DISCUSSIONE DI LAUREA

HybridRepair

Riparazione automatica di NullPointerException, senza Perfect Fault Localization

CANDIDATO

Mario Ponticiello

RELATORE

Porfirio Tramontana

SEDE

Università Federico II di Napoli · A.A. 2025-2026

02 · IL CONTESTO

Manutenzione e Automated Program Repair

$2,08T

Costo del software di bassa qualità negli USA nel solo 2020, secondo il Consortium for Information & Software Quality. Affidare il debugging al solo intervento umano è sempre meno sostenibile.

L'**APR** non aspetta lo sviluppatore: un sistema automatico affronta il difetto in tre passi.

01
Analizza
il codice sorgente

→

02
Localizza
il difetto

→

03
Genera
la patch

03 · LA TIPOLOGIA DI BUG

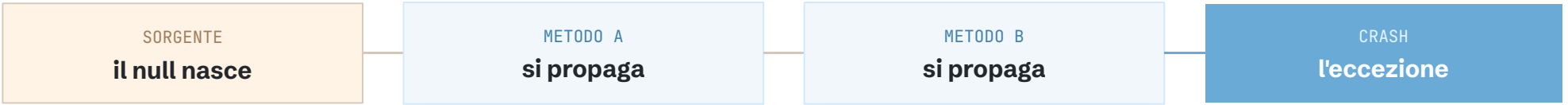
Le NullPointerException

pur essendo semplici, sono tra i **più diffusi**.

37,2%
dei fallimenti in sistemi come Mozilla e Apache

70%
dei crash nelle applicazioni Java in produzione

sono bug del **flusso dei valori**. Il difetto nasce lontano dal crash.



04 · L'IDEA

Dal sintomo alla causa

INPUT

Test falliti + codice sorgente

→

1 · DIAGNOSTICARE

Perché il bug esiste

Risalire per via causale dal crash alla sua radice.

→

2 · GUIDARE

Cosa dovrebbe fare il codice

Tradurre la diagnosi in una specifica semantica per il modello.

→

3 · SINTETIZZARE

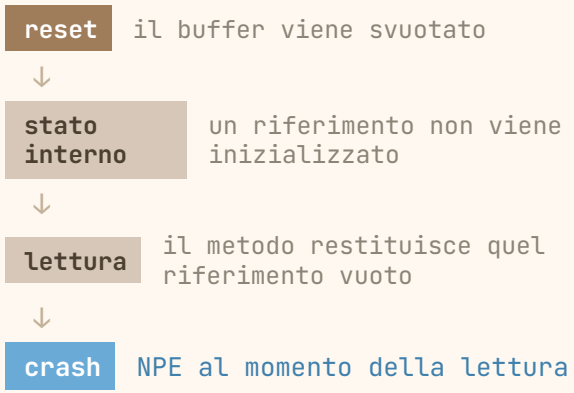
La patch

Corregge la causa individuata, non il sintomo osservato.

05 · ESEMPIO MOTIVAZIONALE

Il bug JacksonCore-8

1 · DIAGNOSTICARE — LA CATENA CAUSALE



2 · GUIDARE — LA SPECIFICA SEMANTICA

Dalla catena causale si ricava **cosa il metodo deve fare** : restituire sempre un contenuto valido, anche vuoto, quando non c'è nulla da restituire ma mai un riferimento non inizializzato.

3 · SINTETIZZARE — LA CORREZIONE

La patch generata rispetta quella specifica: nello scenario testato il valore restituito è sempre valido. Tutti i **test passano**.

06 · FAULT LOCALIZATION

La PFL non è una soluzione

Prima di riparare un bug bisogna sapere **dove si trova** . La maggior parte dei sistemi LLM aggira il problema anziché risolverlo: riceve la posizione del difetto **già come input**.

✗ PERFECT FAULT LOCALIZATION

Una scorciatoia, non una diagnosi

La posizione del difetto è data **già nota in ingresso** . Irrealistica: se uno sviluppatore sapesse già dov'è il bug, lo avrebbe già corretto.

✓ LOGICFL

Localizzazione, non un'assunzione

Ricostruisce la posizione del difetto **in autonomia** , con un ragionamento causale deterministico: nessuna posizione data in ingresso.

171→64 bug risolti da GIANTREPAIR, con e senza PFL: tolta la scorciatoia, la rete di sicurezza sparisce.

07 · IL PIANO

Separare diagnosi e sintesi

DIAGNOSI

Individuare il punto esatto del problema

Richiede un rigore **logico e deterministico** che solo la programmazione logica può garantire.

SINTESI

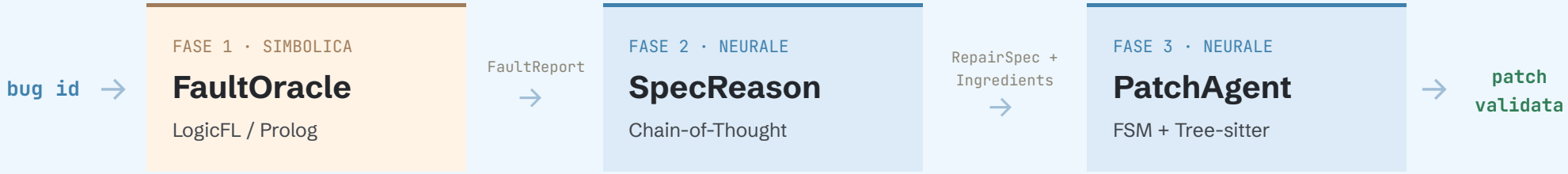
Scrivere codice Java pulito che passa i test

Dominio in cui gli **LLM eccellono**.

Il contributo non è sviluppare i singoli strumenti, ma **progettare l'architettura** che trasforma il referto formale della diagnosi in una guida per la generazione del modello.

08 · ARCHITETTURA

Tre fasi disaccoppiate



FASE 1

Dove e perché

FASE 2

Cosa va modificato

FASE 3

Come

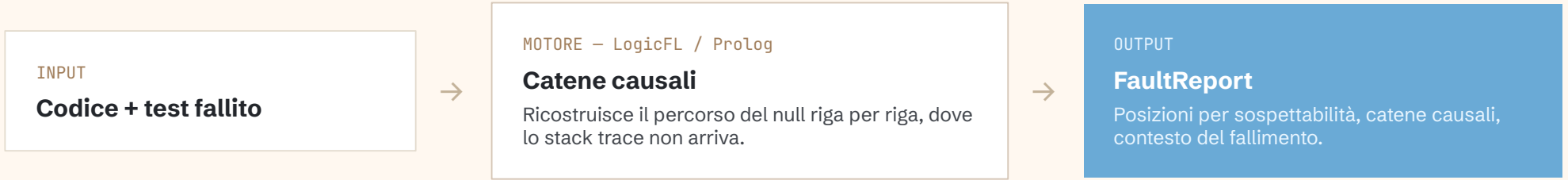
09 · FASE 1 - LOGICA & DETERMINISTICA

FaultOracle

Localizza il difetto e ne fornisce una **diagnosi causale** , solo tramite analisi statica e ragionamento deduttivo in Prolog.

LogicFL

Fault localization in **Prolog** . Costruisce catene causali formali della propagazione del null dando all'LLM un contesto mirato.



SpecReason

Spec per una specifica rigorosa, **Reason** per il ragionamento passo-passo. Trasforma la diagnosi in conoscenza per il modello.

INPUT

FaultReport

Le catene causali e le posizioni prodotte da FaultOracle.

→

COSA SI FA

Ingredient Forge

Raccoglie dal codice il vocabolario reale delle API — campi, metodi, tipi.

Chain-of-Thought

Costringe il modello a esplicitare il ragionamento prima di proporre la correzione.

→

OUTPUT

RepairSpec

Il contesto su cui lavora la fase successiva.

PatchAgent

Traduce la RepairSpec in una modifica concreta del codice, come una **macchina a stati** che attraversa quattro fasi.

INPUT

RepairSpec

La specifica semantica prodotta da SpecReason.

→

COSA SI FA

Esplorazione
Analizza il codice intorno al difetto indicato dalla RepairSpec.

→

Generazione
Propone una modifica concreta al codice Java.

→

Compilazione
Verifica che la modifica sia sintatticamente corretta.

→

Test
Esegue la suite: se fallisce, ritenta con la cronologia dell'errore, fino a 3 volte.

→

OUTPUT

Patch validata

Compila e supera i test.

Obiettivi e domande di ricerca

Misurare in modo rigoroso la capacità del sistema di generare patch **plausibili** e **valide** su bug NPE reali, e confrontare le prestazioni con lo stato dell'arte.

RQ1	Quante patch **plausibili** è in grado di generare HybridRepair sui bug NPE del dataset?
RQ2	Quante delle patch plausibili generate sono anche **valide**, semanticamente equivalenti al Ground Truth?
RQ3	Come si **confrontano** le prestazioni con VibeRepair sullo stesso dataset?

13 · SPERIMENTAZIONE

Dataset e competitor

37 **bug NPE**

estratti da **Defects4J v2**, distribuiti su **12 progetti** Java open source eterogenei.

IL COMPETITOR

VibeRepair

Sistema APR basato solo su LLM, **specification-centric** : traduce il codice in una specifica, la corregge, e genera la patch da quella.

Non localizza il difetto in autonomia: assume la **Perfect Fault Localization** , il punto esatto del bug come dato noto a priori.

14 · SPERIMENTAZIONE

Metriche

PATCH PLAUSIBILE

Compila ed **supera l'intera suite di test** , senza garanzia di correttezza semantica fuori dai casi coperti dai test.

PATCH VALIDA

Una patch plausibile che, dopo revisione manuale, risulta **equivalente al Ground Truth**: risolve la causa, non solo il sintomo.

HR Plaus.
Bug con almeno una patch plausibile, sul totale del dataset.

HR Valida
Precisione: patch plausibili giudicate valide, sul totale delle plausibili.

Plausibile AND Valida
Patch valide sul totale del dataset: base omogenea di confronto con VibeRepair.

15 · RISULTATI

Risultati e confronto con VibeRepair

22/37
Patch plausibili
59,5% — superano la suite di test

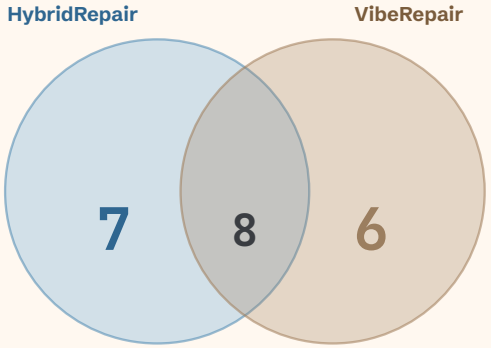
15/22
Patch valide
precisione 68,2% — revisione manuale

40,5%
Validità assoluta
15 valide sull'intero dataset

CONFRONTO CON LO STATO DELL'ARTE

15 vs **14** patch valide, stesso dataset di 37 bug

HybridRepair supera lo stato dell'arte **senza Perfect Fault Localization** , mentre VibeRepair ne dipende strutturalmente.



21 bug su 37 risolti da almeno uno dei due sistemi; +16 da nessuno.

I due approcci sono **complementari** : HybridRepair eccelle sui bug multi-punto, dove la catena causale guida verso tutti i punti da correggere.

16 · LIMITI

Threats to Validity

VALIDITÀ INTERNA

Soggettività nella validazione manuale delle patch, mitigata con un criterio conservativo. Non-determinismo del LLM: i risultati vengono da un'unica esecuzione della pipeline.

VALIDITÀ ESTERNA

Dataset limitato a 37 bug NPE: i pattern osservati potrebbero non trasferirsi ad altre categorie di difetti.

VALIDITÀ DI COSTRUTTO

Il Ground Truth è la soluzione degli sviluppatori originali: alternative ugualmente valide rischiano di essere scartate, un bias che sottostima le prestazioni.

17 · CONCLUSIONI

Unire i due mondi: senza PFL, con fault localization euristica

01

La causa radice, non la correzione superficiale

HybridRepair fa precedere la generazione della patch da una **localizzazione causale autonoma**, LogicFL, e da una **guida semantica**, SpecReason.

02

Competitivo, senza la rete di sicurezza della PFL

HybridRepair supera VibeRepair sullo stesso dataset. Il contributo non sta nel margine numerico, ma nel fatto di localizzare il difetto **in autonomia, con una fault localization euristica** , invece di assumerlo noto a priori come VibeRepair.

Limiti e direzioni: rafforzare l'oracolo di test, una validazione critica indipendente dal modello generatore, ed estendere il dataset ad altre categorie di difetti.

Grazie per l'attenzione

Resto a disposizione per le vostre domande.

CANDIDATO
Mario Ponticiello

HYBRIDREPAIR
Riparazione di NPE