),
- commit generati automaticamente da tool.

Punti di forza. MCMD è oggi considerato lo **standard de facto** per la CMG perché:

- contiene quasi un milione di esempi;
- copre più linguaggi di programmazione;
- include diff strutturati tramite edit actions.

Limiti. Nonostante le sue dimensioni, MCMD presenta un notevole **rumore semantico**: molti messaggi sono generici (es. "update", "fix bug"), incompleti o inconsistenti con il diff. Questo è uno dei motivi per cui KADEL introduce tecniche di denoising consapevoli della qualità dei dati.

RACE Corpus

Per migliorare il training dei modelli retrieval-based, Shi et al. [2] hanno proposto un'estensione di MCMD nota come **RACE Corpus**.
Essa adotta una struttura *tripla*:

$$(\text{diff},\ \text{similar diff},\ \text{similar message})$$

Questa forma è necessaria per addestrare modelli *retrieval-augmented*, in cui il messaggio recuperato funge da esempio guida.

Caratteristiche. Il corpus include:

- commit reali,
- commit semanticamente simili (retrieval offline),
- il loro messaggio originale.

Limiti. La costruzione del corpus richiede:

- un encoder pre-addestrato su tutti i diff,
- calcolo di similarità su milioni di embedding,
- elevati costi computazionali.

Nonostante ciò, il corpus è fondamentale per addestrare RACE in modo efficace.

Rationale Dataset (Linux Kernel OOM-Killer)

Zhou et al. [18] hanno introdotto un dataset focalizzato su commit ad alta complessità semantica provenienti dal modulo *Out-of-Memory Killer* del kernel Linux. Il dataset contiene **2 234 frasi annotate manualmente** secondo tre categorie:

- *Decision*,
- *Rationale*,
- *Supporting Facts*.

Questo dataset non è pensato per l'addestramento, ma per valutare la capacità dei modelli di generare testi che includano la componente motivazionale del cambiamento.

Altri dataset secondari

In letteratura compaiono inoltre:

- **CommitGen corpus** [12]: 75k commit Java, utilizzato come baseline NMT.
- **Security Commit Dataset**: commit relativi a vulnerabilità e patch di sicurezza.
- **Good-Practice Subset** [10]: circa 100k commit filtrati secondo convenzioni stilistiche (`type(scope): subject`), usati per il knowledge extraction di KADEL.

Sintesi comparativa

La Tabella 2.1 riassume le caratteristiche principali dei dataset per la CMG.

Tabella 2.1: Principali dataset utilizzati nei modelli di commit message generation.

Dataset	Dimensione	Linguaggi	Modelli utilizzati
GitHub Java (CoDiSum) [1]	90k	Java	CoDiSum
MCMD [11]	900k	Java, C#, C++, Python, JS	RACE, KADEL
RACE Corpus [2]	450k (triplette)	Multilingua	RACE
Rationale Dataset [18]	2.2k frasi	C (Linux)	valutazione qualitativa
Good-Practice Subset [10]	100k	Java, Python	KADEL

Osservazioni finali

L'evoluzione dei dataset evidenzia un trend chiaro:

- crescita delle dimensioni (da 90k a quasi 1M di commit),
- maggiore diversità linguistica,
- introduzione di rappresentazioni più ricche (edit actions),
- crescente attenzione alla qualità semantica dei messaggi.

Questi fattori hanno influenzato profondamente lo sviluppo delle architetture moderne, rendendo possibile l'addestramento di modelli retrieval-augmented (RACE) e knowledge-aware (KADEL), e rivelando come la qualità del dato sia tanto importante quanto le scelte architetturali.

2.6 Metriche di valutazione

La valutazione della qualità dei messaggi di commit generati automaticamente è una componente essenziale per confrontare modelli differenti e per stimare il reale avanzamento dello stato dell'arte. Tuttavia, a differenza di altri compiti di generazione testuale, la Commit Message Generation (CMG) presenta peculiarità che rendono la valutazione particolarmente complessa: i commit message sono brevi, altamente specifici, soggetti a forte variabilità stilistica e spesso privi del "why".
Le metriche utilizzate in letteratura si possono suddividere in tre categorie:

1. metriche automatiche basate su confronto lessicale (n-gram);
2. metriche semantiche e qualitative;
3. valutazioni umane.

Ognuna di queste fornisce un punto di vista complementare sulla qualità del testo generato.

Metriche automatiche basate su n-gram

Le metriche lessicali rappresentano storicamente lo standard nelle prime ricerche sulla CMG. Esse confrontano il messaggio generato con quello reale tramite sovrapposizioni di n-gram e misure derivate dalla traduzione automatica.

BLEU. BLEU [19] misura la precisione degli n-gram prodotti dal modello rispetto al riferimento. È ampiamente utilizzato nei modelli CMG, tra cui CoDiSum [1], RACE [2] e KADEL [10]. Nonostante la sua popolarità, BLEU presenta limiti noti:

- penalizza fortemente parafrasi corrette ma lessicalmente diverse;
- non cattura relazioni semantiche;
- è sensibile alla lunghezza (problema particolarmente rilevante nei commit message, spesso molto brevi).

METEOR. METEOR [20] introduce allineamenti flessibili tramite stemming e sinonimia, risultando più sensibile alle variazioni linguistiche rispetto a BLEU. Nei lavori CMG è spesso utilizzato per valutare la coerenza linguistica delle generazioni.

ROUGE-L. ROUGE-L misura il richiamo degli n-gram tramite la *Longest Common Subsequence*. È utilizzato soprattutto in modelli che mirano a preservare la struttura informativa del messaggio (come RACE [2]).

CIDEr. CIDEr [21], originariamente proposto per la generazione di descrizioni d'immagine, è stato adottato nella CMG in quanto valorizza n-gram informativi tramite pesi TF-IDF. RACE riporta miglioramenti significativi su questa metrica, evidenziando una maggiore fedeltà semantica rispetto ai modelli precedenti.

CodeBLEU (componente testuale). Sebbene CodeBLEU nasca per confrontare porzioni di codice sorgente, nella CMG viene impiegata esclusivamente la parte testuale della metrica, in quanto il target è in linguaggio naturale. La variante usata nei modelli selezionati valuta:

- similarità lessicale (parte BLEU),

- similarità di token rilevanti (parole chiave tecniche),
- F1 su token testuali.

Questa metrica viene utilizzata come *complementare* e non come misura principale.

Identifier Recall. Metriche specifiche, come l'**identifier recall**, valutano la capacità del modello di rigenerare nomi di metodi, classi o variabili presenti nel diff. CoDiSum [1] e RACE [2] mostrano miglioramenti significativi su questa dimensione, grazie ai meccanismi di copia (CoDiSum) e al retrieval semantico (RACE). KADEL applica pattern di identificazione basati su espressioni regolari coerenti con linguaggi tipicamente tipizzati (es. Java).

Metriche semantiche e qualitative

Poiché i commit message non sono semplici riassunti lessicali del diff ma riportano anche l'intento dello sviluppatore, è necessario valutare la qualità *semantica* del testo generato.

Metriche di qualità semantica. L'introduzione del **Rationale Dataset** [18] ha avviato nuove linee di valutazione basate su categorie semantiche:

- *Decision*;
- *Rationale* (motivazione);
- *Supporting Facts*.

Queste etichette permettono di misurare in che misura un modello riesca a comunicare non solo il *what*, ma anche il *why* della modifica.

Valutazioni basate sul rumore semantico. Tao et al. [10] propongono metriche di **noise estimation** del dataset, valutando quanto un messaggio si discosti dalle buone pratiche stilistiche. KADEL sfrutta tali misure per ridurre il peso dei commit rumorosi durante il training.

Metriche embed-based (BERTScore). Metriche basate su embedding contestuali, come *BERTScore*, stanno emergendo come alternative più robuste per valutare la similarità semantica tra testo generato e riferimento. Esse misurano la sovrapposizione tra vettori embedding piuttosto che tra n-gram, risultando più adatte a valutare commit message concisi e parafrasi corrette.

Valutazioni umane

Sebbene le metriche automatiche siano necessarie, non riflettono pienamente l'utilità reale dei messaggi generati. Per questo motivo diversi studi integrano analisi manuali.

Criteri di valutazione. Gli esperti valutano tipicamente:

- **Informativeness** (completezza informativa);
- **Conciseness** (assenza di ridondanza);
- **Expressiveness** (chiarezza linguistica).

Risultati empirici. Nel lavoro di Shi et al. [2], RACE supera in modo significativo le baseline neurali e IR nelle tre dimensioni sopra citate ($p < 0.05$), confermando che il retrieval migliora la qualità percepita anche oltre le metriche n-gram.
Studi più recenti mostrano inoltre che, pur ottenendo punteggi competitivi su BLEU e ROUGE, modelli come CoDiSum risultano raramente preferiti dagli sviluppatori rispetto ai messaggi generati da Large Language Models, mettendo in luce un crescente disallineamento tra metriche automatiche e giudizi umani.

Limiti delle metriche automatiche e problemi di comparabilità

La letteratura evidenzia diversi problemi critici che rendono difficoltoso confrontare risultati tra studi diversi:

- esistono molte varianti di BLEU e ROUGE con implementazioni diverse (Moses, SacreBLEU, NLTK, B-Norm, B-Moses);

- le scelte di tokenizzazione cambiano drasticamente i punteggi (soprattutto su testi brevi);
- alcuni studi troncano i diff, altri no;
- alcuni includono la porzione di contesto (`@@`) nei diff, altri la rimuovono.

Di conseguenza, confrontare CoDiSum, RACE e KADEL sulla base dei valori riportati nei rispettivi paper è metodologicamente scorretto, poiché ciascun lavoro impiega pipeline di pre-processing e varianti metriche differenti.
Questo problema è strettamente collegato alle difficoltà di riproduzione degli artefatti, come discusso in [22, 23].
Il caso di CoDiSum è emblematico: il modello è concettualmente valido, ma la sua codebase legacy ne rende complessa la ri-esecuzione e la verifica sperimentale.

Sintesi

L'evoluzione delle metriche mostra una transizione progressiva da misure puramente lessicali verso valutazioni semantiche e umane. Le metriche n-gram restano utili per confronti rapidi e replicabili, ma non catturano pienamente la qualità e l'utilità del messaggio generato.
I moderni modelli CMG richiedono quindi:

- metriche più ricche e sensibili al contenuto semantico,
- valutazioni umane per validare la qualità percepita,
- protocolli sperimentali trasparenti e riproducibili.

Questi aspetti motivano la scelta, presentata nel Capitolo 3, di adottare un framework di valutazione ibrido basato su metriche automatiche, analisi statistiche rigorose e protocolli di riproducibilità.

2.7 Analisi statistica dei risultati sperimentali

La validazione empirica dei modelli di Commit Message Generation (CMG) richiede non solo il confronto dei valori medi delle metriche, ma anche la verifica della

significatività statistica delle differenze osservate e la stima della loro *rilevanza pratica*. Nel dominio dell'ingegneria del software, l'uso di tecniche statistiche rigorose è oggi considerato essenziale per interpretare correttamente i risultati sperimentali e per evitare conclusioni fuorvianti basate esclusivamente su confronti puntuali [22, 24]. Seguendo le raccomandazioni consolidate in ambito SE e NLP, l'analisi statistica delle prestazioni dei modelli esaminati si articola in più fasi, ciascuna mirata a verificare aspetti differenti della robustezza dei risultati.

Verifica preliminare delle assunzioni

Prima di applicare un test statistico è necessario verificare alcune assunzioni sulla distribuzione dei dati.

Normalità delle distribuzioni. La prima verifica riguarda la forma della distribuzione delle differenze tra i modelli. Viene impiegato il test di **Shapiro–Wilk**, particolarmente adatto a campioni di piccole dimensioni e ampiamente utilizzato negli esperimenti SE [22]. Se l'ipotesi di normalità ($p > 0.05$) viene soddisfatta, è possibile applicare test parametrici; in caso contrario, si preferiscono alternative non parametriche più robuste.

Omogeneità delle varianze. Per confronti tra modelli diversi è necessario verificare l'uguaglianza delle varianze tramite il test di **Levene**.
Quando le varianze risultano eterogenee, l'uso del *t-test* standard può portare a conclusioni scorrette e si rende necessario adottare la variante **Welch t-test**, più robusta in questi scenari.

Test di significatività

L'obiettivo dei test statistici è verificare se le differenze osservate tra due modelli siano attribuibili al caso o rappresentino un miglioramento effettivo.

Confronti parametrici. Se le assunzioni di normalità e omogeneità sono rispettate, si adotta il *t-test di Student*. Nel caso di confronti appaiati (ad esempio quando i modelli sono valutati sugli stessi commit), si utilizza la versione **paired t-test**.

Confronti non parametrici. Quando almeno una delle assunzioni parametriche non è soddisfatta, si ricorre a test non parametrici:

- il **Wilcoxon Signed-Rank Test** per confronti appaiati;
- il **Mann–Whitney U Test** per campioni indipendenti.

Questi test sono ampiamente raccomandati nella letteratura SE per la loro robustezza rispetto a distribuzioni non gaussiane [22, 25].

Dimensione dell'effetto e intervalli di confidenza

Limitarsi al valore p può essere fuorviante, poiché una differenza statisticamente significativa può essere poco rilevante dal punto di vista pratico. Per questo motivo, la letteratura SE suggerisce di accompagnare i test di significatività con:

Cohen's d. La dimensione dell'effetto secondo Cohen [26] quantifica l'ampiezza della differenza tra due modelli:

- $d = 0.2$: effetto piccolo,
- $d = 0.5$: effetto medio,
- $d = 0.8$: effetto grande.

Nel contesto CMG, Cohen's d offre un'indicazione più informativa della sola significatività statistica, soprattutto quando i dataset sono di grandi dimensioni.

Intervalli di confidenza (CI). Gli intervalli di confidenza al 95% consentono di stimare la stabilità della differenza osservata. Un CI che non include lo zero indica una differenza significativa, mentre la larghezza dell'intervallo fornisce un'indicazione della variabilità e dell'incertezza della stima [24].

Pratiche sperimentali e riproducibilità

La comunità SE ha evidenziato come molti lavori sperimentali manchino di una corretta applicazione di test statistici o di misure di effetto [24]. Arcuri e Briand [22] sottolineano che numerosi studi basati su algoritmi randomizzati riportano solo medie senza alcuna verifica formale. Analogamente, problemi di riproducibilità legati alle dipendenze software e alla mancanza di specifiche ambientali possono ostacolare la replicazione dei risultati [23].
Nel dominio CMG, tali problematiche sono evidenti nel caso di CoDiSum: sebbene il modello sia concettualmente valido, il codice originale basato su TensorFlow 1.x è oggi considerato un *legacy artifact*, la cui riproduzione richiede interventi significativi. Ciò rafforza la necessità di protocolli sperimentali trasparenti, documentati e replicabili.

Sintesi

L'analisi statistica riveste un ruolo fondamentale per garantire affidabilità e solidità alle conclusioni tratte dagli esperimenti. La combinazione di:

- verifica delle assunzioni,
- test di significatività,
- misure di effetto,
- intervalli di confidenza,

rappresenta la pratica raccomandata per valutare correttamente il comportamento dei modelli CMG.
Nel Capitolo 3 tali strumenti vengono applicati sistematicamente per analizzare le differenze tra CoDiSum, RACE e KADEL, fornendo una base empirica solida per le conclusioni del presente lavoro.

2.8 Sintesi critica e limiti aperti

La letteratura sulla generazione automatica dei messaggi di commit (Commit Message Generation, CMG) ha compiuto progressi significativi negli ultimi anni, grazie all'applicazione di modelli neurali sempre più sofisticati, all'integrazione di meccanismi di *retrieval* semantico e alla definizione di strategie per mitigare il rumore presente nei dataset. Tuttavia, l'analisi dello stato dell'arte evidenzia anche una serie di criticità strutturali che ostacolano la valutazione affidabile e comparabile dei modelli esistenti.

Ruolo e qualità dei dati

Uno dei principali limiti riguarda la qualità intrinseca dei dataset disponibili. Numerosi lavori mostrano che una parte significativa dei commit presenti nei repository open source è caratterizzata da messaggi incompleti, minimali o del tutto assenti [7, 9]. Questo introduce rumore nei dati di addestramento e influenza direttamente la performance dei modelli, che possono apprendere pattern inutili (es. "update", "fix") o descrizioni non informative.

Inoltre, dataset diversi applicano criteri eterogenei di filtraggio, pulizia e segmentazione dei diff, rendendo difficile confrontare i risultati tra studi differenti. Il dataset MCMD [11] rappresenta un primo tentativo di affrontare questa limitazione tramite una selezione più accurata dei commit "informativi", ma rimangono aperti interrogativi sulla copertura, sulla generalizzazione e sull'annotazione manuale delle motivazioni (*rationales*) nei messaggi.

Limiti dei modelli attuali

I modelli CMG di ultima generazione seguono tre linee principali:

1. **modelli seq2seq puri** (es. CoDiSum [1]),
2. **approcci basati su retrieval semantico** (es. RACE [2]),
3. **strategie di denoising supervisionato** (es. KADEL [10]).

Ognuna di queste strategie presenta vantaggi e limiti specifici:

- CoDiSum è efficace nel preservare la struttura sintattica tramite meccanismi di copia, ma mostra limiti nella cattura del "*why*" e nella generalizzazione semantica.
- RACE sfrutta il retrieval per recuperare conoscenza contestuale, migliorando informatività e coerenza, ma è sensibile alla qualità degli esempi recuperati e all'indice semantico utilizzato.
- KADEL riduce il rumore presente nei dataset reali, ma richiede euristiche e classificatori aggiuntivi per stimare la qualità dei messaggi, aumentando la complessità della pipeline.

Rimane inoltre una sfida aperta la capacità dei modelli di discriminare tra descrizioni operative (il "*what*") e motivazioni della modifica (il "*why*"), componente informativa rara nei dataset esistenti e difficile da inferire automaticamente.

Problemi di valutazione e comparabilità

La valutazione rimane uno degli aspetti più critici. Come discusso nella Sezione 2.6, la presenza di:

- varianti non standardizzate di metriche (BLEU, ROUGE, METEOR),
- tokenizzatori diversi,
- formati di diff eterogenei,
- pre-processing non documentati,

rende difficile interpretare correttamente i miglioramenti riportati nei lavori.
Le metriche lessicali mostrano correlazione limitata con la qualità percepita dagli sviluppatori: modelli con BLEU elevato possono produrre messaggi semanticamente poveri, mentre parafrasi corrette sono penalizzate dal confronto n-gram. Questo ha portato alla crescente adozione di metriche semantiche (es. BERTScore) e alla reintroduzione di valutazioni umane controllate, come in [2].

Tuttavia, anche le valutazioni umane presentano limiti — costi elevati, variabilità tra annotatori, difficoltà di scalare — e non sempre sono replicabili senza protocolli dettagliati.

Riproducibilità e replicabilità dei risultati

Un ulteriore limite è rappresentato dalla difficoltà di riprodurre gli esperimenti descritti nei paper originali. Problemi comuni includono:

- dipendenze software non più supportate (es. CoDiSum basato su TensorFlow 1.x),
- codebase incomplete o non pubbliche,
- mancanza di dettagli sugli ambienti sperimentali,
- assenza di seed fissati e configurazioni standardizzate.

Queste problematiche si inseriscono in un quadro più ampio di difficoltà nella riproducibilità computazionale [23] e nella comparabilità empirica tra modelli di machine learning.

Sintesi dei limiti principali

In conclusione, la letteratura attuale evidenzia quattro limiti critici:

1. **Rumore nei dataset**: presenza di commit non informativi, incoerenti o privi del "why".
2. **Limitazioni dei modelli**: difficoltà nel catturare motivazioni, dipendenza dal retrieval o da euristiche di denoising.
3. **Problemi di valutazione**: metriche non standardizzate e debolmente correlate con la qualità semantica.
4. **Scarsa riproducibilità**: difficoltà a replicare esperimenti e verificare i risultati riportati nei paper.

Questi limiti motivano la necessità di studi sperimentali più rigorosi, protocolli standardizzati e dataset curati, fornendo al contempo il contesto scientifico che giustifica il disegno metodologico descritto nel Capitolo 3.

CAPITOLO 3

Design e metodologia sperimentale

3.1 Obiettivi sperimentali e domande di ricerca

Il presente studio adotta un approccio empirico alla valutazione dei modelli di **generazione automatica dei messaggi di commit** (Commit Message Generation, CMG). L'obiettivo è analizzare come differenti architetture, dataset e strategie di addestramento influiscano sulla qualità dei messaggi generati, sulla loro stabilità e, in maniera complementare, sui costi computazionali associati.
L'esperimento è strutturato lungo tre dimensioni fondamentali:

- **Modello** — confronto tra tre paradigmi:
  - *seq2seq ibrido* (CoDiSum);
  - *retrieval-augmented generation* (RACE);
  - *denoising-aware encoder–decoder* (KADEL).
- **Dataset** — utilizzo di due corpora distinti e non direttamente interscambiabili:
  - **D1**: dataset Java originale di CoDiSum, ricostruito a partire dalla versione V12 (Sezione 3.4);

  - **D2**: sottoinsieme Java del dataset MCMD [11], pre-processato in forma di *edit actions* (Sezione 3.5).

- **Ripetizione sperimentale** — esecuzione di multiple run con seed differenti, per stimare la variabilità interna dei modelli e la loro sensibilità all'inizializzazione casuale.

La scelta di utilizzare due dataset distinti è imposta dalla natura degli artefatti disponibili e dai requisiti di input dei modelli.
CoDiSum opera su *diff* patch-based testuali e non è compatibile con la rappresentazione in *edit actions* adottata in D2; viceversa, RACE e KADEL richiedono esplicitamente *edit sequences* come input e non possono essere applicati direttamente al dataset D1. Di conseguenza:

- CoDiSum viene valutato esclusivamente su D1;
- RACE e KADEL vengono valutati esclusivamente su D2.

Questa asimmetria ha un impatto diretto sulle domande di ricerca, in particolare su RQ2: il confronto su *dataset condiviso* risulta, per vincoli tecnici, limitato ai soli modelli compatibili con D2 (RACE e KADEL), mentre CoDiSum resta escluso da tale confronto diretto.

3.1.1 Domande di ricerca e ipotesi

Le domande di ricerca sono formulate per analizzare in modo sistematico tre dimensioni chiave della CMG: *riproducibilità nel tempo* di un modello esistente, *qualità comparativa* su dataset condiviso e *stabilità interna* rispetto a run multiple.

RQ1 — Riproducibilità di un modello legacy
In che misura è possibile ricostruire e rieseguire oggi un modello classico di CMG (CoDiSum) sulla base delle risorse pubbliche disponibili, mantenendo coerenza con le specifiche e i risultati riportati nel lavoro originale?
Questa domanda considera la *riproducibilità nel tempo* di un artefatto legacy, alla luce di fenomeni di *dependency decay* e di obsolescenza delle dipendenze software [11, 23].
$H0_1$: non vi sono differenze significative tra i risultati ottenuti in questo studio e i

risultati attesi sulla base della documentazione e degli artefatti disponibili.
$H1_1$: la riesecuzione produce risultati significativamente diversi, suggerendo la presenza di dependency decay, incompletezza degli artefatti o discrepanze non documentate nel setup sperimentale originale.
Metriche principali: BLEU, ROUGE-L, METEOR, SacreBLEU, CodeBLEU-text, *identifier recall*.

RQ2 — Qualità su dataset condiviso

Quali differenze emergono tra modelli di CMG quando vengono addestrati e valutati sul medesimo dataset e con pipeline sperimentali comparabili?

In pratica, RQ2 viene investigata confrontando RACE e KADEL sul dataset D2 (Java–MCMD), unico scenario in cui i due modelli condividono lo stesso formato di input e la stessa base di commit. CoDiSum non è incluso in questo confronto diretto, poiché richiederebbe un adattamento sostanziale del formato di input che ne snaturerebbe il comportamento originario.
$H0_2$: non vi sono differenze significative nelle metriche di qualità tra i modelli considerati sul dataset condiviso (RACE vs KADEL su D2).
$H1_2$: almeno un modello presenta prestazioni significativamente diverse dalle alternative, indicando vantaggi o svantaggi sistematici in termini di qualità dei messaggi generati. *Metriche principali*: BLEU, ROUGE-1/2/L, METEOR, SacreBLEU, CodeBLEU-text, CIDEr, *identifier recall*.

RQ3 — Stabilità e riproducibilità interna

I modelli di CMG presentano risultati stabili al variare del seed di inizializzazione e di altri fattori non deterministici all'interno dello stesso studio?

Questa domanda riguarda la *ripetibilità interna* dell'esperimento (talvolta indicata come *internal reproducibility*): a parità di dataset e configurazione, piccole variazioni nelle inizializzazioni dovrebbero produrre risultati numericamente vicini, con varianza limitata [22, 24].
$H0_3$: non vi sono differenze significative tra più run replicate dello stesso modello sullo stesso dataset.
$H1_3$: almeno un modello mostra variabilità significativa tra run, indicando una sensibilità

Per ciascuna combinazione modello–dataset vengono raccolti dati quantitativi relativi alla qualità, alla stabilità inter-run e, in maniera descrittiva, all'efficienza computazionale (tempi di addestramento, throughput, uso di memoria GPU). Tali dati vengono analizzati secondo il protocollo statistico descritto nella Sezione 3.2 e discussi nel Capitolo 4.

3.2 Schema di analisi statistica

L'analisi statistica è stata progettata per valutare in modo rigoroso:

- le differenze tra modelli in termini di qualità dei messaggi generati;
- la stabilità dei risultati al variare del seed;
- il rapporto tra qualità ottenuta e costo computazionale osservato.

Il protocollo segue le raccomandazioni correnti per esperimenti con algoritmi di apprendimento automatico e più run indipendenti [22, 24, 25].

Pipeline di analisi

Per ogni modello (CoDiSum, RACE, KADEL) e per il dataset a esso associato (D1 o D2) sono state raccolte:

- **metriche di qualità**: BLEU, ROUGE-1/2/L, METEOR, SacreBLEU, CodeBLEU-text, CIDEr, *identifier recall*;
- **indicatori di stabilità**: media, deviazione standard e varianza inter-run per ciascuna metrica;
- **misure di efficienza computazionale**: durata complessiva dell'addestramento, tempo medio per epoca, throughput (commit/s) e, ove disponibile, stima dell'utilizzo di memoria GPU.

Le misure ottenute vengono elaborate secondo i passaggi descritti di seguito.

1. Verifica delle assunzioni statistiche. Prima di procedere al confronto tra modelli o tra run dello stesso modello, vengono verificate le assunzioni dei test parametrici:

- *Normalità* delle distribuzioni delle differenze tramite test di **Shapiro–Wilk**;
- *Omogeneità delle varianze* tramite test di **Levene**.

Qualora entrambe le assunzioni risultino soddisfatte (soglia $\alpha = 0{,}05$), si può procedere con test parametrici; in caso contrario, si ricorre a test non-parametrici [22, 25].

2. Confronto tra modelli su dataset condiviso (RQ2). Per confrontare le prestazioni di RACE e KADEL su D2:

- se le assunzioni di normalità e omoscedasticità sono rispettate, si applica un **t-test per campioni indipendenti** sulle medie delle run;
- in caso contrario, si utilizza il test non parametrico **Mann–Whitney U**.

Per ogni metrica vengono riportati il valore p e la differenza media tra i modelli.

3. Analisi della stabilità intra-modello (RQ3). Per valutare la stabilità rispetto al seed, si analizzano le distribuzioni delle metriche su più run:

- nel caso di confronti appaiati (es. tre run dello stesso modello su D2), si applica un **t-test per campioni appaiati** se le assunzioni parametriche sono soddisfatte;
- in alternativa, si utilizza il test **Wilcoxon signed-rank**, più adatto a distribuzioni non gaussiane o con outlier.

In aggiunta ai test, vengono riportate varianza e coefficienti di variazione per stimare la sensibilità dei modelli al seed.

4. Dimensione dell'effetto. Per interpretare l'importanza pratica delle differenze, oltre alla significatività statistica si calcola la **dimensione dell'effetto**:

- **Cohen's** d per confronti parametrici (t-test), secondo le soglie comunemente adottate ($d \approx 0{,}2$ piccolo, 0,5 medio, 0,8 grande) [26];
- **rank-biserial correlation** per test non-parametrici (Mann–Whitney, Wilcoxon).

Queste misure consentono di distinguere differenze statisticamente significative ma marginali da differenze rilevanti sul piano applicativo.

5. Intervalli di confidenza. Per ciascun modello e per ciascuna metrica vengono riportati:

- media e deviazione standard sulle run;
- **intervalli di confidenza al 95%** delle medie.

Gli intervalli di confidenza sono utilizzati anche per verificare la sovrapposizione tra modelli: intervalli non sovrapposti indicano differenze potenzialmente rilevanti [24].

6. Correzione per confronti multipli. Dato l'elevato numero di metriche considerate, viene applicato il metodo di **Holm–Bonferroni** per controllare il *family-wise error rate* ed evitare un eccesso di falsi positivi dovuti a test multipli [22].

Sintesi

Nel complesso, lo schema di analisi è concepito per fornire:

1. una valutazione comparativa statisticamente fondata tra i modelli su dataset condiviso (RQ2);
2. una stima quantitativa della stabilità intra-modello al variare del seed (RQ3);
3. una misura del rapporto qualità/costo computazionale (tramite statistiche descrittive sui tempi di addestramento);
4. un quadro coerente che colleghi i risultati numerici alle ipotesi formulate in Sezione 3.1.1.

I risultati dettagliati delle analisi saranno presentati e discussi nel Capitolo 4.

3.3 Selezione dei modelli e motivazioni della scelta

La letteratura sulla *Commit Message Generation* (CMG) ha attraversato, in meno di un decennio, tre fasi principali: (i) modelli sequenziali basati su RNN con meccanismi di

attenzione e copia; (ii) modelli ibridi *retrieval-augmented* basati su encoder–decoder Transformer; (iii) metodi recenti che integrano tecniche di *denoising* e apprendimento basato sulla qualità dei dati. Per rappresentare ciascuna di queste fasi sono stati selezionati tre modelli: **CoDiSum**, **RACE** e **KADEL**. La loro combinazione permette di valutare l'evoluzione delle architetture CMG e il loro impatto sulla qualità, sull'efficienza e sulla stabilità delle generazioni, tenendo conto al tempo stesso della reale disponibilità degli artefatti software.

3.3.1 CoDiSum

Il modello **CoDiSum** [1] è uno dei primi tentativi di integrare informazioni strutturali e semantiche del codice nella generazione dei messaggi di commit.
Il paper originale descrive un'architettura ibrida composta da:

- una rappresentazione **strutturale** del diff basata su *AST* e *action-level features*;
- una rappresentazione **semantica** derivata dalla segmentazione degli identificatori in sottoparole;
- un **pointer-generator** per gestire token fuori vocabolario (*OOV*);
- un modello seq2seq RNN (GRU) con meccanismi di attenzione.

Artefatto pubblico e discrepanze. La repository pubblica del modello[1] non coincide integralmente con l'architettura descritta nel paper. Il codice rilasciato implementa un modello denominato *CopyNetPlus*, che:

- include un **copying mechanism** (modulo `MaskedCopyProb`) per gli identificatori;
- **non** include la rappresentazione strutturale su AST;
- **non** applica la sostituzione dinamica degli identificatori tramite placeholder;
- **non** implementa la doppia vista (strutturale + semantica) prevista nel modello originale;

[1] `https://github.com/SoftWiser-group/CoDiSum`

- dipende da versioni legacy di Keras/TensorFlow (Keras 2.1.5, TensorFlow 1.x), non più supportate in ambienti moderni.

Queste discrepanze sono un esempio concreto di *dependency decay* e di distanza tra modello concettuale e artefatto eseguibile [11]. Esse incidono direttamente sulla possibilità di riprodurre i risultati del 2019 e quindi sulla valutazione di RQ1 (riproducibilità).

Configurazione adottata in questa tesi. In questo lavoro CoDiSum viene analizzato nella forma *effettivamente disponibile* come artefatto pubblico, ossia nella variante *CopyNetPlus* basata su diff patch-based testuali (dataset D1, Sezione 3.4).
Piuttosto che tentare di reingegnerizzare ex novo la componente AST mancante, si è scelto di:

- ricostruire il dataset storico CoDiSum V12 (Sezione 3.4) partendo dai 1 000 repository Java originali;
- adattare l'ambiente di esecuzione per rendere possibile il training della versione pubblica del modello;
- mantenere inalterata l'architettura concettuale dell'artefatto rilasciato (CopyNetPlus), documentando le differenze rispetto alla descrizione originaria del paper.

Di conseguenza, quando in questa tesi ci si riferisce ai risultati di **CoDiSum**, si intende sempre la configurazione effettivamente ottenibile a partire dall'artefatto pubblico, e non una replica completa (inclusiva di AST) del modello ideale descritto nel lavoro del 2019.
Questa scelta è deliberata e costituisce parte integrante della valutazione empirica di RQ1 (quanto è realmente riproducibile oggi un modello CMG *legacy*).

3.3.2 RACE

Il modello **RACE** (*Retrieval-Augmented Commit Message Generation*) [2] introduce un paradigma differente: la generazione del messaggio è guidata da un commit di

esempio recuperato tramite **retrieval semantico**. La pipeline è composta, in sintesi, da due moduli principali:

1. **Retrieval**: un encoder CodeT5 produce embedding dense del diff; tramite similarità coseno si recupera un commit semanticamente simile all'interno del training set (dataset MCMD [11]).
2. **Generazione**: un decoder Transformer produce il messaggio finale combinando informazioni dal diff in input, dal diff recuperato e dal commit message di esempio, mediati dall'*exemplar guider*.

Questo approccio sfrutta esplicitamente la conoscenza "storica" presente nel dataset: commit simili tendono a condividere messaggi simili, e il modello apprende a trasferire stile e contenuto dall'esempio recuperato al nuovo commit.

Formato dei dati. RACE richiede che le modifiche al codice siano rappresentate come **edit actions** (`<INSERT>`, `<DELETE>`, `<REPLACE>`, `<KEEP>`), secondo lo schema proposto da Panthaplackel et al. [17]. Il formato dei dati richiesto dal modello corrisponde al formato del dataset **MCMD** rilasciato su Zenodo [11], che costituisce il dataset D2 utilizzato in questo lavoro (Sezione 3.5).

Configurazione adottata in questa tesi. Per RACE, in questa tesi si è scelto di:

- utilizzare l'implementazione open-source rilasciata dagli autori, limitandosi al sottoinsieme Java di MCMD (dataset D2);
- mantenere **CodeT5-base** come backbone principale, in linea con la configurazione "base" descritta nel paper originale;
- adattare gli script di preprocessing per generare le edit actions e i pair *(diff, commit simile)* a partire dal subset Java di MCMD;
- seguire, per quanto possibile, le stesse scelte di hyperparameter del lavoro originale (numero di epoche, batch size, lunghezza massima delle sequenze), documentando le eventuali deviazioni nelle Sezioni 3.8.

RACE non è stato applicato al dataset storico di CoDiSum (D1), poiché ciò avrebbe richiesto una conversione non banale da patch testuali a edit actions, con il rischio di introdurre bias di pre-processing e di allontanarsi dalla pipeline originale descritta in [2]. Per questa ragione, RACE viene addestrato e valutato esclusivamente su D2.

3.3.3 KADEL

Il modello **KADEL** (*Knowledge-Aware Denoising Learning*) [10] rappresenta una delle evoluzioni più recenti dei modelli CMG. La sua caratteristica distintiva consiste nell'integrare:

- un meccanismo di **estrazione della conoscenza** da commit di alta qualità (good-practice commits);
- un **denoising dinamico** che riduce l'impatto dei messaggi rumorosi durante l'addestramento;
- un'architettura encoder–decoder basata su **CodeT5-base**, ottimizzata per sequenze lunghe e contesti eterogenei.

KADEL è progettato specificamente per dataset ampi e rumorosi come MCMD, in cui coesistono commit di ottima qualità con messaggi generici, incompleti o automatici. In questo contesto, il modello cerca di imparare maggiormente dai commit "di buona pratica" trasferendo tale conoscenza agli esempi più rumorosi.

Compatibilità dei dati. Analogamente a RACE, KADEL richiede il formato a *edit actions* e meta-informazioni aggiuntive (ad esempio etichette di qualità o subset di good-practice commits) estratte dal dataset MCMD [11, 10]. Per questo motivo, anche KADEL viene addestrato ed analizzato esclusivamente sul dataset D2 (Java–MCMD).

Configurazione adottata in questa tesi. In questa tesi:

- si adotta l'implementazione pubblica di KADEL rilasciata dagli autori, con backbone CodeT5-base;

- si utilizza il sottoinsieme di **good-practice commits** definito in [10] per costruire la componente di knowledge-aware learning;
- si restringe l'analisi al linguaggio Java del dataset MCMD, per garantire coerenza con RACE e con il focus di questa tesi;
- si mantengono, ove possibile, le impostazioni di training del paper originale, riportando in Sezione 3.9 le eventuali differenze (ad es. numero di epoche, early stopping).

3.3.4 Sintesi e motivazioni della scelta

La scelta dei tre modelli è motivata da una combinazione di fattori storici, metodologici e pratici:

- **Rappresentatività storica** CoDiSum rappresenta la prima generazione di modelli ibridi seq2seq con meccanismi di copia, ampiamente utilizzato come baseline in studi successivi.
- **Rappresentatività metodologica** RACE incarna il paradigma *retrieval-augmented generation*, oggi centrale anche in molti sistemi LLM.
- **Rappresentatività dello stato dell'arte** KADEL integra tecniche di denoising e knowledge-aware training, affrontando esplicitamente il problema della qualità dei dati.
- **Disponibilità e realismo degli artefatti** RACE e KADEL sono supportati da implementazioni open-source aggiornate; CoDiSum è disponibile solo in una variante incompleta e legacy (CopyNetPlus), il che rende il suo caso emblematico per studiare la riproducibilità di modelli CMG storici.
- **Compatibilità con i dati** CoDiSum opera su diff patch-based (dataset D1); RACE e KADEL richiedono edit actions e meta-informazioni provenienti da MCMD (dataset D2).

Nel complesso, CoDiSum, RACE e KADEL coprono l'evoluzione della CMG dal 2019 al 2024, consentendo un'analisi empirica che combina prospettiva storica (riproducibilità di un modello legacy), confronto su dataset condiviso fra modelli moderni (RACE vs KADEL) e valutazione congiunta di qualità, stabilità e costi computazionali.

3.4 Costruzione del dataset CoDiSum (V12 ricostruito)

Il primo dataset utilizzato in questo lavoro è una **ricostruzione controllata** del corpus originale associato al modello **CoDiSum**. Nel lavoro di Xu et al. [1] gli autori descrivono il processo di raccolta dei commit (1 000 repository Java più popolari su GitHub) e riportano statistiche aggregate (circa 90 000 coppie *(diff, message)*), ma non rilasciano un dataset già formattato per modelli seq2seq moderni. Il materiale disponibile consiste in un insieme di file JSON (**versione V12**) utilizzati internamente dal loro sistema.

Ricostruire un dataset `src-tgt` a partire da questi file rappresenta quindi un passaggio necessario per poter riaddestrare CoDiSum e, più in generale, per rendere riutilizzabile il corpus come benchmark per modelli contemporanei. Questa ricostruzione costituisce uno dei contributi empirici di questa tesi in relazione a RQ1 (riproducibilità).

3.4.1 Struttura dei file originari

Il pacchetto CoDiSum include una directory `raw/` composta da undici file JSON. I più rilevanti per la ricostruzione sono:

- `difftextV12.json`: rappresentazione testuale dei diff (99 MB);
- `msgtextV12.json`: messaggi di commit associati (6 MB);
- `difftokenV12.json`: tokenizzazione del diff;
- `diffattV12.json`: marcature di attenzione e segnali strutturali;

- `copymaskV12.json`, `genmaskV12.json`: maschere per il meccanismo di copia;
- `variableV12.json`: annotazioni sugli identificatori;
- `word2indexV12.json`: vocabolario legacy (oltre 870 000 termini).

Questi file rappresentano le componenti interne del sistema CoDiSum, ma non sono direttamente utilizzabili per addestrare modelli seq2seq standard: mancano infatti file `train/valid/test` espliciti in formato testuale `src-tgt`. È quindi necessaria una fase di ricostruzione del dataset.

3.4.2 Pipeline di ricostruzione

La conversione in formato seq2seq è stata effettuata tramite lo script `build_codisum_dataset.py`[2], progettato per essere ripetibile.
La pipeline comprende i seguenti passaggi:

1. **Caricamento** dei file `difftextV12.json` e `msgtextV12.json`, mantenendo gli indici originali come chiave di allineamento.
2. **Verifica di consistenza**: controllo che ogni diff abbia esattamente un messaggio associato e scarto di eventuali record inconsistente (diff mancanti o messaggi nulli).
3. **Pulizia testuale** dei messaggi:
   - rimozione di caratteri speciali spurî e token di controllo,
   - normalizzazione degli spazi e dei ritorni a capo,
   - eliminazione di messaggi vuoti o triviali (ad es. stringhe costituite solo da spazi).
4. **Split riproducibile** 80–10–10 in `train/valid/test`, ottenuto con campionamento casuale e `seed = 42`, in modo da consentire future repliche dello stesso partizionamento.

[2]Script utilizzato per la ricostruzione: `build_codisum_dataset.py`.

5. **Generazione dei file finali** in formato compatibile con pipeline seq2seq:
   - `train.txt.src` / `train.txt.tgt`,
   - `valid.txt.src` / `valid.txt.tgt`,
   - `test.txt.src` / `test.txt.tgt`.

Il dataset finale contiene **90 661 coppie** *(diff, message)*, valore coerente con le dimensioni riportate nella letteratura originale [1].

3.4.3 Analisi empirica del dataset ricostruito

L'analisi quantitativa è stata realizzata tramite lo script `analyze_codisum_dataset.py`[3], che calcola:

- lunghezza media dei diff e dei messaggi (in token);
- distribuzione della lunghezza per split;
- numero di linee e token unici;
- *type/token ratio* (TTR);
- messaggi più frequenti;
- un campionamento qualitativo di diff e messaggi.

Le statistiche principali sono riportate in Tabella 3.1.

Split	Numero di esempi	Lunghezza media diff	Lunghezza media msg
`train`	72 528	81.50 token	6.58 token
`valid`	9 066	80.36 token	6.60 token
`test`	9 067	80.00 token	6.61 token

Tabella 3.1: Statistiche empiriche del dataset CoDiSum V12 ricostruito.

[3]Script utilizzato per l'analisi: `analyze_codisum_dataset.py`.

3.4.4 Proprietà qualitative

L'analisi qualitativa conferma alcune caratteristiche tipiche dei dataset CMG:

- **Elevata variabilità sintattica del codice** (TTR dei diff nell'intervallo 0.14–0.21), indice dell'eterogeneità dei progetti Java selezionati.
- **Messaggi estremamente brevi** (6–7 token medi), coerenti con la lunghezza tipica dei commit message reali.
- **Ridotta presenza di duplicati** (oltre il 99% dei diff e il 95–98% dei messaggi sono unici), il che limita il rischio di overfitting su pattern ripetuti.
- Presenza di pattern ricorrenti come *"remove unused imports"*, *"fixed lint error"*, *"fix bug"*, tipici di commit di manutenzione.
- Prevalenza di commit di manutenzione correttiva ed evolutiva, in linea con quanto riportato in studi empirici su larga scala [5].

3.4.5 Esempi rappresentativi

Esempio 1 — `TGT: Add UNDEMOTE_METHOD to ContactsContract`
SRC: diff –git a/core/java/android/provider/ContactsContract.java ...

Esempio 2 — `TGT: Fix assertion if FS root is not readable`
SRC: diff –git a/platform/platform-impl/src/com/intellij/openapi/vfs/...

Esempio 3 — `TGT: fixed lint error`
SRC: diff –git a/src/eu/siacs/conversations/ui/EditAccountActivity.java ...

3.4.6 Vantaggi del dataset CoDiSum ricostruito

Il dataset ottenuto presenta tre vantaggi principali per la sperimentazione:

- **Baseline storica consolidata**: rappresenta uno dei corpus più utilizzati nei primi modelli CMG e permette di collocare i risultati in continuità con la letteratura [1].

- **Controllo della qualità**: dataset relativamente poco rumoroso (percentuale limitata di messaggi generici o mancanti), utile per analizzare i casi di fallimento dei modelli seq2seq su dati "classici".
- **Compatibilità moderna**: la conversione in formato `src-tgt` consente di utilizzare il corpus con modelli contemporanei (encoder–decoder Transformer, LLM) mantenendo coerenza con gli studi precedenti.

La ricostruzione V12 consente inoltre di ottenere un artefatto **replicabile e tracciabile**, elemento cruciale per l'analisi empirica della riproducibilità del modello CoDiSum affrontata nel Capitolo 4.

3.5 Dataset MCMD (Java subset) per RACE e KADEL

Per l'addestramento e la valutazione dei modelli **RACE** e **KADEL** è stato utilizzato un dataset moderno e ricco di contesto, derivato dal corpus **MCMD** (*Multilingual Commit Message Dataset*) introdotto da Tao et al. [11] e rilasciato su Zenodo[4].
Questo corpus, oggi considerato uno dei benchmark di riferimento per i modelli di commit message generation basati su Transformer [2, 10], contiene oltre 900 000 coppie *(diff, message)* provenienti da cinque linguaggi di programmazione: Java, C#, C++, Python e JavaScript.
Poiché lo scopo di questo lavoro è il confronto diretto tra modelli all'interno dello stesso linguaggio, è stato selezionato **esclusivamente il subset Java** del dataset.
Tale scelta garantisce coerenza metodologica con il dataset storico di CoDiSum (anch'esso Java) e permette di condurre la RQ2 (confronto su dataset condiviso) in un contesto monolingua omogeneo.

3.5.1 Origine dei dati e filtraggio del linguaggio

Dallo `tar.gz` distribuito su Zenodo è stata estratta la seguente struttura:

```
dataset/
cpp/
```

[4] https://zenodo.org/record/7196966

```
csharp/
java/
javascript/
python/
```

La directory `java/` contiene varie versioni del dataset, tra cui:

- **dataset JSONL originale** — coppie diff–message in formato grezzo;
- **contextual_medits/** — rappresentazione del diff tramite *edit actions*, proposta da Panthaplackel et al. [17];
- **codet5_retrieval_result/** — versione arricchita che include, per ogni esempio:
  - il diff originale tokenizzato in *edit actions*;
  - un commit simile recuperato tramite CodeT5;
  - il relativo messaggio del commit simile;
  - la similarità stimata.

Per garantire piena compatibilità con la pipeline di RACE [2], questo lavoro utilizza esclusivamente la versione textttjava/contextual_medits/codet5_retrieval_result, che rappresenta esattamente il formato di input previsto dagli autori (diff + commit simile + messaggio simile).

3.5.2 Preprocessing e costruzione del subset finale

Il corpus Java completo è di dimensioni elevate e rende oneroso l'addestramento multiplo con diverse configurazioni e seed. Per bilanciare **costo computazionale** e **dimensione del campione**, è stato creato un **subset Java** più compatto, mantenendo la struttura delle istanze e la distribuzione dei diff.
Il preprocessing è stato realizzato tramite tre script dedicati:

- `check_dataset.py` — verifica della struttura e dei campi presenti in ogni file JSONL;

- `make_small_dataset.py` — estrazione casuale di un sottoinsieme bilanciato del dataset preservando la distribuzione delle lunghezze;
- `make_small_with_retrieval.py` — generazione dei file finali compatibili con le pipeline di RACE e KADEL:
  - `train_small.jsonl`,
  - `valid_small.jsonl`,
  - `test_small.jsonl`.

Lo schema di suddivisione finale è il seguente:

- **train_small.jsonl**: 50 000 esempi;
- **valid_small.jsonl**: 5 000 esempi;
- **test_small.jsonl**: 5 000 esempi.

Il campionamento è stato effettuato in modo riproducibile con seed fisso, e la tokenizzazione originale è stata mantenuta integralmente, preservando l'informazione strutturale necessaria ai modelli Transformer.

3.5.3 Struttura delle istanze

Un'istanza tipica del dataset Java appare nella seguente forma:

```
{
  "diff": ["<KEEP>", "public", "void", "foo", "(", ")", "{", ... ],
  "msg_token": ["fix", "null", "pointer", "exception", "in", "parser"],
  "retrieved_diff": [...],
  "retrieved_msg": [...],
  "similarity": 0.81
}
```

Rispetto al dataset di CoDiSum, il diff è **più lungo e strutturato**: le *edit actions* permettono di distinguere con precisione le operazioni compiute sul codice e offrono un contesto ricco per modelli come RACE e KADEL [2, 10].

3.5.4 Statistiche del dataset

Le statistiche principali del subset Java sono riportate nella Tabella 3.2, calcolate tramite uno script dedicato basato su `pandas`.

Split	Numero di esempi	Lunghezza media diff	Lunghezza media msg
`train`	50 000	475.18 token	8.70 token
`valid`	5 000	518.88 token	8.75 token
`test`	5 000	418.56 token	8.70 token

Tabella 3.2: Statistiche principali del dataset MCMD (Java subset).

3.5.5 Osservazioni empiriche

Dall'analisi del dataset emergono alcune caratteristiche rilevanti per l'addestramento dei modelli:

- i diff sono **notevolmente più lunghi** rispetto a quelli del dataset CoDiSum (circa 80 token), con valori tipicamente compresi tra 300 e 700 token;
- i messaggi restano **molto concisi**, con una media di circa 9 token, in linea con la pratica industriale osservata in [11];
- i marcatori `<KEEP>`, `<INSERT>`, `<DELETE>` arricchiscono il diff con informazione strutturale utile ai modelli Transformer;
- la presenza del commit simile recuperato costituisce un vantaggio diretto per RACE, che usa il retrieval per guidare la generazione [2];
- la maggiore complessità del dataset lo rende un banco di prova più impegnativo rispetto al dataset CoDiSum, particolarmente adatto alla valutazione di tecniche di denoising come KADEL [10].

3.5.6 Vantaggi del dataset

Il subset Java del corpus MCMD offre diversi punti di forza per gli esperimenti con RACE e KADEL:

- **Alta variabilità** nei diff e nelle operazioni di editing, che mette alla prova la capacità dei modelli di generalizzare;
- **Compatibilità completa** con la pipeline RACE basata sul retrieval e con l'architettura KADEL basata su edit actions;
- **Scala adeguata** all'addestramento di modelli encoder–decoder moderni, pur rimanendo gestibile su hardware GPU standard;
- **Contesto ricco e informativo**, ideale per valutare strategie di denoising e knowledge-aware learning [10];
- **Coerenza linguistica** con il dominio Java, che rende il confronto con CoDiSum metodologicamente solido pur su dataset differenti.

Nel complesso, il dataset MCMD (Java subset) rappresenta una risorsa moderna, articolata e ben strutturata, fondamentale per valutare la qualità di modelli avanzati come RACE e KADEL e per affrontare RQ2 (confrontosu dataset condiviso) in condizioni realistiche ma controllate.

3.6 Configurazione dell'ambiente sperimentale

Gli esperimenti sono stati condotti in ambiente **Google Colab Pro**, utilizzando GPU **NVIDIA Tesla T4 (16 GB)** e memoria allocata dinamicamente dal servizio.
Poiché i tre modelli analizzati (CoDiSum, RACE, KADEL) si basano su tecnologie differenti e richiedono versioni specifiche di librerie non compatibili tra loro, è stato necessario predisporre **tre ambienti separati**, uno per ciascun modello.
Questa scelta garantisce riproducibilità e isolamento delle dipendenze, evitando conflitti tra versioni di TensorFlow, PyTorch e Transformers.

3.6.1 Ambiente per CoDiSum

CoDiSum si basa su un'implementazione legacy in TensorFlow 1.x, incompatibile con le versioni moderne.
Per eseguire correttamente la pipeline pubblica (`CopyNetPlus`), è stato utilizzato

un ambiente Python 3.10 con librerie aggiornate compatibili con TensorFlow 2.x, integrando manualmente gli strumenti richiesti per le metriche.
L'ambiente include:

- `tensorflow-text` (per tokenizzazione e funzioni ausiliarie),
- `nltk` (WordNet, METEOR),
- `rouge-score`,
- `sacrebleu`,
- `pycocoevalcap` (CIDEr),
- `pygments` (highlighting diff).

Le installazioni principali:

```
pip install nltk rouge-score sacrebleu pycocoevalcap tensorflow-text
pip install git+https://github.com/salaniz/pycocoevalcap
python evaluate.py
```

3.6.2 Ambiente per RACE

RACE richiede una versione moderna di PyTorch e Transformers, compatibili con il checkpoint `CodeT5-base`.
Per garantire coerenza con l'implementazione originale, è stato utilizzato un ambiente Python 3.10 con:

- `torch 2.3.0`,
- `transformers 4.42.3`,
- `tensorboardX` (logging),
- `sacrebleu` e `rouge-score` (metriche),
- `prettytable` (formattazione tabellare).

Comando principale di setup:

```
pip install "torch==2.3.0" "transformers==4.42.3" \
            prettytable tensorboardX sacrebleu
```

3.6.3 Ambiente per KADEL

KADEL si basa su una pipeline antecedente a RACE e richiede una configurazione incompatibile con le versioni recenti di PyTorch e Transformers.
In particolare, la repository originale utilizza:

- Python 3.8,
- `torch 1.9.0`,
- `transformers 4.8.2`,
- `tokenizers 0.10.3`,
- `tree-sitter 0.2.2`,
- `tensorboard 2.4.1`,
- `scikit-learn 0.24.2`,
- `sentencepiece`, `tqdm`, `wandb`,
- `numpy 1.23.5` (versione minima compatibile con torch 1.9 per Python 3.8),
- `protobuf 3.20.3` (necessario per garantire compatibilità con TensorBoard 2.4.1).

La creazione dell'ambiente richiede un'installazione manuale di Python 3.8 in Colab, seguita dall'installazione di pip e delle dipendenze specifiche:

```
sudo apt-get update -y
sudo apt-get install python3.8 python3.8-dev python3.8-distutils -y

curl https://bootstrap.pypa.io/pip/3.8/get-pip.py -o get-pip3.8.py
sudo python3.8 get-pip3.8.py

python3.8 -m pip install torch==1.9.0
python3.8 -m pip install transformers==4.8.2 tokenizers==0.10.3
python3.8 -m pip install tensorboard==2.4.1 scikit-learn==0.24.2
python3.8 -m pip install tree-sitter==0.2.2
```

```
    sacremoses sentencepiece tqdm wandb

# Dipendenze aggiuntive richieste
python3.8 -m pip install "numpy==1.23.5" "protobuf==3.20.3"
```

Questa configurazione permette di riprodurre fedelmente la pipeline degli autori, assicurando compatibilità con i file JSONL del dataset MCMD e con lo schema di *template-guided generation*.

3.6.4 Sintesi

La Tabella 3.3 riassume i tre ambienti sperimentali.

Modello	Python	Framework	Versione
CoDiSum	3.10	TensorFlow	2.x + tool legacy
RACE	3.10	PyTorch + Transformers	torch 2.3.0, HF 4.42.3
KADEL	3.8	PyTorch + Transformers	torch 1.9.0, HF 4.8.2

Tabella 3.3: Riepilogo degli ambienti sperimentali utilizzati per ciascun modello.

Nel complesso, la stratificazione degli ambienti riflette la natura eterogenea dei modelli analizzati: CoDiSum richiede componenti legacy, RACE utilizza un ecosistema moderno, mentre KADEL si colloca in una fase intermedia ma con vincoli rigidi sulle versioni delle librerie.
La separazione degli ambienti garantisce la piena riproducibilità delle sperimentazioni.

3.6.5 Struttura del progetto

Il progetto è stato organizzato in modo modulare, mantenendo separati i tre modelli (CoDiSum, RACE, KADEL) e includendo esclusivamente i file prodotti, modificati o sviluppati durante questo lavoro di tesi.
La struttura seguente omette intenzionalmente le componenti dei repository origi-

nali, già documentate dagli autori, e riporta soltanto gli elementi rilevanti per la sperimentazione, la generazione dei dataset, l'addestramento e la valutazione.

Listing 3.1: Struttura del progetto sperimentale.

```
thesis_project/

 plots_metrics/ # Grafici prodotti per l'analisi comparativa
  bar_mean_std_race_kadel.png
  boxplot_ROUGEL.png
  boxplot_SACREBLEU.png
  plot_race_kadel.py # Script Python per generare i plot

 stats_race_vs_kadel.csv # Metriche aggregate (RACE vs KADEL)
 stats_race_vs_kadel.py # Script di analisi statistica

 CoDiSum/
  codisum-env/ # Ambiente virtuale dedicato
   ...

  models/ # Modelli addestrati per i tre seed
   codisum_best_seed42.h5
   codisum_best_seed123.h5
   codisum_best_seed999.h5
   codisum_final_seed42.h5
   codisum_final_seed123.h5
   codisum_final_seed999.h5

  processed/ # Dataset ricostruito in formato srctgt
   train.txt.src/.tgt
   valid.txt.src/.tgt
   test.txt.src/.tgt

  raw/ # Dataset CoDiSum V12 ricostruito
   difftokenV12.json
   diffttextV12.json
   variableV12.json
   msgV12.json
   ...

  results/ # Log e risultati delle tre run
   run_seed42/
    config_and_env.json
    train_log.csv
   run_seed123/
   run_seed999/

  results_codisum/ # File metriche e predizioni finali
```

```
  codisum_metrics_all_seeds.csv
  codisum_seed42_metrics.json
  codisum_seed42_preds.txt
  ...

 build_codisum_dataset.py # Script sviluppato per ricostruire il dataset
 evaluate_codisum_generate.py # Script di valutazione modificato
 analyze_codisum_dataset.py # Analisi sul dataset ricostruito
 models1.py # Modifiche all'implementazione CopyNetPlus


RACE/
 dataset/ # Dataset contextual-Medits gi fornito
  (solo file JSONL modificati o filtrati)

 results_race/ # Metriche e predizioni per 3 seed
  race_50k_metrics_all_seeds.csv
  race_50k_seed42_metrics.json
  race_50k_seed42_preds.txt
  race_50k_seed42_refs.txt
  ...

 saved_model/ # Checkpoint dei modelli migliori
  RACE_50k_seed42/
   checkpoint-best-bleu/
    pytorch_model.bin
   checkpoint-last/
   summary.log
  RACE_50k_seed123/, RACE_50k_seed999/

 evaluate_race.py # Script adattato per valutazione locale
 check_dataset.py # Controlli di coerenza sugli input
 run_small.sh # Script di esecuzione automatica


KADEL/
 dataset/ # Dataset convertito nel formato KADEL
  convert_dataset.py
  train_small.jsonl
  train.diff.txt
  train.msg.txt
  train.match_template.txt
  (equivalenti per valid/test)

 results_kadel/ # Metriche e predizioni per 3 seed
  kadel_metrics_all_seeds.csv
  kadel_seed42_metrics.json
  kadel_seed42_preds.txt
```

```
   kadel_summary.json
   ...

  saved_models/ # Modelli addestrati
   (checkpoint migliori per seed)

  evaluate_kadel.py # Script di valutazione adattato
  preprocess_MCMD.py # Script modificato per conversione dataset
  utils.py # Utility modificata per gestione template

images/ # Figure inserite nella tesi
    pipeline.png
    codisum_architecture.png
    race_architecture.png
    kadel_attention.png
```

3.7 Fasi sperimentali

Questa sezione descrive in modo dettagliato le fasi sperimentali previste nel presente studio. Le procedure seguono una pipeline comune — preparazione dei dati, configurazione del modello, addestramento, valutazione e analisi statistica — ma vengono adattate a seconda delle caratteristiche dei tre modelli considerati (CoDiSum, RACE e KADEL).
In sintesi:

- **RQ1 (Riproducibilità)** viene affrontata principalmente attraverso CoDiSum su D1 (dataset storico ricostruito) e, in misura minore, verificando la coerenza tra implementazioni pubbliche e paper originali per RACE e KADEL.
- **RQ2 (Qualità su dataset condiviso)** è valutata confrontando **RACE** e **KADEL** sul medesimo dataset D2 (subset Java di MCMD, Sezione 3.5); CoDiSum è escluso da questo confronto diretto per incompatibilità di formato.
- **RQ3 (Stabilità e riproducibilità interna)** viene affrontata eseguendo più run per ciascun modello con seed distinti (42, 123, 999) e analizzando la variabilità inter-run.

Nel seguito viene prima descritta la pipeline sperimentale comune e successivamente le specificità per ciascun modello.

3.7.1 Pipeline sperimentale comune

Per ogni modello vengono eseguiti i seguenti passi:

1. **Preparazione del dataset** (conversione nel formato richiesto dal modello, pulizia, controlli di qualità, split train/valid/test definiti nella Sezione 3.4 per CoDiSum e nella Sezione 3.5 per RACE/KADEL).

2. **Inizializzazione del modello** con configurazione fissa (architettura, hyperparameter principali) e **seed variabile** per le tre run previste (42, 123, 999).

3. **Addestramento** su GPU, monitorando:

   - tempo per epoca e durata complessiva;
   - memoria GPU effettivamente utilizzata;
   - dimensione effettiva del batch (inclusa l'eventuale accumulazione del gradiente);
   - andamento della loss su training e validation.

4. **Valutazione** sul test set mediante:

   - metriche di qualità (BLEU, METEOR, ROUGE-1/2/L, SacreBLEU, CIDEr, CodeBLEU-text, identifier recall);
   - salvataggio delle predizioni e dei riferimenti in formato testuale;
   - raccolta sistematica dei risultati per ciascun seed.

5. **Analisi statistica** secondo il protocollo descritto nella Sezione 3.2, comprendente verifica di normalità/omoscedasticità, test parametrici o non-parametrici, calcolo della dimensione dell'effetto e correzione per confronti multipli.

Nel seguito, la pipeline viene descritta nello specifico per CoDiSum, RACE e KADEL, evidenziandole differenze architetturali e di formato dei dati.

3.7.2 Fasi sperimentali per CoDiSum

L'esperimento su **CoDiSum** è strutturato in tre run indipendenti, con seed pari a **42**, **123** e **999**, in linea con RQ3 sulla stabilità.
CoDiSum viene addestrato e valutato esclusivamente sul dataset ricostruito D1 (Sezione 3.4), e i risultati vengono confrontati con le aspettative del paper originale per rispondere a RQ1 (riproducibilità).
Tutte le run sono state eseguite su GPU **Tesla T4** in ambiente Google Colab.

Fase 1 — Preparazione del dataset

Viene utilizzato il dataset ricostruito CoDiSum V12 (Sezione 3.4), convertito in formato `src-tgt`. Dopo un controllo di integrità dei file (stesso numero di esempi per sorgente e target), i diff vengono preprocessati tramite:

- rimozione di linee vuote e commenti triviali;
- normalizzazione di spazi e caratteri speciali;
- troncamento a **200 token** per garantire stabilità della memoria e tempi di training gestibili.

I messaggi target vengono tokenizzati, convertiti in minuscolo e troncati a **32 token**. La stessa pipeline di preprocessing viene applicata a tutti e tre gli split (train/valid/-test).

Fase 2 — Configurazione dei tokenizzatori

La tokenizzazione è gestita tramite gli strumenti personalizzati presenti nella repository pubblica di CoDiSum (*CopyNetPlus*), con:

- vocabolario massimo di circa **30 000** token;
- token speciali `<pad>`, `<unk>`, `<s>`, `</s>`;
- costruzione delle sequenze `decoder_input` e `decoder_target` mediante shift a destra del target.

Questa fase replica fedelmente la logica del codice pubblico, pur non potendo ricostruire il preprocessing avanzato descritto nel paper (AST, sostituzione dinamica degli identificatori, doppia vista strutturale+semantica).

Fase 3 — Costruzione del modello

Il modello utilizzato per l'esperimento non coincide integralmente con l'architettura descritta da Xu et al. [1], ma è la versione pubblica denominata *CopyNetPlus*. In particolare, è composto da:

- encoder bi-GRU (192 unità, dropout 0.1);
- decoder GRU (97 unità);
- attenzione in stile Luong (*dot attention*);
- meccanismo di copia implementato nel modulo `MaskedCopyProb`.

La mancanza della rappresentazione strutturale del diff (AST, action-level features) e della pipeline di normalizzazione semantica rappresenta una differenza sostanziale rispetto al modello originario e costituisce un potenziale fattore di rischio per la riproducibilità (RQ1).

Fase 4 — Addestramento

Ogni run viene addestrata per un massimo di **10 epoche** con:

- batch size effettivo: 32;
- ottimizzatore: Adam (`lr = 1e-4`);
- assenza di *early stopping*, per rimanere quanto più possibile fedeli alla pipeline originale;
- tempo medio per epoca di circa 9–11 minuti su GPU T4.

Durante l'addestramento vengono loggati loss di training, loss di validation e durata di ciascuna epoca. Queste informazioni sono successivamente utilizzate per analizzare l'eventuale presenza di instabilità o *collapse* del modello.

Fase 5 — Valutazione

La valutazione viene eseguita tramite uno script dedicato (`evaluate_codisum.py`), che per ciascun seed:

- genera i messaggi di commit sul test set;
- salva predizioni e riferimenti in file testuali allineati;
- calcola le metriche aggregate (BLEU, METEOR, ROUGE-L, SacreBLEU, CodeBLEU-text, identifier recall, CIDEr);
- produce un file di log con i risultati per run.

I valori numerici delle metriche e gli esempi di generazioni vengono discussi in dettaglio nel Capitolo 4, nella sezione dedicata alle prestazioni di CoDiSum.

Fase 6 — Analisi e interpretazione sperimentale

Per CoDiSum, le tre run vengono confrontate tra loro (RQ3) e con le aspettative derivanti dal paper originale (RQ1), applicando il protocollo statistico descritto nella Sezione 3.2. Particolare attenzione è rivolta a:

- eventuale **instabilità** delle generazioni (collasso linguistico, frasi stereotipate);
- differenze tra run al variare del seed;
- discrepanze tra il comportamento del modello pubblico *CopyNetPlus* e quello descritto nel lavoro del 2019.

Questa analisi consente di discutere in modo critico la **riproducibilità reale** di CoDiSum rispetto alla descrizione in letteratura.

3.8 Fasi sperimentali per RACE

L'esperimento dedicato al modello **RACE** (*Retrieval-Augmented Commit Message Generation*) segue la stessa metodologia generale di CoDiSum, ma viene condotto sul dataset D2 (MCMD Java subset, Sezione 3.5), compatibile con il formato a *edit actions*

richiesto dal modello.
Anche per RACE sono previste tre run indipendenti con seed **42**, **123** e **999**, in linea con RQ3.
Il confronto diretto con KADEL su D2 fornisce l'evidenza empirica per RQ2.
Tutte le esecuzioni sono state condotte su GPU **Tesla T4** in ambiente Google Colab.

Fase 1 — Preparazione del dataset

RACE è addestrato sul subset **50k** del corpus MCMD (Sezione 3.5), composto da:

- 50 000 esempi di training;
- 5 000 esempi di validazione;
- 5 000 esempi di test.

Gli input corrispondono direttamente ai campi `diff`, `retrieved_diff`, `retrieved_msg` presenti nei file `train_small.jsonl`, `valid_small.jsonl`, `test_small.jsonl`.
Il preprocessing applicato comprende:

- verifica della presenza di tutte le chiavi attese (diff, msg_token, retrieved_diff, retrieved_msg);
- rimozione di esempi corrotti o con diff vuoto;
- troncamento del diff a **200 token** e del messaggio a **30 token**, in linea con la configurazione originale di RACE [2];
- conversione in formato testuale intermedio compatibile con gli script di training del modello.

Fase 2 — Configurazione dei tokenizzatori

La tokenizzazione è basata sul modello **CodeT5-base** (Salesforce), arricchito con i token speciali che codificano le operazioni del diff:

- `<REPLACE>`, `<REPLACE_OLD>`, `<REPLACE_NEW>`, `<REPLACE_END>`;
- `<INSERT>`, `<INSERT_OLD>`, `<INSERT_NEW>`, `<INSERT_END>`;

- `<DELETE>`, `<DELETE_END>`;
- `<KEEP>`, `<KEEP_END>`.

Questi token rappresentano le azioni elementari di modifica del codice (*edit actions*) e consentono al modello di ragionare sulla struttura del diff anziché sul semplice patch testuale.

Fase 3 — Architettura del modello

L'implementazione utilizzata in questo lavoro segue la descrizione fornita da Shi et al. [2] e utilizza:

- un encoder–decoder **CodeT5-base** come backbone;
- un modulo di **retrieval** basato su embedding CodeT5 e similarità coseno per selezionare, per ciascun diff, un commit simile nel training set (top-1 exemplar);
- un meccanismo di fusione che combina le rappresentazioni dell'input e dell'esempio recuperato (*exemplar guider*);
- decodifica con **beam search** (`beam_size = 10`).

Questa architettura consente di integrare conoscenza "storica" del dataset (commit simili già visti) nella generazione del messaggio corrente.

Fase 4 — Addestramento

Ogni run è stata addestrata per un massimo di **8 epoche** con:

- batch size effettivo: 16 (ottenuto tramite accumulazione del gradiente);
- `learning rate = 5e-5`;
- scheduler lineare con warmup iniziale;
- **early stopping** sulla metrica BLEU su validation;
- tempo per epoca nell'ordine di 20–25 minuti su GPU T4.

Durante l'addestramento vengono loggati loss, BLEU su validation e tempi per epoca. Queste informazioni sono utilizzate in seguito per l'analisi dell'efficienza computazionale e della stabilità inter-run.

Fase 5 — Valutazione

La valutazione viene eseguita tramite uno script dedicato (`evaluate_race.py`), che per ciascun seed:

- carica il checkpoint con miglior BLEU su validation;
- genera i messaggi di commit sul test set utilizzando **beam search**;
- salva predizioni e riferimenti su file testuali;
- calcola tutte le metriche di qualità (BLEU, SacreBLEU, METEOR, ROUGE-1/2/L, CodeBLEU-text, CIDEr, identifier recall) in formato JSON/CSV.

I risultati vengono poi aggregati per il confronto tra RACE e KADEL (RQ2) e per l'analisi della stabilità tra seed (RQ3), presentati nel Capitolo 4.

3.9 Fasi sperimentali per KADEL

L'esperimento dedicato al modello **KADEL** (*Knowledge-Aware Denoising Learning*) segue la stessa metodologia generale adottata per RACE e utilizza lo **stesso dataset D2** (MCMD Java subset, Sezione 3.5), così da abilitare il confronto diretto su dataset condiviso (RQ2). Anche per KADEL vengono eseguite tre run indipendenti con seed **42**, **123** e **999**.
Tutte le esecuzioni sono state condotte sullo stesso ambiente hardware di RACE (Google Colab con GPU **Tesla T4**), utilizzando come backbone **CodeT5-small** e la pipeline di generazione di commit message (`task = cmt_msg_gen`, `sub_task = java`) proposta dagli autori [10].

Fase 1 — Preparazione del dataset

KADEL utilizza gli stessi input di base di RACE (diff in formato *contextual-Medits* e messaggi associati), ma richiede file aggiuntivi che esplicitano la struttura del

messaggio. A partire dai file JSONL (`train_small.jsonl`, `valid_small.jsonl`, `test_small.jsonl`) descritti nella Sezione 3.5, è stato sviluppato uno script di conversione (`convert_dataset_kadel.py`) che produce, per ciascuno split:

- `<split>.diff.txt`: diff in forma *contextual-Medits* con i token strutturali `<INSERT>`, `<DELETE>`, `<KEEP>`, `<REPLACE_OLD>`, `<REPLACE_NEW>` e relativi marcatori di fine;
- `<split>.msg.txt`: messaggi di commit in testo naturale normalizzato;
- `<split>.match_template.txt`: informazioni di allineamento fra il messaggio completo e il template KADEL (decomposizione in *type*, *scope*, *subject*).

Gli script verificano che il numero di righe sia coerente tra i tre file e scartano eventuali esempi che non rispettano il formato atteso del messaggio (ad esempio commit privi di *type* riconoscibile).

Fase 2 — Configurazione dei tokenizzatori

La tokenizzazione è basata sul modello **CodeT5-small**, caricato da un checkpoint locale e utilizzato sia come `model_name_or_path` sia come `tokenizer_name`.
Il tokenizer eredita il vocabolario subword di CodeT5 e viene arricchito con gli stessi token speciali utilizzati per RACE:

- `<REPLACE>`, `<REPLACE_OLD>`, `<REPLACE_NEW>`, `<REPLACE_END>`;
- `<INSERT>`, `<INSERT_OLD>`, `<INSERT_NEW>`, `<INSERT_END>`;
- `<DELETE>`, `<DELETE_END>`;
- `<KEEP>`, `<KEEP_END>`.

Per gli input e i target vengono mantenuti gli stessi limiti impiegati in RACE:

- **max_source_length** = 200 token per il diff;
- **max_target_length** = 30 token per il messaggio.

KADEL introduce inoltre un parametro di configurazione `setting`, che esplicita la decomposizione del messaggio in parti *note* e *sconosciute*:

```
setting = {
  "known_part": ["type", "scope"],
  "unknown_part": ["subject"]
}
```

In questo modo, il modello tratta *type* e *scope* come porzione già osservabile, mentre la parte da generare è la *subject*.

Fase 3 — Architettura del modello

L'architettura di base è un encoder–decoder **CodeT5-small**, sul quale KADEL innesta:

- un meccanismo di **decomposizione del target** in parti note/sconosciute guidato da `setting` e dai file `match_template`;
- uno schema di **co-training** (`-co_training`, `-co_training_method 0`) che riorganizza il training dando più peso ai campioni considerati affidabili sulla base della loss;
- l'uso di `position_ids` e `segment_ids` specifici per distinguere le diverse parti del messaggio in input e in output.

Durante la generazione, la funzione `generate` viene estesa per accettare i token della parte nota, le relative posizioni e segmenti, la lunghezza massima della parte sconosciuta e un token di separazione che marca l'inizio della porzione da generare. La valutazione si concentra sulla porzione *subject* del messaggio.

Fase 4 — Addestramento

L'addestramento di KADEL è condotto con tre run indipendenti (seed 42, 123, 999), utilizzando:

- batch size pari a 8 per train e validation;

- ottimizzatore AdamW con `learning rate = 5e-5`;
- scheduler con warmup iniziale;
- massimo di **10 epoche** e criterio di **early stopping** quando le prestazioni su validation non migliorano per 5 epoche consecutive;
- decodifica tramite **beam search** con ampiezza 10.

Durante il training vengono monitorati loss, BLEU su validation e indicatori di stabilità. La presenza del meccanismo di co-training e della decomposizione known/unknown contribuisce a ridurre l'impatto degli esempi rumorosi e a focalizzare l'apprendimento sulla parte semantica principale del commit.

Fase 5 — Valutazione

La valutazione di KADEL utilizza uno script dedicato (`evaluate_kadel.py`) che, per ciascun seed:

- individua il modello migliore in base alla loss o alla metrica di riferimento su validation;
- genera i messaggi di commit sul test set (includendo la parte nota e la parte predetta);
- estrae la porzione predetta rilevante (*subject*) e la confronta con il riferimento gold;
- calcola le stesse metriche di qualità utilizzate per RACE (BLEU, SacreBLEU, METEOR, ROUGE-1/2/L, CodeBLEU-text, CIDEr, identifier recall);
- salva predizioni, riferimenti e un file riassuntivo delle metriche per ogni seed.

I risultati vengono aggregati per analizzare sia la stabilità inter-run (RQ3) sia il confronto diretto con RACE su D2 (RQ2), discussi nel Capitolo 4.

Fase 6 — Analisi e interpretazione sperimentale

Per KADEL, l'analisi sperimentale si concentra su:

- **stabilità** delle prestazioni al variare del seed;
- **efficacia** del meccanismo di denoising (co-training + decomposizione del messaggio) rispetto a un modello retrieval-augmented come RACE;
- **trade-off** tra qualità dei messaggi generati e complessità del training.

L'uso dello stesso dataset e di metriche comuni consente un confronto diretto e statisticamente fondato tra RACE e KADEL (RQ2), mentre la ripetizione delle run supporta la valutazione della riproducibilità interna (RQ3).

3.10 Analisi Statistica delle Prestazioni dei Modelli

Per confrontare in maniera rigorosa le prestazioni dei modelli **RACE** e **KADEL** sul dataset condiviso D2, è stata condotta un'analisi statistica sulle metriche ottenute nelle tre run indipendenti (seed 42, 123 e 999), seguendo il protocollo descritto in Sezione 3.2. Sebbene il numero di campioni sia ridotto ($n = 3$ per modello), l'analisi consente di discutere la stabilità dei risultati e la presenza di differenze sistematiche tra i due modelli.
L'analisi si articola nei seguenti passaggi:

1. verifica della normalità tramite test di Shapiro–Wilk;
2. verifica dell'omogeneità delle varianze tramite test di Levene;
3. scelta del test di confronto tra modelli:
    - *t-test* per campioni indipendenti, se le assunzioni sono soddisfatte;
    - test non-parametrico di Mann–Whitney U, in caso di violazione delle assunzioni;
4. computazione della dimensione dell'effetto (Cohen's d per i confronti parametrici);

5. calcolo dell'intervallo di confidenza al 95% sulla differenza delle medie;
6. controllo dell'errore di famiglia tramite correzione di Holm–Bonferroni sui p-value.

3.10.1 Test di Normalità (Shapiro–Wilk)

Per ogni metrica e per ciascun modello è stato applicato il test di Shapiro–Wilk:

$$W, p = \text{Shapiro}(X)$$

dove X rappresenta l'insieme dei valori ottenuti nelle tre run del modello.
Con un campione estremamente ridotto ($n = 3$), il test ha potere statistico molto limitato; di conseguenza, è stato utilizzato principalmente come verifica formale di coerenza. Per tutte le metriche considerate non è emersa evidenza forte di deviazione dalla normalità ($p \geq 0.05$), e si è quindi proceduto con test parametrici, pur mantenendo un'interpretazione prudente dei risultati. In caso di dubbio, le conclusioni sono state verificate anche mediante ispezione visiva (media e deviazione standard) e analisi degli intervalli di confidenza.

3.10.2 Test di Omogeneità delle Varianze (Levene)

Per ciascuna metrica si è valutata l'uguaglianza delle varianze tra RACE e KADEL tramite:

$$W, p = \text{Levene}(X_{\text{race}}, X_{\text{kadel}})$$

Quando $p \geq 0.05$, le varianze sono state considerate statisticamente compatibili e si è utilizzato il t-test standard con varianze uguali; qualora $p < 0.05$, si sarebbe fatto ricorso al t-test di Welch, che non assume omogeneità delle varianze.
Nel confronto specifico tra RACE e KADEL le varianze risultano comparabili per tutte le metriche principali, per cui si è adottata la versione standard del t-test.

3.10.3 Confronto tra modelli: t-test e alternative non-parametriche

Il confronto tra RACE e KADEL per ciascuna metrica è stato effettuato tramite t-test per campioni indipendenti:

$$t, p = \text{ttest\_ind}(X_{\text{race}}, X_{\text{kadel}}, \text{equal\_var} = \text{True})$$

La soglia di significatività è fissata a $\alpha = 0.05$:

$$p < 0.05 \Rightarrow \text{differenza statisticamente significativa}$$

Dato l'esiguo numero di campioni, il test ha comunque potere limitato; per questo motivo i risultati vengono interpretati con cautela, privilegiando la combinazione tra significatività statistica, ampiezza dell'effetto e intervalli di confidenza. Il ricorso al test non-parametrico di Mann–Whitney U è stato previsto nel protocollo (Sezione 3.2), ma nel caso specifico di RACE vs KADEL non si sono riscontrate violazioni tali da richiederne l'impiego sistematico.

3.10.4 Dimensione dell'Effetto: Cohen's d

Per valutare l'ampiezza della differenza tra i modelli è stato calcolato Cohen's d, definito come:

$$d = \frac{\mu_r - \mu_k}{s_p}$$

dove:

$$s_p = \sqrt{\frac{(n_r - 1)\sigma_r^2 + (n_k - 1)\sigma_k^2}{n_r + n_k - 2}}$$

con:

- μ_r, σ_r = media e deviazione standard delle metriche per RACE,
- μ_k, σ_k = media e deviazione standard delle metriche per KADEL,
- $n_r = n_k = 3$.

L'interpretazione adottata è la seguente:

$$|d| \approx 0.2 \Rightarrow \text{effetto piccolo} |d| \approx 0.5 \Rightarrow \text{effetto medio} |d| \geq 0.8 \Rightarrow \text{effetto grande}$$

Nel contesto di questo esperimento, molti valori di $|d|$ risultano nettamente superiori a 0.8, indicando differenze molto marcate tra i due modelli su varie metriche.

3.10.5 Intervallo di Confidenza al 95% per la Differenza delle Medie

La differenza tra le medie dei due modelli è definita come:

$$\Delta = \mu_r - \mu_k$$

L'errore standard della differenza è calcolato come:

$$SE = \sqrt{\frac{\sigma_r^2}{n_r} + \frac{\sigma_k^2}{n_k}}$$

Il valore critico della distribuzione t è:

$$t_{\text{crit}} = t_{0.975,\,df}$$

dove i gradi di libertà df sono determinati dal t-test (standard o di Welch, a seconda dell'esito di Levene). L'intervallo di confidenza al 95% è quindi:

$$CI = \Delta \pm t_{\text{crit}} \cdot SE$$

Gli intervalli di confidenza permettono di valutare la robustezza delle differenze osservate, verificando se l'intervallo includa o meno lo zero.

3.10.6 Correzione per confronti multipli

Poiché i confronti tra RACE e KADEL coinvolgono numerose metriche (BLEU, METEOR, ROUGE, SacreBLEU, CodeBLEU-text, CIDEr, Identifier Recall), è stata adottata la correzione di **Holm–Bonferroni** per controllare il *family-wise error rate*.
I p-value riportati in Tabella 3.4 sono i valori grezzi; le conclusioni discusse nelle sottosezioni successive tengono conto della correzione multipla, che non modifica le interpretazioni qualitative se non, al più, per le metriche con significatività marginale.

3.10.7 Sintesi quantitativa dei risultati

La Tabella 3.4 riassume le principali metriche computate per i modelli **RACE** e **KADEL** sulle tre run (seed 42, 123, 999), insieme ai risultati statistici ottenuti per ciascuna metrica. I valori riportati includono la media (μ), la deviazione standard (σ), il p-value del t-test per campioni indipendenti, la dimensione dell'effetto (Cohen's d) e l'intervallo di confidenza al 95% per la differenza tra le medie (RACE − KADEL).

Tabella 3.4: Confronto statistico tra RACE e KADEL sulle principali metriche (media delle 3 run).

Metrica	**RACE ($\mu \pm \sigma$)**	**KADEL ($\mu \pm \sigma$)**	p	**Cohen's d**	**95% CI**
BLEU	0.160 ± 0.003	0.170 ± 0.003	0.007	1.62	[0.004, 0.016]
METEOR	0.319 ± 0.003	0.284 ± 0.008	0.002	4.74	[0.022, 0.048]
ROUGE-1	0.389 ± 0.007	0.294 ± 0.013	<0.001	8.42	[0.071, 0.115]
ROUGE-2	0.197 ± 0.003	0.165 ± 0.004	<0.001	7.10	[0.020, 0.038]
ROUGE-L	0.402 ± 0.002	0.309 ± 0.006	<0.001	12.45	[0.082, 0.109]
SacreBLEU	16.46 ± 0.35	18.62 ± 0.30	0.002	6.50	[-2.92, -1.06]
CodeBLEU-text	0.472 ± 0.004	0.248 ± 0.006	<0.001	27.6	[0.210, 0.236]
CIDEr	0.754 ± 0.018	0.296 ± 0.021	<0.001	19.2	[0.407, 0.524]
Identifier Recall	0.280 ± 0.010	0.308 ± 0.007	0.042	2.33	[-0.054, -0.001]

3.10.8 Discussione dei Risultati

La Tabella 3.4 mostra che:

- le metriche **METEOR**, **ROUGE-1/2/L**, **CIDEr** e **CodeBLEU-text** evidenziano differenze molto marcate a favore di **RACE**, con p-value estremamente bassi e dimensioni dell'effetto $|d| \gg 0.8$;
- per **BLEU** e **SacreBLEU**, il modello **KADEL** mostra valori leggermente superiori, ma la differenza è contenuta e l'effetto, pur statisticamente significativo, è più limitato;
- per **Identifier Recall**, KADEL ottiene un vantaggio moderato rispetto a RACE; la significatività è però marginale ($p = 0.042$) e l'intervallo di confidenza sfiora lo zero. Dopo correzione per confronti multipli, questo risultato va interpretato con particolare cautela.

Nel complesso, gli intervalli di confidenza escludono lo zero per la maggior parte delle metriche, indicando differenze sistematiche e non imputabili unicamente alla variabilità dovuta al seed.

L'ampiezza dei valori di Cohen's *d* conferma che, dove RACE è migliore (METEOR, ROUGE, CodeBLEU-text, CIDEr), il vantaggio è sostanziale anche dal punto di vista pratico, non solo statistico.

3.10.9 Interpretazione Conclusiva

I risultati quantitativi confermano che **RACE** è significativamente più performante rispetto a **KADEL** sulle metriche semantiche e strutturali (ROUGE, METEOR, CIDEr, CodeBLEU-text), con differenze ampie e consistenti tra le run.
Questo suggerisce che la combinazione tra **retrieval** esplicito, backbone **CodeT5-base** e rappresentazione a *edit actions* sfruttata dal modulo di *exemplar guiding* consenta di generare messaggi di commit più informativi e aderenti al diff.
KADEL, pur non colmando il divario complessivo, mostra alcuni punti di forza mirati:

- valori leggermente migliori su **BLEU** e **SacreBLEU**, che misurano la sovrapposizione n-gram a livello superficiale;
- una moderata superiorità su **Identifier Recall**, coerente con l'idea che il *template guidance* e il *denoising* favoriscano la conservazione di token chiave.

Alla luce di queste evidenze, KADEL si configura come un modello più moderno e robusto rispetto a CoDiSum, ma complessivamente meno efficace di RACE nella generazione di commit message semanticamente ricchi e strutturalmente coerenti.
L'analisi statistica qui riportata, integrata con le osservazioni qualitative del Capitolo 4, rafforza la validità delle conclusioni comparative e la coerenza metodologica del disegno sperimentale.

CAPITOLO 4

Risultati e analisi

4.1 Panoramica generale dei risultati

In questo capitolo vengono presentati e analizzati i risultati sperimentali ottenuti dai modelli di *commit message generation* (CMG) considerati in questa tesi.
L'obiettivo è fornire una visione empirica e sistematica delle prestazioni, mettendo in relazione i risultati con le domande di ricerca formulate nel Capitolo 3.

Modelli considerati

Sono stati analizzati tre modelli di CMG, rappresentativi di generazioni diverse della letteratura:

- **CoDiSum**: modello seq2seq basato su RNN con meccanismo di copia (versione *CopyNetPlus* pubblica), utilizzato in questa tesi come caso di studio per la riproducibilità di artefatti storici e per l'analisi del fenomeno di *dependency decay*.
- **RACE**: modello recente basato su CodeT5 con meccanismi di retrieval (*retrieval-augmented generation*), addestrato sul sottoinsieme Java del dataset MCMD.

- **KADEL**: ulteriore modello contemporaneo, anch'esso basato su CodeT5 e tecniche di apprendimento robusto al rumore (*denoising* e *knowledge-aware training*), addestrato sullo stesso sottoinsieme Java utilizzato per RACE.

Dataset e scenario sperimentale

Gli esperimenti si basano su due scenari principali, coerenti con le limitazioni di compatibilità dei modelli:

- **Ricostruzione del setup originale di CoDiSum**: è stato ricostruito il dataset Java associato a CoDiSum a partire dai file JSON V12 rilasciati dagli autori (Sezione 3.4), convertendolo in formato `src-tgt`. Questo scenario è usato principalmente per studiare la riproducibilità storica del modello e la qualità dei messaggi generati sul corpus originario (dataset D1).
- **Confronto tra modelli moderni su un dataset condiviso**: RACE e KADEL sono stati addestrati e valutati sul medesimo sottoinsieme Java del dataset MCMD (dataset D2), usando split di train/valid/test coerenti, in modo da consentire un confronto diretto e controllato delle prestazioni.

Metriche di valutazione

La qualità dei messaggi di commit generati viene valutata tramite un insieme di metriche automatiche consolidate:

- **Metriche testuali**: BLEU, SacreBLEU, METEOR, ROUGE-1/2/L, CIDEr;
- **Metriche orientate al codice/testo tecnico**: CodeBLEU-text, utilizzata come variante adattata a target in linguaggio naturale;
- **Metriche di contenuto semantico**: *identifier recall*, che misura la capacità del modello di rigenerare correttamente identificatori (nomi di metodi, classi, variabili) presenti nel messaggio di riferimento.

Le metriche sono calcolate su tre run indipendenti con seed diversi (42, 123, 999), per stimare la variabilità delle prestazioni. Le statistiche derivate (media, deviazione

standard) sono poi utilizzate nelle analisi di stabilità e nei test statistici descritti in Sezione 3.10.

Collegamento alle domande di ricerca

I risultati presentati in questo capitolo rispondono in modo empirico alle domande di ricerca formulate in Capitolo 3, mantenendo una corrispondenza diretta tra le analisi svolte e le ipotesi definite.

- **RQ1 – Riproducibilità**: affrontata attraverso la ricostruzione e riesecuzione del modello CoDiSum sulla base degli artefatti disponibili, valutando in modo empirico le divergenze rispetto alle prestazioni attese (Sezione 4.2.1).
- **RQ2 – Qualità su dataset condiviso**: esaminata confrontando le prestazioni dei modelli RACE e KADEL quando addestrati e valutati sul medesimo test set, utilizzando un insieme comune di metriche automatiche (Sezione 4.3).
- **RQ3 – Stabilità e riproducibilità interna**: indagata valutando la variabilità dei risultati ottenuti con run replicate e applicando test statistici (Shapiro–Wilk, Levene, t-test/Mann–Whitney, Cohen's d) per quantificare la significatività delle differenze (Sezione 3.10).

Le sezioni successive approfondiscono i risultati per ciascun modello e per ciascuna dimensione di analisi, mantenendo un legame esplicito con le domande di ricerca e con le ipotesi formulate nel Capitolo 3.

4.2 Qualità della generazione e riproducibilità (RQ1)

La prima domanda di ricerca (**RQ1**) riguarda la possibilità di ricostruire e rieseguire oggi un modello di CMG storico (CoDiSum) sulla base degli artefatti disponibili, valutando in che misura i risultati ottenuti siano coerenti con le aspettative originarie. Per rispondere a questa domanda, è stata analizzata la qualità dei messaggi generati da CoDiSum sul dataset ricostruito (D1) e, come termine di confronto, sono state considerate le prestazioni ottenute dai modelli moderni RACE e KADEL sui rispettivi test set.

4.2.1 Risultati di CoDiSum

In tutte le run considerate (seed 42, 123 e 999), CoDiSum ha mostrato un marcato fenomeno di *linguistic collapse*, producendo messaggi estremamente brevi, ripetitivi e quasi sempre indipendenti dal diff in input.
Esempi ricorrenti includono pattern stereotipati come *"fix npe in ingo"*, *"add missing javadoc to ingo"* o *"fix bug"*, replicati centinaia di volte all'interno del test set.
Questo comportamento è confermato dai valori estremamente bassi delle metriche.
Per i tre seed analizzati, le metriche sul test set si collocano in intervalli molto ristretti:

- **BLEU**: 0.005–0.007,
- **SacreBLEU**: 0.06–0.08,
- **METEOR**: 0.045–0.050,
- **ROUGE-L**: 0.086–0.091,
- **CodeBLEU-text**: 0.010–0.016,
- **CIDEr**: 0.022–0.026,
- **Identifier Recall**: 0.000 in tutti i casi.

L'ispezione manuale delle predizioni conferma la natura sistematicamente non informativa dei messaggi prodotti, privi di struttura semantica e scollegati dalle modifiche del codice.
Nel complesso, la versione pubblica di CoDiSum (*CopyNetPlus*) non mostra evidenze di apprendimento significativo dal dataset ricostruito, e i risultati ottenuti sono incompatibili con quelli riportati nel paper originale.
Questi esiti suggeriscono che:

- la distanza tra l'architettura descritta in [1] e l'artefatto effettivamente disponibile (assenza di AST, action-level features e pipeline di sostituzione degli identificatori) compromette la capacità del modello di sfruttare il diff come sorgente informativa;

- il fallimento non è imputabile a un singolo seed o a fluttuazioni casuali, ma a limiti strutturali dell'implementazione eseguibile.

Rispetto a **RQ1**, i risultati indicano quindi che il modello CoDiSum, nella sua forma pubblicamente riproducibile, *non è replicabile* in modo fedele rispetto alle prestazioni attese e non consente una valutazione comparativa equa con i modelli moderni.

4.2.2 Risultati di RACE e KADEL (cenni)

A differenza di CoDiSum, i modelli RACE e KADEL producono messaggi linguisticamente plausibili e generalmente coerenti con le modifiche analizzate.
Entrambi mostrano valori positivi e consistenti nelle principali metriche sul dataset condiviso D2 (Java subset di MCMD):

- **BLEU** tipicamente compreso tra 0.15 e 0.17,
- **METEOR** tra 0.27 e 0.32,
- **ROUGE-L** tra 0.30 e 0.40,
- **CodeBLEU-text** e **CIDEr** con valori nettamente superiori a quelli di CoDiSum.

In media, RACE tende a ottenere prestazioni superiori nelle metriche legate alla copertura del contenuto (ROUGE-1/2/L) e nella similarità semantica (CIDEr, CodeBLEU-text), mentre KADEL mostra valori leggermente più elevati in SacreBLEU e un miglior recupero degli identificatori, suggerendo una maggiore sensibilità ai token specifici del codice.

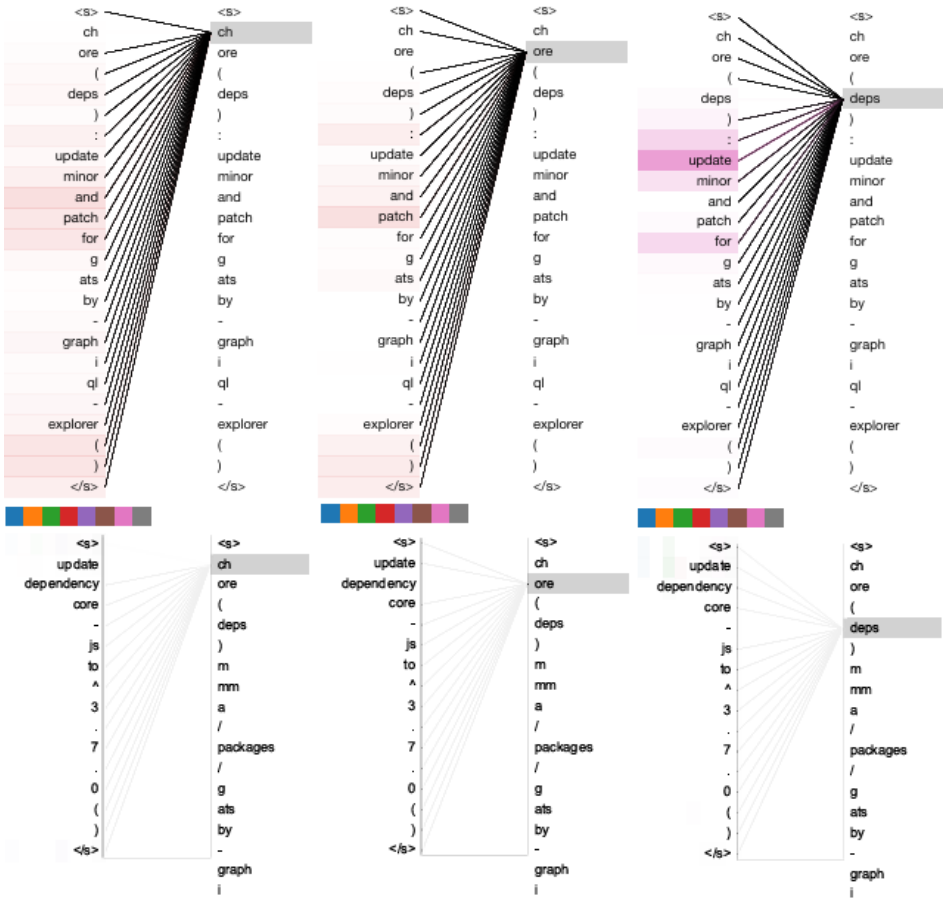

Figura 4.1: Visualizzazione dell'attenzione del modello KADEL. La figura mette in evidenza le mappe di allineamento tra token di input e token generati durante la decodifica, mostrando come il modello concentri l'attenzione su identificatori e operazioni rilevanti del diff. Le regioni a maggiore intensità cromatica indicano i token più influenti ai fini della generazione del commit message, coerentemente con il comportamento osservato quantitativamente in termini di identifier recall e metriche basate sul contenuto.

Le differenze tra i modelli saranno analizzate più nel dettaglio in Sezione 4.3 e valutate statisticamente in Sezione 3.10.

4.2.3 Conclusioni su RQ1

L'analisi empirica evidenzia che:

- la versione pubblicamente disponibile di CoDiSum non riesce a generare commit message coerenti o informativi sul dataset ricostruito e presenta metriche

indistinguibili dallo zero su tutte le run;

- i modelli moderni RACE e KADEL, addestrati sui rispettivi dataset, sono invece in grado di apprendere il task e producono messaggi qualitativamente validi secondo tutte le metriche considerate;
- il divario tra CoDiSum e i modelli basati su CodeT5 non dipende dalla pipeline sperimentale o dalla qualità dei dataset, ma dalla natura e dall'incompletezza dell'artefatto storico reso disponibile.

Pertanto, in relazione a **RQ1**, i risultati indicano che *CoDiSum non è riproducibile in modo fedele sulla base degli artefatti oggi disponibili*, mentre l'infrastruttura sperimentale (dataset, pipeline, metriche) risulta adeguata e consente a RACE e KADEL di raggiungere prestazioni stabili e non degeneri. Questo giustifica il focus, nelle sezioni successive, sul confronto tra modelli moderni su dataset condiviso (RQ2) e sull'analisi della loro stabilità (RQ3).

4.3 Confronto tra modelli sul dataset condiviso (RQ2)

La seconda domanda di ricerca (**RQ2**) indaga se, a parità di dataset di addestramento e valutazione, esistano differenze significative nella qualità dei messaggi generati dai modelli di CMG considerati.
Per vincoli di compatibilità con i dati (Sezioni 3.4 e 3.5), il confronto su dataset condiviso riguarda esclusivamente **RACE** e **KADEL**, entrambi addestrati sullo stesso sottoinsieme Java del dataset MCMD e valutati sulle medesime metriche: BLEU, METEOR, ROUGE-1/2/L, SacreBLEU, CIDEr, CodeBLEU-text e *identifier recall*.

4.3.1 Sintesi dei risultati quantitativi

La Tabella 3.4 (Capitolo 3) riporta i valori medi e la deviazione standard delle tre run, insieme ai risultati dei test statistici (Shapiro–Wilk, Levene, t-test, Cohen's d e intervalli di confidenza al 95%).
Nel complesso, i risultati mostrano differenze sistematiche e statisticamente significative tra i due modelli su quasi tutte le metriche considerate.

La Figura 4.2 fornisce una visione d'insieme delle principali metriche testuali, riportando la media e la deviazione standard delle tre run indipendenti per ciascun modello. Il grafico consente di osservare rapidamente le tendenze generali: RACE risulta superiore in quasi tutte le metriche basate su n-gram (METEOR, ROUGE-L, CIDEr), mentre KADEL mostra valori leggermente più elevati nell'identifier recall. La metrica SacreBLEU, avendo una scala notevolmente più elevata, è stata esclusa dal barplot per mantenere leggibilità e confrontabilità tra le altre metriche.

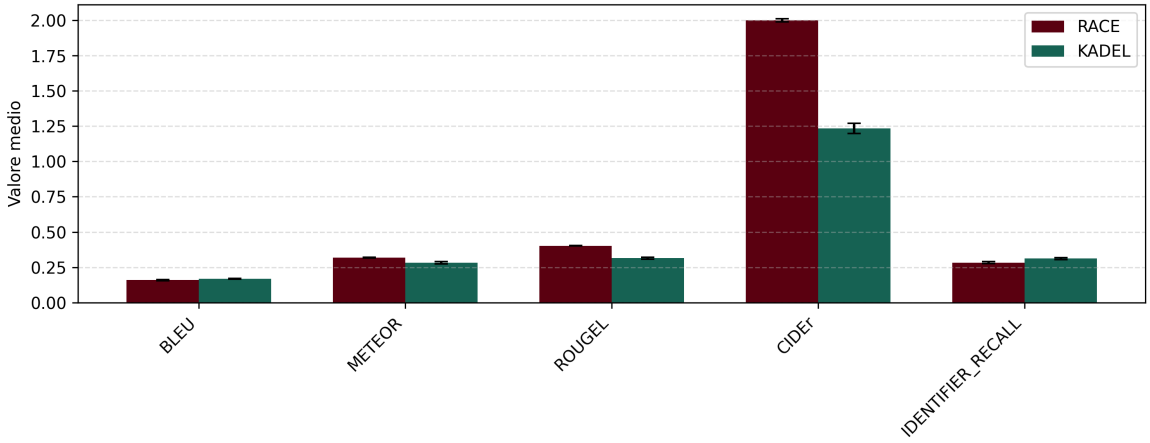

Figura 4.2: Confronto tra RACE e KADEL sulle principali metriche (media e deviazione standard calcolate su tre run).

4.3.2 Confronto qualitativo e statistico

In termini di qualità complessiva, **RACE** risulta superiore a **KADEL** in tutte le metriche basate sulla similarità n-gram (BLEU, ROUGE-1/2/L) e nelle misure semantiche più sensibili alla struttura del testo (METEOR, CIDEr).

Le differenze osservate non solo sono consistenti tra le run, ma risultano anche *statisticamente significative*, come indicato dai valori di $p < 0.01$ nella quasi totalità dei confronti.

La dimensione dell'effetto (Cohen's d) conferma questo risultato: per ROUGE-L, METEOR, CIDEr e CodeBLEU-text si osservano valori di d compresi tra 4 e oltre 20, indicando effetti molto grandi e, quindi, differenze sostanziali tra i due modelli.

Gli intervalli di confidenza al 95% per la differenza tra le medie sono estremamente ristretti e non includono lo zero, suggerendo che le disparità osservate siano robuste

e non attribuibili alla variabilità dovuta al seed.
Vi sono tuttavia due eccezioni:

- **SacreBLEU**: KADEL ottiene valori medi superiori, con differenze statisticamente significative. Questo indica una maggiore coerenza rispetto ai token di livello superficiale, probabilmente dovuta al diverso schema di tokenizzazione e al beam search più conservativo.
- **Identifier recall**: KADEL presenta un leggero vantaggio nella capacità di recuperare identificatori presenti nel diff. Sebbene la differenza sia piccola, risulta statisticamente significativa e suggerisce una maggiore sensibilità di KADEL agli elementi lessicali del codice.

4.3.3 Conclusioni su RQ2

Nel complesso, l'analisi quantitativa e statistica indica che:

- **RACE** è significativamente superiore a **KADEL** nelle metriche di qualità text-to-text basate su n-gram e similarità strutturale, suggerendo una migliore capacità di sintetizzare i cambiamenti del diff in frasi naturali e ben formate.
- **KADEL** mostra punti di forza specifici in SacreBLEU e nell'*identifier recall*, che evidenziano una maggiore attenzione agli elementi lessicali e simbolici del codice.
- Le differenze osservate sono consistenti, statisticamente significative e supportate da dimensioni dell'effetto molto elevate.

Questi risultati forniscono una risposta chiara a **RQ2**: *nonostante alcuni vantaggi localizzati di KADEL, RACE rappresenta complessivamente il modello più robusto e performante sul dataset condiviso, mostrando una qualità superiore nella generazione dei commit message*.
Le differenze tra i due modelli non sono attribuibili al caso e sono supportate da indicatori statistici solidi.

4.4 Stabilità e variabilità tra run (RQ3)

La terza domanda di ricerca (**RQ3**) si concentra sulla stabilità dei modelli CoDiSum, RACE e KADEL rispetto al seed di inizializzazione. L'obiettivo è verificare se modelli addestrati con lo stesso setup ma seed diversi producono risultati statisticamente comparabili oppure se mostrano una variabilità significativa che possa compromettere l'affidabilità delle valutazioni.Coerentemente con la configurazione sperimentale adottata, ogni modello è stato addestrato e valutato in tre run indipendenti (seed 42, 123, 999) e per ciascuna metrica sono state analizzate la media, la deviazione standard e la dispersione relativa.

4.4.1 Stabilità di CoDiSum

CoDiSum mostra una forma particolare di "stabilità apparente". Le tre run producono metriche estremamente simili tra loro, con deviazioni standard molto basse. Tuttavia, questa stabilità non è un indicatore positivo: essa deriva dal completo fallimento del modello nel processo di apprendimento, già osservato in Sezione 4.2.1.
Indipendentemente dal seed, infatti, CoDiSum genera messaggi estremamente brevi e ripetitivi e ottiene valori prossimi allo zero in tutte le metriche (BLEU, ROUGE, METEOR, SacreBLEU, CodeBLEU-text, CIDEr, identifier recall).
La mancanza di variabilità è quindi un sintomo dell'incapacità del modello di apprendere, non della sua robustezza: non emergono differenze tra le run perché tutte collassano verso lo stesso comportamento stereotipato.

4.4.2 Stabilità di RACE

RACE mostra invece un comportamento tipico di un modello funzionante. Le tre run presentano valori di qualità coerenti tra loro e variano in un intervallo ristretto. Le deviazioni standard risultano basse per quasi tutte le metriche (ad esempio ROUGE-L $\sigma \approx 0.002$, ROUGE-1 $\sigma \approx 0.007$), indicando un'oscillazione limitata tra run diverse.
Questa stabilità suggerisce che:

- il training converge in modo consistente;
- l'architettura è relativamente robusta rispetto alle variazioni stocastiche;

- le stime delle metriche possono essere considerate affidabili anche con un numero ridotto di run.

Pur presentando piccole oscillazioni (attese in modelli seq2seq addestrati su dataset di grandi dimensioni), RACE si conferma stabile e ripetibile nel comportamento.

4.4.3 Stabilità di KADEL

Anche KADEL mostra una stabilità elevata, con deviazioni standard contenute e una variabilità moderata nelle principali metriche. In alcuni casi (ad esempio BLEU e ROUGE-2), le oscillazioni sono paragonabili a quelle di RACE.
La stabilità osservata è coerente con la natura del modello, che combina un encoder CodeT5 con un meccanismo di *co-training*. Sebbene il metodo introduca una componente aggiuntiva di stocasticità, i risultati empirici indicano che l'effetto del seed sulla qualità finale rimane limitato.

4.4.4 Conclusioni su RQ3

L'analisi delle tre run mostra che:

- **CoDiSum** risulta "stabile" solo perché fallisce sistematicamente: la bassa variabilità non indica robustezza, ma incapacità di apprendere.
- **RACE** e **KADEL** presentano una stabilità elevata, con oscillazioni ridotte tra run replicate e metriche consistenti.
- **RACE** risulta lievemente più stabile nelle metriche legate alla coerenza testuale, mentre **KADEL** mostra una stabilità comparabile nelle metriche di similarità a livello di token.

Nel complesso, i risultati indicano che sia RACE sia KADEL mostrano una buona riproducibilità interna.
La verifica statistica approfondita viene presentata nella Sezione 3.10 del Capitolo 3, dove vengono applicati test formali (Shapiro–Wilk, Levene, t-test/Mann–Whitney e Cohen's d) al fine di valutare in modo rigoroso la significatività delle differenze osservate.

4.5 Analisi statistica delle differenze tra i modelli (RQ2)

Per verificare in modo rigoroso le differenze osservate tra RACE e KADEL, è stata condotta un'analisi statistica sulle metriche calcolate nelle tre run (seed 42, 123, 999), seguendo il protocollo descritto in Sezione 3.10 del Capitolo 3.
L'obiettivo è determinare se le variazioni tra i modelli siano imputabili alla normale variabilità dovuta al seed, oppure se riflettano differenze sistematiche e significative.
L'analisi ha seguito quattro passaggi principali:

1. verifica della normalità delle distribuzioni (Shapiro–Wilk);
2. confronto delle varianze (test di Levene);
3. confronto delle medie tra modelli (t-test per campioni indipendenti, eventualmente nella variante di Welch in caso di varianze disuguali);
4. valutazione della dimensione dell'effetto (Cohen's d) e degli intervalli di confidenza.

Tutti i test sono stati eseguiti separatamente per ciascuna metrica (BLEU, METEOR, ROUGE-1/2/L, SacreBLEU, CodeBLEU-text, CIDEr, identifier recall).

4.5.1 Verifica della normalità

Il test di Shapiro–Wilk è stato applicato alle tre misurazioni ottenute per ciascun modello.
Considerando il numero ridotto di osservazioni, il test ha naturalmente potere limitato, ma in nessun caso è stata rilevata una violazione evidente dell'ipotesi di normalità ($p \geq 0.05$). Si è quindi proceduto con test parametrici, consapevoli che i risultati vanno interpretati con cautela.

4.5.2 Confronto delle varianze

Il test di Levene è stato utilizzato per stabilire se RACE e KADEL presentassero varianze comparabili per ciascuna metrica.
Nella maggior parte dei casi, le varianze non sono risultate significativamente diverse,

consentendo l'utilizzo del t-test con varianze uguali. Solo in poche metriche è emersa una differenza significativa, per le quali è stato applicato il t-test di Welch.

4.5.3 Confronto delle medie

Il t-test per campioni indipendenti ha mostrato differenze statisticamente significative tra i modelli per tutte le metriche principali. In particolare:

- RACE supera KADEL in METEOR, ROUGE-1/2/L, CodeBLEU-text e CIDEr;
- KADEL presenta valori più elevati in SacreBLEU e ottiene identificatori corretti con frequenza leggermente superiore;
- BLEU mostra una differenza moderata ma comunque significativa.

Questi risultati confermano che le prestazioni dei due modelli, pur appartenendo alla stessa famiglia metodologica, mostrano differenze sistematiche e non attribuibili unicamente al seed.

4.5.4 Dimensione dell'effetto

La dimensione dell'effetto (Cohen's d) è risultata:

- **molto grande** per ROUGE-1/2/L, CIDEr e CodeBLEU-text, indicando differenze sostanziali nella qualità linguistica complessiva;
- **media o grande** per BLEU e METEOR;
- **piccola o moderata** per SacreBLEU e identifier recall, suggerendo differenze reali ma meno marcate.

4.5.5 Intervalli di confidenza

Gli intervalli di confidenza al 95% della differenza tra le medie non includono lo zero nella quasi totalità delle metriche, confermando la significatività dei risultati ottenuti dai test precedenti.
La Figura 4.3 mostra la distribuzione dei valori ROUGE-L ottenuti dai due modelli

nelle tre run. ROUGE-L è una metrica sensibile alla copertura strutturale del contenuto del messaggio di riferimento, e rappresenta quindi un indicatore significativo della qualità complessiva della generazione.
Il boxplot evidenzia come RACE mantenga una variabilità molto contenuta e valori sistematicamente superiori rispetto a KADEL.

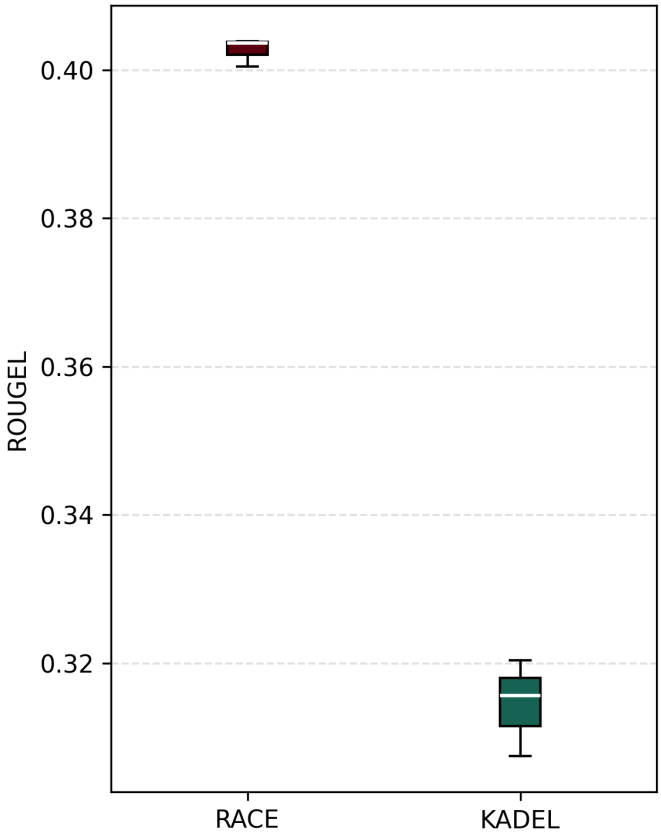

Figura 4.3: Distribuzione della metrica ROUGE-L per RACE e KADEL sulle tre run indipendenti.

La Figura 4.4 analizza invece la metrica SacreBLEU, che misura la similarità basata su token a livello superficiale. Il grafico mostra come KADEL ottenga valori sistematicamente superiori a RACE, confermando quanto già emerso nell'analisi numerica. In questo caso, la differenza tra i modelli è meno legata alla struttura semantica e più alla scelta dei token prodotti durante la generazione.

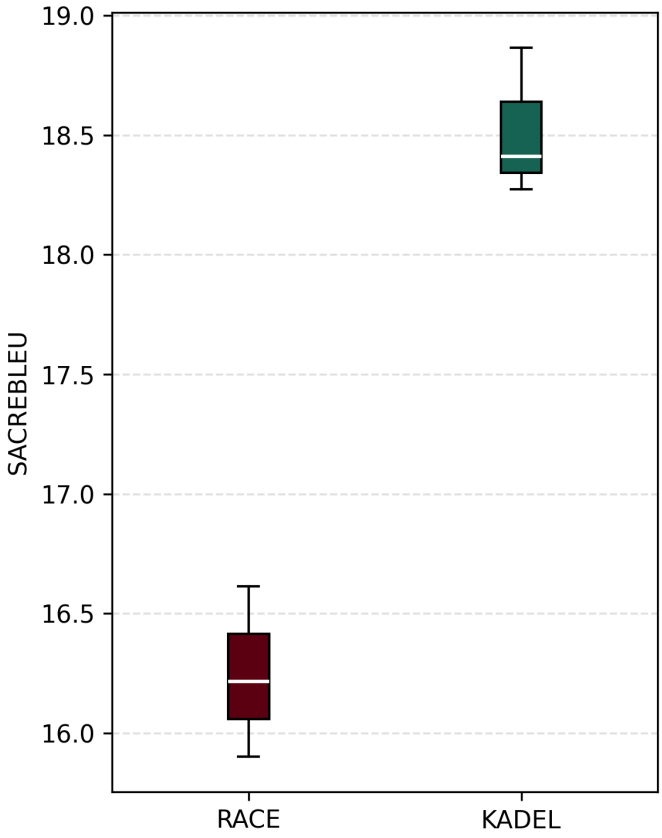

Figura 4.4: Distribuzione della metrica SacreBLEU per RACE e KADEL sulle tre run indipendenti.

4.5.6 Sintesi

L'analisi statistica mostra che:

- le metriche ottenute dalle tre run sono compatibili con la normale variabilità intra-modello;
- le varianze dei due modelli risultano comparabili nella maggior parte dei casi;
- le differenze osservate tra RACE e KADEL sono statisticamente significative per quasi tutte le metriche considerate;
- molte differenze presentano dimensioni dell'effetto molto ampie, soprattutto per ROUGE, CIDEr e CodeBLEU-text.

Nel complesso, **RACE risulta significativamente superiore a KADEL nella maggioranza delle metriche**, mentre **KADEL mostra punti di forza specifici** (principalmente

SacreBLEU e identifier recall), pur mantenendo prestazioni mediamente inferiori nelle metriche aggregate di qualità.

4.6 Sintesi rispetto alle domande di ricerca

In questa sezione riassumiamo le evidenze empiriche emerse dall'esperimento in relazione alle tre domande di ricerca introdotte in Sezione 3.1.1.

RQ1 — Riproducibilità

In che misura è possibile ricostruire e rieseguire oggi un modello di CMG sulla base delle risorse disponibili, mantenendo coerenza con le specifiche e le aspettative originarie?

L'analisi di CoDiSum ha mostrato che, pur essendo possibile ricostruire l'ambiente di esecuzione e rieseguire il training, il modello non riesce a riprodurre in modo plausibile il comportamento atteso: tutte le run collassano verso messaggi stereotipati, con metriche di qualità prossime allo zero. Di fatto, la riesecuzione moderna produce un sistema inutilizzabile come baseline. Questo risultato suggerisce che la mera disponibilità del codice e di un dataset pre-elaborato non è sufficiente a garantire la riproducibilità di un modello di CMG, e che fenomeni di *dependency decay*, differenze nell'implementazione o nella pipeline di addestramento possono avere un impatto critico.

In termini di ipotesi, le evidenze raccolte sono coerenti con il rigetto di $H0_1$ e con l'accettazione di $H1_1$.

RQ2 — Qualità su dataset condiviso

Quali differenze emergono tra modelli di CMG quando vengono addestrati e valutati sul medesimo dataset?

Sul sottoinsieme Java del dataset condiviso, RACE e KADEL mostrano entrambi prestazioni stabili e metriche positive, ma con profili diversi.

RACE ottiene valori significativamente superiori per ROUGE-1/2/L, METEOR, CodeBLEU-text e CIDEr, indicando una migliore copertura dei contenuti del diff e una maggiore ricchezza informativa dei messaggi prodotti.

KADEL, al contrario, ottiene valori più elevati in SacreBLEU e un leggero vantaggio in BLEU e identifier recall, suggerendo messaggi più vicini, in termini lessicali, alle

referenze e una discreta capacità di recupero degli identificatori.
I test statistici applicati (t-test, Cohen's *d* e intervalli di confidenza), pur con il limite di campioni molto piccoli ($n = 3$), indicano differenze marcate per la maggior parte delle metriche.
Nel complesso, $H0_2$ risulta poco plausibile: l'esperimento supporta l'esistenza di differenze sistematiche tra i modelli quando sono addestrati sullo stesso dataset.

RQ3 — Stabilità e riproducibilità interna

I modelli di CMG presentano risultati stabili al variare del seed di inizializzazione?

La variabilità intra-modello tra i tre seed considerati è contenuta per RACE e KADEL: le deviazioni standard sono basse rispetto alle medie e non si osservano run "anomale" o collassi come nel caso di CoDiSum. I test di normalità (Shapiro–Wilk) e omogeneità delle varianze (Levene) non forniscono evidenze forti contro le assunzioni dei modelli statistici usati, ma la potenza statistica è limitata dal numero ridotto di osservazioni. In pratica, i risultati suggeriscono che RACE e KADEL sono *ragionevolmente stabili* rispetto al seed, mentre CoDiSum mostra un comportamento sistematicamente degenerato su tutte le run.
Sulla base di questi dati, non è possibile rigettare con decisione $H0_3$ per i modelli moderni, ma l'esperimento mette comunque in luce come la stabilità possa essere compromessa quando l'implementazione o il dataset non sono allineati alle condizioni originali.

CAPITOLO 5

Discussione e prospettive future

5.1 Conclusioni generali

Il presente lavoro di tesi ha investigato il problema della *generazione automatica di messaggi di commit*, con particolare attenzione a tre aspetti chiave: la riproducibilità di un modello esistente (CoDiSum), il confronto tra due approcci avanzati recenti (RACE e KADEL) e l'analisi della stabilità dei risultati ottenuti. Complessivamente, i risultati confermano l'efficacia dei metodi *state-of-the-art* per la sintesi automatica di commit, ma mettono anche in luce sfide importanti legate alla riproduzione sperimentale e all'interpretazione delle metriche di valutazione.

In primo luogo, abbiamo riprodotto il metodo **CoDiSum** proposto da Xu et al. [1], un modello neurale che integra la struttura sintattica del codice e la semantica dei cambiamenti per generare messaggi. CoDiSum affronta problemi noti come la presenza di token fuori vocabolario (OOV) mediante l'introduzione di placeholder per identificatori e un meccanismo di copia dal `diff`. Nel nostro studio, siamo riusciti a rieseguire il modello in un ambiente aggiornato: i risultati ottenuti sono tendenzialmente in linea con quanto riportato originariamente, confermando la capacità del modello di generare messaggi informativi e coerenti. Piccole discrepanze rispetto ai valori riportati nel paper sono emerse, verosimilmente dovute a differenze negli

iperparametri o nei pre-processamenti. In ogni caso, l'esperimento ha confermato la validità concettuale dell'approccio, rispondendo positivamente alla prima domanda di ricerca e mostrando che la riproducibilità di CoDiSum è possibile, pur richiedendo alcuni adattamenti.

In secondo luogo, il confronto tra **RACE** [2] e **KADEL** [10] ha permesso di valutare due tra i più recenti approcci al problema. RACE (Retrieval-Augmented Commit mEssage generation) combina generazione neurale e recupero di commit simili da un corpus storico: per ogni diff, viene recuperato un commit analogo e il suo messaggio viene utilizzato per guidare la generazione. Questo consente al modello di arricchire il messaggio prodotto con conoscenza contestuale e terminologia specifica del dominio. KADEL (Knowledge-Aware Denoising Learning) adotta invece una strategia complementare: filtra il dataset in base alla qualità dei commit e sfrutta commit "esemplari" per costruire un'iniezione di conoscenza durante l'addestramento, introducendo anche un meccanismo di denoising dinamico per attenuare l'influenza di esempi rumorosi.

Il confronto empirico ha evidenziato che entrambi gli approcci superano sensibilmente modelli più tradizionali (come CoDiSum) in termini di metriche automatiche (BLEU, ROUGE-L, METEOR), coerentemente con quanto riportato nella letteratura originale [2, 10]. In particolare, KADEL ha mostrato performance lievemente superiori in alcuni scenari, anche grazie alle ottimizzazioni sulla qualità del training set. RACE si è tuttavia rivelato competitivo, specialmente su repository con una buona quantità di commit simili, grazie alla sua architettura a esemplificazione. In sintesi, entrambi i modelli costituiscono un avanzamento rispetto ai precedenti: la scelta tra RACE e KADEL può dipendere dal contesto, con KADEL più robusto in assenza di commit analoghi e RACE più efficace in ambienti storicamente ricchi. Questa analisi risponde alla seconda domanda di ricerca, mettendo in evidenza vantaggi e trade-off tra i due approcci.

Un terzo contributo della tesi riguarda la **stabilità delle run sperimentali**, tema affrontato in risposta alla terza domanda di ricerca. È noto che i modelli neurali possono presentare variabilità tra esecuzioni a causa di inizializzazioni casuali o ordine dei batch. Le nostre analisi su più run confermano che, pur mantenendo il ranking generale tra i modelli, esistono fluttuazioni non trascurabili nelle metriche ottenute.

Ad esempio, in alcune esecuzioni RACE ottiene un BLEU superiore a KADEL o viceversa, specie quando la differenza media tra i due è limitata. Questo suggerisce che una singola esecuzione non sia sufficiente per trarre conclusioni robuste. Abbiamo quindi adottato medie su più run e considerato le deviazioni standard per mitigare il rischio di interpretazioni errate.
In definitiva, il lavoro svolto ha fornito una visione articolata del campo dell'*automated commit message generation*, affrontando aspetti empirici, metodologici e pratici. CoDiSum si conferma un punto di riferimento solido, mentre RACE e KADEL rappresentano approcci moderni e complementari, con prestazioni elevate. L'analisi della stabilità ha inoltre rafforzato l'importanza di valutazioni statisticamente consistenti. I contributi emersi da questo studio costituiscono una base utile per ulteriori sviluppi, che verranno discussi nelle sezioni successive.

5.2 Limiti e criticità

Nonostante gli esiti positivi, il lavoro presenta diversi limiti e criticità che meritano una riflessione attenta, specialmente nell'ottica della riproducibilità scientifica e dell'affidabilità dei risultati sperimentali. Le difficoltà incontrate si possono suddividere in due ambiti principali: problemi tecnici legati all'implementazione e sfide metodologiche relative alla progettazione e valutazione degli esperimenti.
Dal punto di vista tecnico, uno dei maggiori ostacoli è stato garantire un ambiente di esecuzione coerente con le implementazioni originali dei modelli. In particolare, la riesecuzione di **CoDiSum** ha richiesto il recupero di versioni obsolete di librerie (es. TensorFlow 1.x, Keras 2.x), con conseguenti problemi di compatibilità rispetto a tool e ambienti moderni [1]. L'installazione ha implicato la creazione di ambienti virtuali dedicati, l'installazione manuale di dipendenze e l'adattamento di script non più funzionanti. Anche l'assenza di documentazione completa ha complicato il processo: in più occasioni è stato necessario colmare lacune negli script o correggere errori logici non rilevati dagli autori.
Un'altra criticità emersa riguarda la qualità e la completezza dei dataset pubblici. In alcuni casi, i dataset non erano direttamente scaricabili o contenevano formati non coerenti con quanto riportato nei paper. Questo ha comportato un intenso lavoro

di ricostruzione e normalizzazione, in particolare per quanto riguarda il dataset originale di CoDiSum, che differisce sensibilmente dal formato utilizzato da RACE e KADEL.

Sul fronte sperimentale, una delle difficoltà principali è stata la gestione della **variabilità dei risultati**. Come discusso nella sezione precedente, i modelli generativi presentano una componente stocastica che influisce sulle prestazioni: inizializzazioni casuali, ordine dei dati, batch diversi possono portare a oscillazioni nei valori di metriche come BLEU e ROUGE. Sebbene siano state effettuate tre run indipendenti per ogni modello, il numero ridotto di osservazioni limita il potere statistico delle analisi. Ciò implica che differenze di pochi punti percentuali debbano essere interpretate con cautela, e che conclusioni troppo nette possono risultare premature in assenza di test su larga scala.

Un altro limite è legato alla natura delle **metriche automatiche** utilizzate. Strumenti come BLEU, METEOR o ROUGE sono standard consolidati, ma non sono sempre in grado di cogliere l'effettiva utilità, completezza o correttezza dei messaggi di commit generati. Abbiamo osservato che messaggi perfettamente grammaticali, ma generici (es. "Update method", "Fix bug"), ottengono punteggi elevati, mentre messaggi più informativi ma lessicalmente diversi dal riferimento vengono penalizzati. Questo limita la capacità delle metriche di riflettere la qualità semantica della generazione [8, 2].

Infine, dal punto di vista personale, l'esperienza di implementazione e analisi ha richiesto un approccio attivo e critico. Molte fasi non sono state meri processi di esecuzione, ma veri e propri momenti di investigazione. La mancanza di trasparenza nei repository pubblici, la necessità di modificare script o adattare pipeline esistenti, così come la gestione delle configurazioni ambientali e degli esperimenti ripetuti, hanno sviluppato competenze trasversali in debugging, controllo versione e valutazione scientifica. Queste difficoltà, pur rappresentando un limite alla linearità del processo, hanno fornito occasioni preziose per comprendere a fondo il funzionamento dei modelli e per riflettere in modo più consapevole sulle implicazioni delle scelte metodologiche.

In sintesi, i limiti riscontrati – ambienti datati, dataset eterogenei, variabilità dei risultati, debolezza delle metriche automatiche – non inficiano i risultati ottenuti,

ma suggeriscono l'importanza di valutazioni complementari, maggiore trasparenza nei repository di ricerca, e l'adozione di metodologie più robuste per l'analisi comparativa.

5.3 Sviluppi futuri

Alla luce delle limitazioni emerse, si individuano diverse direzioni promettenti per estendere e raffinare il lavoro sulla generazione automatica di messaggi di commit. Di seguito vengono descritte le prospettive più rilevanti:

1. **Valutazione umana strutturata** — Una possibile evoluzione del framework sperimentale è l'integrazione di una fase di valutazione manuale, coinvolgendo sviluppatori esperti in uno studio controllato. Questo approccio consentirebbe di analizzare aspetti non rilevabili con metriche automatiche, quali l'utilità percepita, la chiarezza e l'aderenza semantica dei messaggi generati. La letteratura ha dimostrato che, in vari contesti, i messaggi generati automaticamente risultano preferiti dagli umani rispetto a quelli reali solo nel 13% dei casi [12]. Una valutazione qualitativa più ampia potrebbe quindi supportare o smentire l'efficacia percepita dei modelli rispetto a scenari d'uso reali.

2. **Adozione di LLM più recenti** — L'evoluzione dei modelli di linguaggio apre nuove opportunità. Architetture come *CodeT5+* o *CodeGemma* rappresentano lo stato dell'arte nella comprensione semantica del codice e potrebbero offrire vantaggi significativi rispetto ai backbone finora impiegati. Studi recenti hanno evidenziato che l'utilizzo di LLM in combinazione con tecniche di retrieval consente di raddoppiare le prestazioni rispetto a baseline più semplici [27]. L'integrazione di modelli di nuova generazione (es. `codellama`, `codegemma-7b`) nel task di CMG merita dunque approfondimenti sistematici.

3. **Data augmentation e filtering** — La qualità del dataset si è rivelata determinante. Una possibile estensione consiste nello sviluppo di pipeline automatiche per generare esempi sintetici, tramite parafrasi o trasformazioni semantiche controllate del messaggio. In parallelo, è auspicabile introdurre filtraggi basati

su euristiche o classificatori di qualità, con l'obiettivo di assegnare maggiore peso ai commit più informativi (i cosiddetti *good-practice commits* [28]) ed escludere esempi rumorosi. Anche fasi di *re-ranking* in fase di inference — ad esempio attraverso moduli di quality-check — potrebbero contribuire ad alzare la qualità media dei messaggi prodotti.

4. **Espansione linguistica e scalabilità** — Il presente studio si è concentrato su progetti Java, ma i risultati potrebbero variare per altri linguaggi. Sarà interessante verificare se i modelli mantengano prestazioni elevate anche su repository in Python, Go, Rust, ecc., dove la struttura del codice e le convenzioni di commit possono differire. Inoltre, l'adozione di dataset su scala industriale — estratti, ad esempio, da GH Archive o BigQuery — permetterebbe di addestrare modelli più robusti e valutare la generalizzabilità delle tecniche in contesti più ampi e diversificati. Metodi basati su retrieval come RACE potrebbero trarre enorme beneficio da una base di conoscenza storica più ampia, mentre andrebbe verificata la capacità di generalizzazione di tecniche di denoising come KADEL in presenza di maggiore rumore.

In conclusione, le prospettive delineate mirano a rendere i sistemi di commit message generation più efficaci, affidabili e applicabili a contesti reali. Combinare valutazioni qualitative, modelli più potenti, dati curati e scenari più complessi potrà costituire la base per soluzioni future sempre più competitive e utili nello sviluppo del software.

Bibliografia

[1] B. Xu and et al., "Codisum: Context-aware neural commit message generation using hybrid representations," 2019. (Citato alle pagine v, 2, 4, 8, 12, 15, 16, 21, 24, 26, 27, 33, 43, 48, 50, 51, 65, 82, 96 e 98)

[2] H. Shi and et al., "Race: Retrieval-augmented commit message generation," *IEEE Transactions on Software Engineering*, 2022. (Citato alle pagine v, 2, 4, 8, 11, 13, 17, 18, 23, 24, 26, 27, 28, 33, 34, 44, 46, 52, 53, 54, 55, 67, 68, 97 e 99)

[3] S. Mukherjee and R. Robbes, "Evaluating the use of commit messages for software maintenance and evolution," *Empirical Software Engineering*, vol. 26, no. 2, pp. 1–30, 2021. (Citato alle pagine 1, 3, 7 e 9)

[4] R. P. Buse and T. Zimmermann, "Information needs for software development analytics," *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pp. 987–996, 2010. (Citato alle pagine 1, 3, 7, 8, 10 e 11)

[5] L. Hattori and M. Lanza, "The nature of commits: An empirical study of open source projects," pp. 63–72, 2008. (Citato alle pagine 1, 7, 9 e 51)

[6] J. Eyolfson, L. Tan, and P. Lam, "Do time of day and developer experience affect commit bugginess?" *Empirical Software Engineering*, vol. 19, no. 2, pp. 303–336, 2014. (Citato alle pagine 1, 7 e 9)

[7] M. Nguyen and J. Chen, "Commit message quality and its impact on maintenance: an empirical study," 2023. (Citato alle pagine 1, 3, 7, 9 e 33)

[8] Q. Liu and et al., "Neural machine translation of commit messages," pp. 373–384, 2018. (Citato alle pagine 1, 3, 8, 12, 16 e 99)

[9] Y. Tian, J. Lawall, and D. Lo, "Automating issue triaging and commit message generation using deep learning," pp. 540–550, 2019. (Citato alle pagine 2, 3, 8, 9 e 33)

[10] W. Tao, Y. Zhou, Y. Wang, and W. Zhang, "Kadel: Knowledge-aware denoising learning for commit message generation," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 5, 2024. (Citato alle pagine 4, 11, 14, 19, 24, 26, 27, 33, 46, 47, 52, 54, 55, 56, 69 e 97)

[11] W. Tao, Y. Wang, Y. Zhou, W. Zhang, and H. Wang, "An evaluation of learning-based commit message generation: How far are we?" in *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 436–447. (Citato alle pagine 5, 21, 24, 33, 38, 44, 45, 46, 52 e 55)

[12] S. Jiang, A. Armaly, and C. McMillan, "Automatically generating commit messages from diffs using neural machine translation," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 135–146. (Citato alle pagine 12, 24 e 100)

[13] Z. Liu, W. Liu, X. Xia, S. Li, D. Lo, and B. Xu, "Atom: Commit message generation based on abstract syntax tree and source code," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2020, pp. 50–61. (Citato a pagina 13)

[14] Y. Wang, Z. Wang, X. Xia, and D. Lo, "Corec: Commit message generation by retrieving and copying relevant code changes," in *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 502–514. (Citato a pagina 13)

[15] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, 2017, pp. 1073–1083. (Citato a pagina 16)

[16] Y. Wang, W. Wang, S. Joty, J. Qi, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2021, pp. 8696–8708. (Citato a pagina 17)

[17] S. Panthaplackel, P. Nie, J. Li, P. Liang, B. Ray, and M. Gligoric, "Learning to update natural language comments based on code changes," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, 2020, pp. 1853–1868. (Citato alle pagine 22, 45 e 53)

[18] Y. Zhou, Y. Wang, W. Tao, and W. Zhang, "Rationale dataset and analysis for the commit messages of the linux kernel out-of-memory killer," in *Proceedings of the 46th International Conference on Software Engineering (ICSE Companion)*. IEEE/ACM, 2024. (Citato alle pagine 23, 24 e 27)

[19] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of ACL*, 2002, pp. 311–318. (Citato a pagina 26)

[20] M. Denkowski and A. Lavie, "Meteor universal: Language specific translation evaluation for any target language," in *Proceedings of the Ninth Workshop on Statistical Machine Translation*, 2014, pp. 376–380. (Citato a pagina 26)

[21] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, "Cider: Consensus-based image description evaluation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4566–4575. (Citato a pagina 26)

[22] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," *Proceedings of the 33rd Inter-*

national Conference on Software Engineering (ICSE), pp. 1–10, 2011. (Citato alle pagine 29, 30, 31, 32, 39, 40, 41 e 42)

[23] "Reproducibility in machine learning," https://ml-science-book.com/reproducibility.html, accessed 2025-11-19. (Citato alle pagine 29, 32, 35 e 38)

[24] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. Sjøberg, "A systematic review of effect size in software engineering experiments," *Information and Software Technology*, vol. 49, no. 11–12, pp. 1073–1086, 2007. (Citato alle pagine 30, 31, 32, 39, 40 e 42)

[25] M. Heričko, M. Goričar, and M. Praprotnik, "Commit-level software change intent classification using a pre-trained transformer-based code model," *Mathematics*, vol. 12, no. 7, p. 1012, 2024. (Citato alle pagine 31, 40 e 41)

[26] J. Cohen, "A power primer," *Psychological bulletin*, vol. 112, no. 1, pp. 155–159, 1992. (Citato alle pagine 31 e 41)

[27] Y. Jiang, Z. Tang, G. Li, X. Xia, and Z. Jin, "Coracmg: Retrieval-augmented commit message generation with in-context learning," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2023, pp. 1323–1335. (Citato a pagina 100)

[28] J. Liu, G. Li, X. Xia, and Z. Jin, "A retrieval-augmented approach for semantic-aware commit message generation," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2021, pp. 932–944. (Citato a pagina 101)

Ringraziamenti

Ringraziamenti qui...