0x06tudio 0x06tudio@gmail.com
Progetto didattico di Ingegneria del Software
Università degli Studi di Padova

Studio Tecnologico: Comprensione Comportamentale del Codice (AI Code Analysis)

Approfondimento Tecnologico Modulo 5 – Progetto SpecCheck (Capitolato C9)

Informazioni sul documento

Versione:	0.1.0
Stato:	Approvato
Data ultimo aggiornamento:	2026/11/16
Uso:	Interno
Destinatari:	0x06tudio

Registro delle modifiche

Versione	Data	Descrizione	Autore	Verificatore
0.1.0	2026/11/16	Stesura del documento	Alberto Cavalletto	Catalin Chirilov

Indice

1 Introduzione **3**
1.1 Scopo del documento . . . 3
1.2 Riferimenti . . . 3
1.2.1 Riferimenti normativi . . . 3
1.2.2 Riferimenti informativi . . . 3

2 Contesto e Sfide nel Capitolato C9 **3**

3 Analisi delle Tecnologie Chiave **4**
3.1 Inferenza Comportamentale tramite LLM (Intent Extraction) . . . 4
3.2 Analisi Statica (AST) vs Analisi Semantica Neurale . . . 4
3.3 Validazione Inversa (Reverse Engineering Semantico) . . . 4

4 Architettura Proposta e Proof of Concept (PoC) **5**
4.1 Flusso operativo del PoC . . . 5

5 Valutazione Comparativa e Analisi dei Rischi **5**
5.1 Confronto tra Strategie di Verifica del Codice . . . 5
5.2 Matrice dei Rischi e Mitigazioni . . . 6

6 Conclusioni e Raccomandazioni per il Progetto **6**

1 Introduzione

1.1 Scopo del documento

Il presente documento costituisce lo studio tecnologico di dettaglio relativo al Modulo 5: Comprensione Comportamentale del Codice (AI Code Analysis) per il progetto SpecCheck (Capitolato C9, proposto da Zucchetti S.p.A.).

L'obiettivo dell'analisi è definire metodologie, pattern architetturali e librerie software per realizzare la Fase 2 del capitolato (Specifiche dei Requisiti verso Codice Sorgente): verificare in modo deterministico e semantico che il codice sorgente implementi realmente i vincoli, le funzionalità e le regole di business descritte nelle specifiche, rilevando scostamenti logici, omissioni ed eventuali comportamenti non richiesti (allucinazioni del codice sorgente).

1.2 Riferimenti

1.2.1 Riferimenti normativi

- Capitolato d'appalto C9: *SpecCheck – Sviluppo di Skills per OpenCode per la verifica della tracciabilità end-to-end da casi d'uso a requisiti e codice sorgente*, Zucchetti S.p.A., A.A. 2026/2027;
- Regolamento del Progetto Didattico (PD1.pdf), Prof. Tullio Vardanega, Prof. Riccardo Cardin;
- Norme di Progetto, redatte dal gruppo 0x06 Studio.

1.2.2 Riferimenti informativi

- Documentazione e architettura agenti OpenCode;
- Studio tecnologico interno: *Tracciabilità Semantica: Casi d'Uso vs Specifiche* (0x06 Studio, Modulo 4);
- Studio tecnologico interno: *Architettura di Integrazione, Tooling e Dashboarding* (0x06 Studio, Modulo 6);
- Studio tecnologico interno: *Benchmark per Agenti Software e Metodologie Spec-Driven* (0x06 Studio, Modulo 2);
- Parser AST Tree-sitter: libreria multi-linguaggio per la generazione incrementale di alberi sintattici;
- Standard SARIF (Static Analysis Results Interchange Format), OASIS Standard;
- Feng et al. (2020), *CodeBERT: A Pre-Trained Model for Programming and Natural Languages*;
- Chen et al. (2021), *Evaluating Large Language Models Trained on Code* (HumanEval).

2 Contesto e Sfide nel Capitolato C9

La Fase 2 del capitolato C9 pone il gruppo di fronte a una serie di problemi aperti nel panorama del software engineering moderno assistito da agenti LLM:

1. **I limiti dell'analisi statica classica.** Compilatori, linter (es. ESLint, Flake8) e suite di test unitari verificano la conformità sintattica e il passaggio di asserzioni puntuali. Tuttavia, una funzione può essere sintatticamente valida e superare i test unitari pur implementando un calcolo errato sul dominio applicativo o tralasciando controlli di autorizzazione esplicitati nella specifica.

2. **I limiti dell'analisi neurale pura (Full-Code Prompting).** Fornire interi file sorgente all'interno del contesto dell'LLM genera problemi di *Lost-in-the-Middle*, elevato consumo di token e allucinazioni semantiche. Su repository enterprise, la verifica deve scalare senza eccedere le finestre di contesto.

3. **Vincolo di esecuzione su modelli locali (Gemma 4 26B e Qwen 3.8 27B).** Come indicato dal proponente Zucchetti, SpecCheck deve poter operare in ambienti con Small Language Models (SLM) locali senza dipendere necessariamente da API cloud proprietarie. Ne consegue che il payload fornito all'LLM debba essere rigorosamente pre-elaborato e ridotto al minimo rumore sintattico.

3 Analisi delle Tecnologie Chiave

3.1 Inferenza Comportamentale tramite LLM (Intent Extraction)

L'inferenza comportamentale non si limita a sintetizzare le righe di codice (*Code Summarization*), ma ne estrae la semantica operativa traducendola in uno schema formale composto da:

- **Precondizioni:** vincoli sui parametri in ingresso e controlli di guardia;
- **Postcondizioni:** trasformazioni di stato, valori ritornati ed effetti collaterali (scrittura su database, emissione di eventi);
- **Gestione degli errori:** eccezioni sollevate e scenari di fallback.

3.2 Analisi Statica (AST) vs Analisi Semantica Neurale

Per ottenere la massima affidabilità senza allucinazioni, il Modulo 5 propone un'architettura ibrida e complementare:

- **Strumenti deterministici (AST con Tree-sitter):** vengono utilizzati per estrarre la mappa dei simboli, le firme delle funzioni, le interfacce, le dipendenze e l'albero di controllo del flusso. L'AST rimuove formattazione, commenti obsoleti e dettagli di basso livello, isolando la struttura logica;
- **Modello neurale (LLM vincolato):** riceve esclusivamente il frammento AST rilevante e il requisito corrispondente estratto dalla Fase 1, limitandosi a eseguire il confronto semantico (*Natural Language Inference*).

3.3 Validazione Inversa (Reverse Engineering Semantico)

La metodologia implementata segue un flusso in tre stadi:

1. **Estrazione:** l'analisi AST unita all'LLM genera la specifica *de facto* del componente («Cosa fa il codice»);
2. **Confronto:** la specifica estratta viene confrontata con la specifica di progetto formalizzata nella Fase 1 («Cosa doveva fare il codice»);
3. **Verdetto:** assegnazione dello stato di conformità secondo una quadri-partizione standardizzata:

- `COVERED`: conformità piena tra specifica e implementazione;

- `PARTIAL`: la logica principale è presente, ma mancano validazioni o rami secondari previsti;
- `CONTRADICTION`: il codice esegue un comportamento opposto o incompatibile con il requisito;
- `UNTRACKED_CODE`: codice che introduce funzionalità o endpoint non tracciati in alcuna specifica (requisito opzionale del capitolato).

4 Architettura Proposta e Proof of Concept (PoC)

In conformità con quanto progettato nel Modulo 6 (Tooling e Integrazione OpenCode), la logica di analisi del codice risiede all'interno del Sidecar ed è incapsulata nel tool deterministico `speccheck_verify_code`.

4.1 Flusso operativo del PoC

1. L'agente OpenCode riceve il comando di verifica su un modulo;
2. Il Sidecar invoca Tree-sitter per parsare il file ed estrarre la lista delle classi e funzioni;
3. Il Sidecar recupera dal database vettoriale locale (costruito dal Modulo 4) i requisiti collegati a quel contesto tramite similarità semantica;
4. Viene assemblato un prompt snello contenente solo il requisito e l'estratto AST della funzione target;
5. Il modello locale (Gemma 4 26B o Qwen 3.8 27B) produce il verdetto validato tramite lo schema dati strutturato;
6. I risultati vengono aggregati nel formato SARIF e inviati alla dashboard HTML.

5 Valutazione Comparativa e Analisi dei Rischi

5.1 Confronto tra Strategie di Verifica del Codice

Criterio	**Unit Test Tradizionali**	**Full-Code Prompting (tutto il codice nell'LLM)**	**Pipeline Ibrida (AST + NLI Strutturato)**
Copertura Logica	Bassa (limitata ai soli test scritti)	Media (vulnerabile ad allucinazioni)	Alta (ispezione semantica su AST)
Consumo Token	Nullo	Altissimo (proporzionale alle linee di codice)	Basso (viene inviato solo lo scheletro AST)
Resa su SLM Locali	Non applicabile	Bassa (saturazione della finestra di contesto)	Ottima (contesto compatto e mirato)
Spiegabilità	Minima (Pass o Fail)	Bassa	Alta (output strutturato e motivato)

5.2 Matrice dei Rischi e Mitigazioni

N.	Rischio	Impatto	Probabilità	Strategia di Mitigazione
1	File di grandi dimensioni che superano la finestra di contesto	Alto	Media	Scomposizione modulare basata su AST. L'analisi avviene per singola unità logica (funzione o metodo) e non per l'intero file.
2	Falsi positivi generati dall'LLM su requisiti complessi	Medio	Media	Doppia validazione (*Adversarial Review*) per i soli verdetti di contraddizione prima della segnalazione definitiva.
3	Eterogeneità dei linguaggi target nel codice sorgente	Medio	Bassa	Utilizzo del parser unificato Tree-sitter, che garantisce grammatiche pronte per C, Python, JavaScript, TypeScript e Java.

6 Conclusioni e Raccomandazioni per il Progetto

Lo studio condotto evidenzia che demandare per intero la verifica del codice sorgente all'inferenza non vincolata di un LLM non è compatibile con i requisiti di affidabilità e determinismo richiesti da Zucchetti per il progetto SpecCheck.

Si raccomanda al gruppo 0x06 Studio di adottare le seguenti linee guida per il Proof of Concept (PoC) e per il Minimum Viable Product (MVP):

1. **Adozione di Tree-sitter nel Sidecar:** incapsulare il parsing sintattico nei tool deterministici del Sidecar, escludendo l'uso dell'LLM per compiti sintattici elementari;
2. **Standardizzazione dello schema di scambio:** aderire formalmente alla quadrupla di conformità (`COVERED`, `PARTIAL`, `CONTRADICTION`, `UNTRACKED_CODE`) per garantire l'immediata interoperabilità con la dashboard del Modulo 6;
3. **Ottimizzazione per Small Language Models:** mantenere i prompt al di sotto dei 1500 token complessivi mediante estrazione selettiva dei nodi AST, assicurando l'esecuzione fluida e a costo zero su Gemma 4 26B e Qwen 3.8 27B.