Università del Piemonte Orientale

Dipartimento di Scienze e Innovazione Tecnologica

Corso di Laurea in Informatica

Relazione per la prova finale

Modelli di Processo e Stratificazione a Larga Scala per l'Analisi Comportamentale della Collaborazione su GitHub

Tutore interno:

Prof.ssa Stefania Montani

Candidato:

Federico Mondelli

Anno Accademico 2024/2025

Sommario

Il presente lavoro propone una piattaforma software per l'analisi su larga scala dei processi di sviluppo Open Source, affrontando le criticità legate al volume e all'eterogeneità dei dati provenienti da GitHub Archive[3].

L'architettura realizzata disaccoppia il sistema in due domini indipendenti, Acquisizione ed Elaborazione, ciascuno governato da una propria Clean Architecture[6].

Questa separazione consente di gestire il reperimento dei dati in modalità streaming e resiliente, e la successiva fase analitica senza caricamento integrale in memoria, superando i limiti della memoria fisica.

La metodologia adottata introduce una pipeline di Stratificazione che segmenta i repository in Archetipi basati su metriche quantitative, riducendo l'entropia del dataset.

Su questa base si innesta un Process Mining[9] incentrato sul contributore che, mappando l'attività dello sviluppatore nel contesto del singolo progetto, porta alla luce le reali dinamiche operative, rendendo tangibili e misurabili le differenze strutturali tra i diversi ecosistemi software.

Indice

Capitolo 1

Introduzione

La disponibilità di grandi moli di dati relativi alle attività di sviluppo software, accessibili tramite piattaforme come GitHub[4], offre opportunità senza precedenti per la comprensione delle dinamiche collaborative nell'Open Source.

Tuttavia, l'analisi di tali dati pone significative sfide ingegneristiche: i volumi massivi (Big Data), la natura non strutturata degli eventi e la necessità di correlare azioni distribuite nel tempo richiedono architetture software capaci di scalare oltre le risorse di una singola workstation.

1.1 Scopo del lavoro in breve

Il presente elaborato, redatto a conclusione di uno studio guidato interno presso il Dipartimento di Scienze e Innovazione Tecnologica (DISIT), ha per oggetto la progettazione di una pipeline analitica scalabile.

L'obiettivo primario è la trasformazione di stream grezzi di eventi in modelli di processo strutturati, permettendo l'elaborazione di grandi moli di dati su hardware standard per ricostruire e visualizzare i flussi operativi reali messi in atto dai contributori nei progetti Open Source.

1.2 Pregresso

Il punto di avvio dell'attività è rappresentato esclusivamente dalla disponibilità dei dataset pubblici del progetto GitHub Archive (GHArchive)[3], che aggrega l'attività di GitHub in archivi JavaScript Object Notation (JSON) massivi.

La necessità di colmare il divario tecnologico tra il dato grezzo disponibile e un modello interpretabile, in assenza di strumenti integrati capaci di gestire l'intero ciclo di vita del dato (dall'acquisizione al mining) su risorse limitate, costituisce la base su cui si innesta il lavoro.

1.3 Strumenti

Lo svolgimento dell'attività è stato supportato dall'utilizzo di specifiche tecnologie imposte dal contesto di studio:

- **Sorgente Dati:** Utilizzo esclusivo di GHArchive[3], iniziativa che storicizza e rende accessibile l'intera timeline pubblica degli eventi GitHub[5] sotto forma di archivi orari.
- **Motore di Analisi:** Adozione obbligatoria della libreria Process Mining for Python[7] (PM4Py)[2] per l'esecuzione degli algoritmi di discovery; tale vincolo ha determinato la scelta di Python[7] come ecosistema di sviluppo per l'intera infrastruttura.

1.4 Specifiche

A livello metodologico, il requisito fondamentale imposto per l'analisi riguarda la prospettiva di osservazione dei dati.

Il requisito è analizzare le attività svolte dallo sviluppatore nelle singole repository.

Definendo la traccia come unione di utente e progetto, il sistema ricostruisce le sequenze di azioni eseguite sul singolo progetto, garantendo che ogni flusso di lavoro resti legato al suo specifico contesto.

Capitolo 2

Lavoro Svolto

2.1 Panoramica del Sistema

La soluzione sviluppata articola due Clean Architecture[6] indipendenti, disaccoppiando l'acquisizione dall'analisi per gestire l'elaborazione out-of-core di massicci volumi di dati.

Mediante metriche multidimensionali, i repository vengono stratificati in Archetipi, riducendo l'eterogeneità per abilitare confronti statistici significativi.

Tale base abilita l'estrazione di modelli comportamentali che, superando la visione frammentata dei singoli repository, ricostruiscono il flusso lavorativo degli sviluppatori, permettendo di modellare visivamente e quantificare le divergenze strutturali tra le diverse categorie di progetti Open Source.

2.2 Analisi dei Vincoli e Strategia Architetturale

La progettazione del sistema è stata guidata dalla necessità di superare limiti fisici e computazionali.

Di seguito vengono descritte le principali criticità individuate e le soluzioni architetturali adottate.

2.2.1 Disaccoppiamento Asincrono

L'analisi dei vincoli ha evidenziato l'incompatibilità tra la latenza di rete dell'Ingestione (I/O bound) e la complessità computazionale dell'Analisi (CPU bound).

Si è scelto di separare il sistema in due entità asincrone che comunicano solo via File System: l'Ingestor costruisce il Data Lake locale una tantum, mentre l'Analyzer opera iterativamente su dati statici.

Questo approccio garantisce flessibilità nella selezione del range temporale di studio, permettendo di definire dinamicamente il perimetro di analisi su dati già persistiti senza richiedere nuovi accessi alla rete.

2.2.2 Doppia Clean Architecture

Per garantire l'isolamento delle dipendenze e la manutenibilità, sono state istanziate due architetture esagonali distinte.

L'Ingestor incapsula logiche di networking HyperText Transfer Protocol (HTTP) e compressione, mentre l'Analyzer integra motori statistici come Polars[10] e PM4Py[2], mantenendo i rispettivi Domain Layer puri e testabili separatamente.

2.3 Sottosistema di Ingestione (Resilienza e I/O)

Il modulo di ingestione rappresenta il punto di ingresso dei dati nel sistema. È stato progettato per massimizzare l'affidabilità del download e garantire la consistenza del dataset locale a fronte di errori di rete.

2.3.1 Pipeline in Streaming

Per mantenere l'occupazione di memoria costante ($O(1)$), l'acquisizione avviene rigorosamente in streaming.

Il sistema esegue download, parsing JSON e distillazione (estrazione dei soli campi rilevanti) riga per riga, senza mai caricare l'intero archivio orario in Random Access Memory (RAM).

2.3.2 Idempotenza Basata su Indice

L'Ingestor consulta un indice persistente per saltare il lavoro già completato con successo.

In caso di errore durante l'elaborazione, l'indice non viene aggiornato; questo meccanismo segnala l'ora come "non completata", permettendo la ri-schedulazione del download nelle esecuzioni successive senza generare stati inconsistenti.

2.3.3 Resilienza alla Sorgente (Gestione 404)

Per mitigare l'instabilità di GitHub Archive, le ore non reperibili alla fonte vengono registrate nell'indice come intervalli validi ma privi di eventi ("assenti").

Questo accorgimento previene la creazione di falsi buchi temporali nel dataset, permettendo al sistema di chiudere contabilmente la giornata (24 slot gestiti) senza bloccare la pipeline di consolidamento.

2.3.4 Flessibilità Operativa

Il sistema espone tramite CLI diverse modalità per coprire ogni scenario operativo:

- **Modalità Atomica:** Elaborazione di specifiche ore o gruppi di ore per il debugging puntuale.
- **Modalità Batch:** Elaborazione di interi range temporali per il popolamento massivo.

2.4 Ottimizzazione dello Storage e Validazione

La gestione efficiente dello spazio su disco e la coerenza temporale sono prerequisiti fondamentali per l'analisi. Questa sezione illustra le tecniche di consolidamento e verifica applicate ai dati acquisiti.

2.4.1 Consolidamento: JSONL to Parquet

Il consolidamento dello storage non richiede un comando esplicito, ma viene innescato automaticamente dal sistema al raggiungimento della completezza giornaliera.

I file temporanei JSONL vengono convertiti in un unico file Apache Parquet[8], abilitando la compressione efficiente e ottimizzando le future letture analitiche tramite projection pushdown.

2.4.2 Validazione Temporale

Un modulo di raccordo analizza gli indici generati dall'Ingestor per identificare il periodo contiguo più lungo di dati disponibili.

Ciò garantisce che l'Analisi successiva operi su un range temporale coerente, evitando lacune nei dati che invaliderebbero le metriche temporali del Process Mining.

2.5 Sottosistema di Analisi (Efficienza e Metodologia)

L'Analyzer è il motore computazionale della tesi, responsabile della trasformazione del dato grezzo in modelli di processo.

Le scelte implementative qui descritte mirano a gestire la complessità computazionale su grandi volumi di dati.

2.5.1 Granularità Operativa

L'Analyzer espone tre stadi di esecuzione distinti: preparazione dei dati, modellazione dei processi e analisi comparativa.

Questa segmentazione permette il caching su disco dei risultati intermedi, evitando di ricalcolare metriche costose per ogni nuova visualizzazione o affinamento dell'analisi.

2.5.2 Elaborazione Out-of-Core

Per processare milioni di repository, l'intero flusso sfrutta la Lazy Evaluation di Polars[10].

Il sistema costruisce piani di esecuzione che vengono risolti scrivendo i risultati in streaming, direttamente su disco, senza mai caricare l'intero dataset processato nella memoria centrale.

2.5.3 Sicurezza: Schema-on-Read

Mentre l'Ingestione massimizza la velocità di scrittura, la validazione semantica rigorosa è delegata alla fase di lettura dell'Analyzer che applica i seguenti filtri:

- Vengono rimossi gli eventi appartenti ad agenti auomatici (bot).
- Vengono normalizzati i nomi degli eventi per per esplicitarne la semantica operativa.
- Selezionamento solo delle azioni pertinenti al flusso di lavoro.

2.5.4 Calcolo e Normalizzazione delle Metriche

Il sistema elabora quattro indicatori dimensionali per ogni repository.

Per garantire la confrontabilità tra progetti con diversa anzianità, i valori cumulativi vengono normalizzati dividendoli per l'età esatta del progetto (in giorni):

- **Workload:** Volume di attività tecnica.
- **Collaboration:** Numero di attori distinti coinvolti attivamente.
- **Engagement:** Volume di interazioni sociali.
- **Popularity:** Metriche di interesse esterno.

Questa operazione converte il dato grezzo in una misura di intensità operativa giornaliera, permettendo di valutare la velocità di sviluppo intrinseca a prescindere dalla durata storica del progetto.

2.5.5 Soglie Dinamiche e Quantili

In risposta alla distribuzione asimmetrica dei dati, il sistema non adotta soglie statiche, ma calcola i riferimenti statistici adattandosi dinamicamente alla popolazione attiva.

Vengono individuati tre Quantili critici che fungono da livelli di discriminazione:

- **Q50 (Mediana):** Valore centrale che identifica lo standard operativo medio.
- **Q90 (Top 10%):** Soglia di eccellenza che isola i progetti con attività strutturata.
- **Q99 (Top 1%):** Livello critico per l'identificazione dei progetti outlier.

2.5.6 Classificazione e Definizione degli Archetipi

Sulla base delle soglie calcolate, ogni repository viene classificata per ciascuna metrica assegnando un'etichetta semantica che ne definisce il posizionamento relativo:

- **Zero:** Assenza di attività rilevata sulla metrica specifica.
- **Low:** Attività presente ma inferiore alla mediana ($\leq$ Q50).
- **Medium:** Attività superiore alla media ($\leq$ Q90).
- **High:** Attività intensa, nel decile superiore ($\leq$ Q99).
- **Giant:** Attività estrema, superiore al 99-esimo percentile.

L'Archetipo è definito dalla combinazione vettoriale di queste quattro etichette.

Questo permette di isolare cluster omogenei (ad esempio, progetti ad alto Workload ma zero Collaboration) su cui applicare il Process Mining in modo mirato.

2.6 Preparazione al Process Mining

Prima di applicare gli algoritmi di mining, i dati necessitano di una trasformazione semantica. In questa fase vengono definiti i paradigmi fondamentali che guidano la ricostruzione dei flussi di lavoro.

2.6.1 Prospettiva Incentrata sullo Sviluppatore

Il fulcro della tesi è l'adozione di una prospettiva focalizzata sull'operatore umano. La "traccia" del processo è identificata univocamente dalla coppia **Attore + Repository** (`actor_id` concatenato a `repo_id`).

Questo paradigma permette di modellare la sequenza di lavoro dello sviluppatore all'interno di uno specifico contesto, evitando di mescolare flussi appartenenti a progetti diversi.

2.6.2 Aggregazione per Archetipo

Il sistema aggrega le tracce di tutti gli attori appartenenti allo stesso Archetipo in un unico log logico.

Questo approccio astrae le peculiarità del singolo progetto per far emergere il "comportamento medio" macroscopico dello strato analizzato.

2.7 Esecuzione del Mining e Output

Questa fase rappresenta il cuore dell'elaborazione algoritmica.

Viene qui descritto l'approccio tecnico utilizzato per estrarre modelli di processo leggibili e performanti dai dati preparati.

2.7.1 Preprocessing Lazy e Materializzazione Selettiva

Per ottimizzare l'uso delle risorse, la preparazione dei dati per l'algoritmo è delegata a Polars[10].

Il dataset viene materializzato in memoria (EventLog) solo al momento dell'esecuzione, lavorando esclusivamente sul sottoinsieme di dati già puliti e pertinenti all'Archetipo corrente.

2.7.2 Algoritmo Heuristics Miner

È stato scelto l'algoritmo Heuristics Miner[1] per la sua robustezza al rumore.

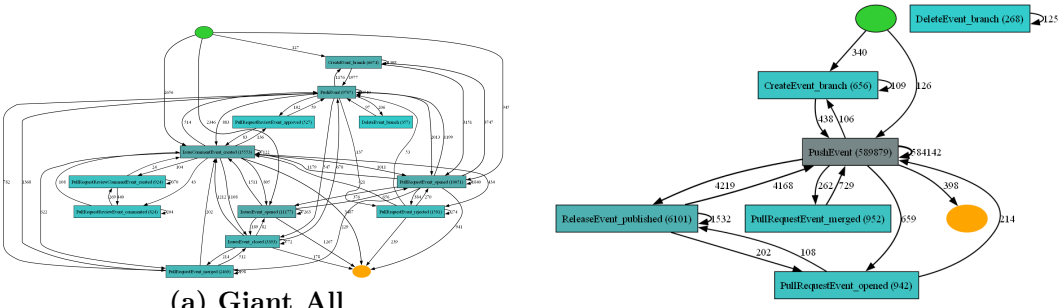
I parametri di soglia (*dependency threshold*) sono stati tarati per potare gli archi rari, restituendo modelli leggibili che rappresentano i flussi di lavoro principali (*mainstream behavior*).

2.7.3 Dualità Frequenza-Performance

Il mining produce due output distinti per ogni Archetipo: il modello delle **Frequenze** (percorsi più battuti) e quello delle **Performance** (tempi medi di attraversamento).

Questa dualità permette di analizzare parallelamente sia le abitudini operative sia i colli di bottiglia temporali.

La Figura2.1 illustra concretamente la diversità strutturale emersa dal mining. È possibile osservare la contrapposizione tra un workflow reticolare e denso di interazioni, tipico dei progetti "Giant" (a sinistra), e un flusso marcatamente lineare e sequenziale, caratteristico dei progetti a bassa collaborazione (a destra).



(a) Giant_All **(b) Giant_Pop_LowCollab**

Figura 2.1: Confronto tra i modelli di frequenza generati dall'algoritmo Heuristics Miner. (a) L'archetipo *Giant_All* mostra un processo complesso con forti cicli di revisione (Issue/Commenti). (b) L'archetipo *Giant_Pop_LowCollab* evidenzia un comportamento lineare focalizzato sulla contribuzione diretta del codice (PushEvent), con scarse ramificazioni collaborative.

2.8 Analisi Comparativa Strutturale

L'ultimo step della pipeline quantifica le differenze tra gli Archetipi emersi.

L'analisi si articola in tre fasi sequenziali automatiche.

2.8.1 Profilazione Morfologica (Identikit)

Ogni Archetipo è sottoposto preliminarmente a una profilazione individuale.

Il sistema calcola indici di Densità del Grafo e Complessità Ciclomatica per valutare il grado di intreccio del flusso di lavoro, isolando inoltre le attività dominanti e i colli di bottiglia temporali specifici di quel cluster.

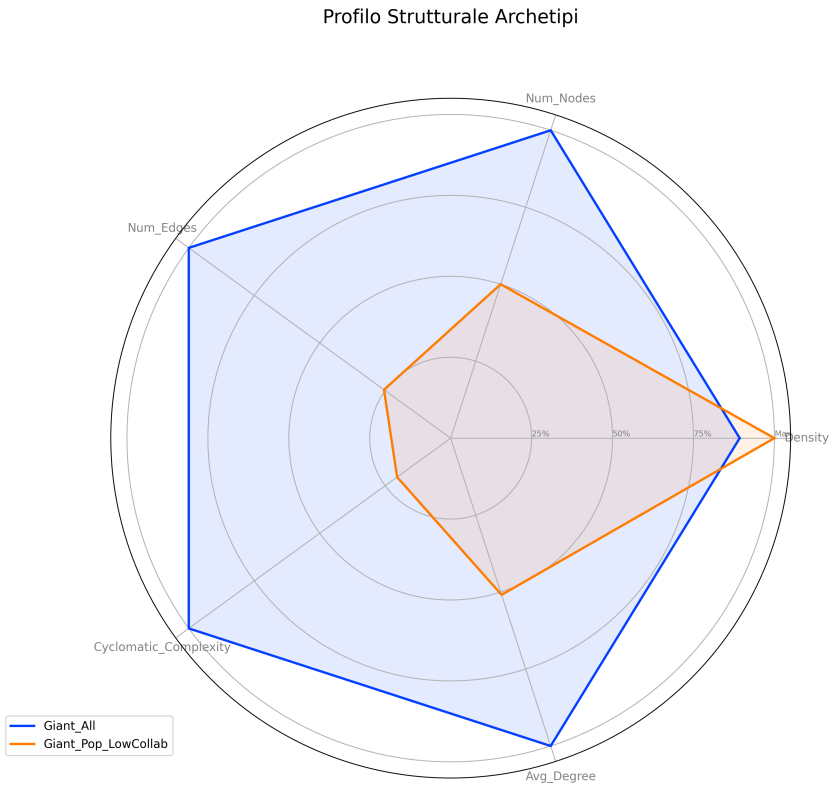


Figura 2.2: Radar Chart che confronta il profilo morfologico (Densità, Complessità) dei diversi Archetipi.

2.8.2 Metriche di Similarità Globale

Per il confronto matriciale tra diversi archetipi specifici, vengono adottate due metriche complementari:

- **Indice di Jaccard:** Misura la sovrapposizione topologica pura (nodi e archi condivisi), ignorando il peso delle connessioni.
- **Distanza di Frobenius Normalizzata:** Quantifica la distanza Euclidea tra le matrici di adiacenza normalizzate, valutando la divergenza nella distribuzione del carico di lavoro o dei tempi, indipendentemente dai volumi assoluti.

2.8.3 Divergenza a Coppie (TVD e Heatmap)

Per l'analisi di dettaglio tra coppie, viene calcolata la **Total Variation Distance (TVD)**, derivata dalla distanza di Manhattan, che esprime la divergenza comportamentale come valore percentuale.

A supporto dell'indagine, vengono generate **Heatmap Differenziali** ($\Delta = M_A - M_B$) che spazializzano le differenze, evidenziando quali transizioni specifiche causano lo scostamento tra i due processi.

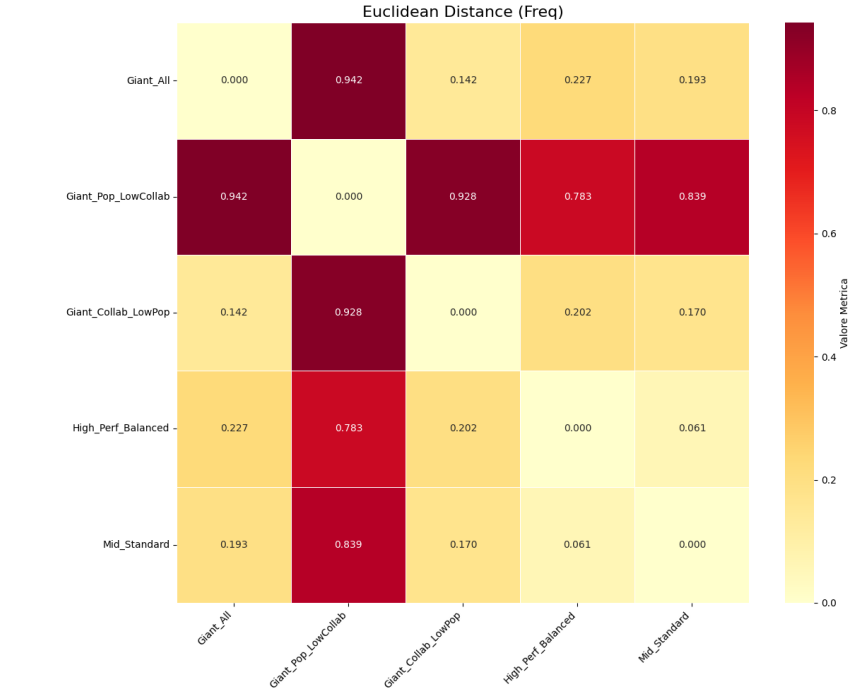


Figura 2.3: Matrice delle distanze di Frobenius (Frequenza) tra tutti gli Archetipi.

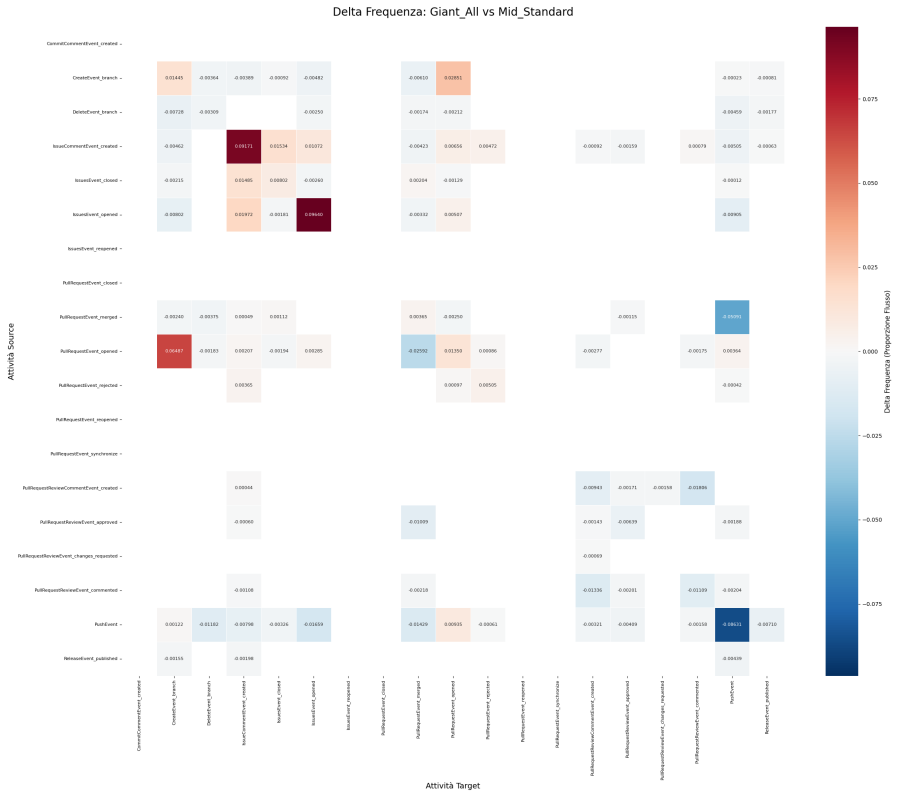


Figura 2.4: Heatmap Differenziale (Frequenza) tra due Archetipi. Il colore rosso/blu indica le transizioni caratteristiche di ciascun gruppo.

Capitolo 3

Conclusioni e sviluppi futuri

Il lavoro ha validato l'efficacia di un approccio scalabile per l'analisi di processi su larga scala.

L'adozione di strategie out-of-core e storage colonnare (Parquet)[8] ha permesso di superare i vincoli di memoria delle workstation standard, abilitando il Process Mining su dataset massivi senza ricorrere a complessi cluster di calcolo distribuiti.

3.1 Risultati conseguiti

Sotto il profilo architetturale, il disaccoppiamento asincrono ha garantito la resilienza del sistema, preservando l'integrità del Data Lake anche a fronte di interruzioni di rete.

Sul piano metodologico, la stratificazione in Archetipi si è rivelata essenziale: l'uso di quantili dinamici ha isolato cluster omogenei, riducendo l'entropia e producendo modelli leggibili.

L'approccio focalizzato sull'attore, raffinato identificando la traccia sulla coppia attore-repository, ha permesso di ricostruire fedelmente i flussi di lavoro.

Questo ha evidenziato come le abitudini operative degli sviluppatori cambino strutturalmente in base alla complessità e alla tipologia del progetto analizzato.

3.2 Sviluppi futuri

L'infrastruttura pone le basi per diverse evoluzioni:

- **Calcolo Distribuito:** Migrazione verso framework come Apache Spark, sfruttando l'architettura a step per parallelizzare l'analisi su cluster e ridurre ulteriormente i tempi.
- **Classificazione Comportamentale tramite Conformance Checking:** Misurazione del grado di aderenza (fitness) della traccia del singolo sviluppatore rispetto ai modelli di tutti gli Archetipi, al fine di individuare quale profilo di riferimento ne descriva meglio il reale comportamento operativo.
- **Estensione delle Fonti:** L'astrazione del Domain Layer permetterebbe di integrare altre sorgenti dati (es. GitLab, Bitbucket) per validare se i pattern comportamentali emersi siano specifici di GitHub o generalizzabili all'intero ecosistema Open Source.

Ringraziamenti

Desidero ringraziare la Prof.ssa [Stefania Montani] e la Prof.ssa [Anna Sapienza] per la guida tecnica e la disponibilità dimostrata durante lo svolgimento di questo studio guidato interno.

Un ringraziamento va inoltre al Dipartimento di Scienze e Innovazione Tecnologica per aver fornito il contesto e gli strumenti necessari alla realizzazione del progetto.

Infine, ringrazio la mia famiglia e i miei amici per il costante sostegno durante l'intero percorso di studi.

Bibliografia

[1] Alessandro Berti, Sebastiaan van Zelst, et al. Heuristics miner documentation. `https://pm4py-source.readthedocs.io/en/stable/pm4py.algo.discovery.heuristics.html`. Ultimo accesso: novembre 2025.

[2] Alessandro Berti, Sebastiaan van Zelst, et al. Pm4py web documentation. `https://pm4py-source.readthedocs.io/en/stable/`. Ultimo accesso: Novembre 2025.

[3] GH Archive. Gh archive: Github's public timeline. `https://www.gharchive.org/`. Ultimo accesso: novembre 2025.

[4] GitHub, Inc. Github development platform. `https://github.com/`. Ultimo accesso: novembre 2025.

[5] GitHub, Inc. Github events documentation, 2024. Ultimo accesso: novembre 2025.

[6] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[7] Python Software Foundation. Python language reference. `https://www.python.org/`. Ultimo accesso: novembre 2025.

[8] The Apache Software Foundation. Apache parquet documentation, 2024. Accessed: 2025-11-26.

[9] Wil van der Aalst. *Process Mining: Data Science in Action*. Springer, 2016.

[10] Ritchie Vink and Contributors. Polars user guide: Lazy api, 2024. Ultimo accesso: novembre 2025.