Reverse Engineering Report for Mark

a.prisco50@studenti.unisa.it 0522501976
c.ranieri7@studenti.unisa.it 0522501977

June 30, 2025

1 Version History

Versione	Data	Descrizione
0.1	27/06/2025	Stesura della struttura del documento.
1.0	27/06/2025	Prima stesura del documento.
1.1	27/06/2025	Modifica della sezione *Codice*.
1.2	27/06/2025	Modifica della sezione *Diagrammi Statici*.
1.3	27/06/2025	Modifica della sezione *Requisiti Funzionali e User Stories*.
1.4	27/06/2025	Modifica della sezione *Codice*.

2 Riferimenti

All'interno del documento vengono citate le seguenti informazioni.

Riferimento	Descrizione
IGES_Mark_CCR	Documento relativo alla comprensione del codice in cui sono stati identificati una serie di problemi relativi ai dettagli tecnici del sistema.

3 Introduzione

Di seguito sono riportate tutte le informazioni relative alla fase di reverse engineering.

Obiettivo del Documento L'obiettivo di questo documento è tenere traccia delle informazioni recuperate attraverso un processo di reverse engineering.

Contesto del Progetto Mark è un tool scritto in Python che dato un progetto, tramite l'ausilio di una Knowledge Base e delle detection rules, può classificare progetti ML in tre diverse categorie: ML-Consumer, ML-Producer e ML-Producer e Consumer.

4 Codice

Di seguito sono riportate le informazioni relative ad alcuni metodi riportati nel codice.

Nota. Ci si è voluto concentrare solo sui metodi considerati di maggiore interesse poiché meno chiari. In particolare ci si è soffermati solo sui moduli *producer_classifier_by dict.py* e *consumer_classifier_by dict.py*

producer_classifier_by dict.py

Nome Metodo	*__init__(self, output_folder="Producers/")*
Input	*output_folder*: stringa (percorso cartella)
Output	Nessuno (costruttore)
Funzione	Inizializza *self.output_folder* e crea la cartella chiamando *init_producer_analysis_folder()*.

Nome Metodo	*init_producer_analysis_folder(self)*
Input	Nessuno
Output	Nessuno (scrive file e crea cartelle)
Funzione	Crea la cartella e un file *results_first_step.csv* con colonne base oppure fa backup del file esistente.

Nome Metodo	*check_for_training_method(self, file, library_dict_path)*
Input	*file*, *library_dict_path*: stringhe
Output	Tupla: lista di librerie, lista di dizionari con keyword, lista vuota
Funzione	Legge il file, cerca keyword da librerie produttori usando regex e ritorna i risultati.

Nome Metodo	*analyze_single_file(self, file, repo, library_dict_path)*
Input	*file*, *repo*, *library_dict_path*: stringhe
Output	Tupla: *libraries*, *keywords*, *file_path*
Funzione	Chiama *check_for_training_method* e ritorna i risultati. Stampa messaggio se trova keyword.

Nome Metodo	*analyze_project_for_producers(self, repo_contents, project, dir, library_dict_path)*
Input	*repo_contents*, *project*, *dir*, *library_dict_path*: stringhe
Output	*pandas.DataFrame* con colonne: *ProjectName*, *Is ML producer*, *libraries*, *where*, *keywords*, *line_number*
Funzione	Analizza ricorsivamente file *.py* e *.ipynb*, raccoglie risultati e salva csv.

Nome Metodo	*analyze_projects_set_for_producers(self, input_folder, library_dict_path)*
Input	*input_folder*, *library_dict_path*: stringhe
Output	*pandas.DataFrame* aggregato dei risultati
Funzione	Per ogni progetto controlla se già analizzato; altrimenti chiama *analyze_project_for_producers* e aggiorna csv.

Nome Metodo	*load_producer_library_dict(input_file)* (static method)
Input	*input_file*: stringa
Output	*pandas.DataFrame*
Funzione	Metodo statico per caricare il dizionario delle librerie produttori.

Nome Metodo	*baseline_check(project, dir, df)*
Input	*project*, *dir*: stringhe; *df*: *DataFrame*
Output	booleano
Funzione	Verifica se progetto e sottocartella sono già presenti in *df*.

Nome Metodo	*build_regex_pattern(keyword)*
Input	*keyword*: stringa
Output	regex compilato (*re.Pattern*)
Funzione	Costruisce un pattern regex per keyword, con gestione spazi opzionali tra parole.

consumer_classifier_by dict.py

Nome Metodo	*__init__(self, output_folder="Consumers/")*
Input	*output_folder*: stringa, cartella dove salvare i risultati (default: "Consumers/")
Output	Nessuno (costruttore)
Funzione	Crea la cartella di output se non esiste e inizializza l'analisi chiamando *init_consumer_analysis_folder()*.

Nome Metodo	*check_ml_library_usage(self, file, library_dict)*
Input	*file*: stringa (percorso del file); *library_dict*: *pandas.DataFrame*
Output	Sottoinsieme di *library_dict* filtrato sulle librerie effettivamente presenti
Funzione	Estrae le librerie tramite *get_libraries*, normalizza i nomi e filtra il dizionario.

Nome Metodo	*init_consumer_analysis_folder(self)*
Input	Nessuno
Output	Nessuno
Funzione	Crea la sottocartella *Consumer_Analysis* e un file vuoto *results_consumer.csv*; se il file esiste, ne salva un backup.

Nome Metodo	*check_training_method(self, file, producer_library)*
Input	*file*: stringa (percorso); *producer_library*: stringa (path csv)
Output	Booleano (True o False)
Funzione	Verifica la presenza di metodi di training ML usando keyword definite nel dizionario producer.

Nome Metodo	*check_for_inference_method(self, file, consumer_library, producer_library, rules_3)*
Input	*file*: stringa; *consumer_library*, *producer_library*: path csv; *rules_3*: booleano
Output	Tupla: *(list_of_libraries, list_of_keywords_found, list_load_keywords)*
Funzione	Analizza il file per keyword di inference; se *rules_3* è attiva, esclude attività di training. Gestisce eccezioni.

Nome Metodo	*analyze_single_file(self, file, repo, consumer_library, producer_library, rules_3)*
Input	*file*, *repo*, *consumer_library*, *producer_library*: stringhe; *rules_3*: booleano
Output	Tupla: *(libraries, keywords, list_load_keywords, file)*
Funzione	Chiama *check_for_inference_method* per analizzare il file e stampa messaggi se trova keyword.

Nome Metodo	*analyze_project_for_consumers(self, repo_contents, project, in_dir, consumer_library, producer_library, rules_3, rules_4)*
Input	*repo_contents*, *project*, *in_dir*, *consumer_library*, *producer_library*: stringhe; *rules_3*, *rules_4*: booleani
Output	*pandas.DataFrame* con colonne: *['ProjectName', 'Is ML consumer', 'libraries', 'where', 'keywords', 'line_number']*
Funzione	Esplora ricorsivamente file *.py* e *.ipynb*, applica filtri, analizza ogni file con *analyze_single_file* e salva risultati in un *DataFrame*.

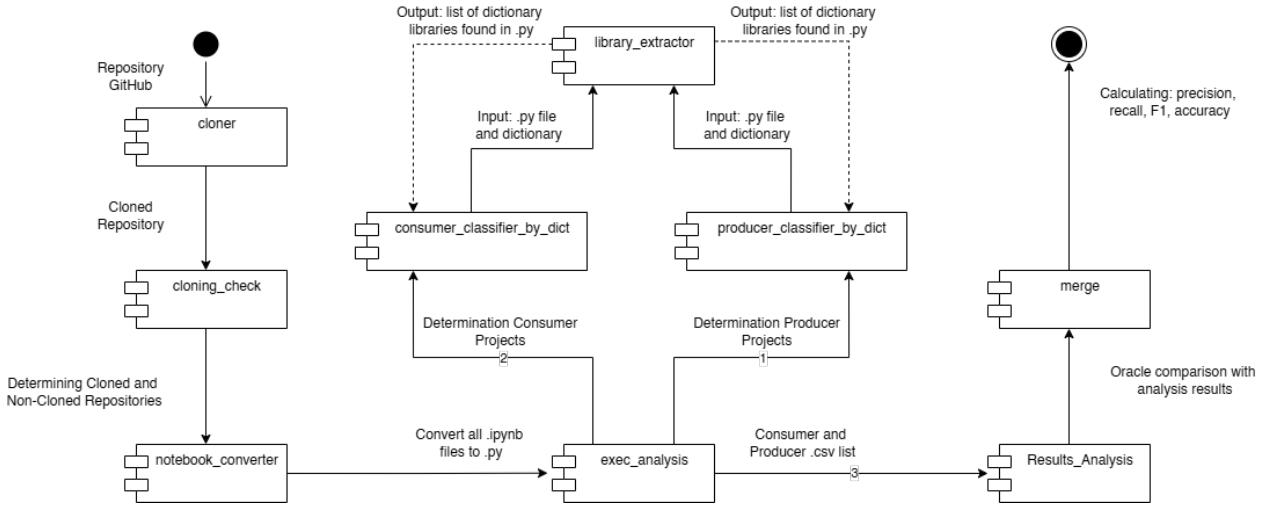
Nome Metodo	*analyze_projects_set_for_consumers(self, input_folder, consumer_library, producer_library, rules_3, rules_4)*
Input	*input_folder*, *consumer_library*, *producer_library*: stringhe; *rules_3*, *rules_4*: booleani
Output	*pandas.DataFrame* aggregato con tutti i risultati
Funzione	Carica *results_consumer.csv* esistente, analizza ogni progetto nel folder di input con *analyze_project_for_consumers* e aggiorna il csv.

5 Diagrammi Statici

Di seguito sono riportati tre paragrafi che descrivono una serie di diagramma statici che si è stati in grado di recuperare sulla base delle informazioni relative al codice.

Nota. Non si é ritenuto necessario per la comprensione del sistema la realizzazione dei sequence diagram, degli state chart e degli activity diagram data la natura procedurale del sistema e perché si é ritenuto che il sistema, attraverso il supporto dei diagrammi sotto descritti, fosse sufficientemente chiaro.

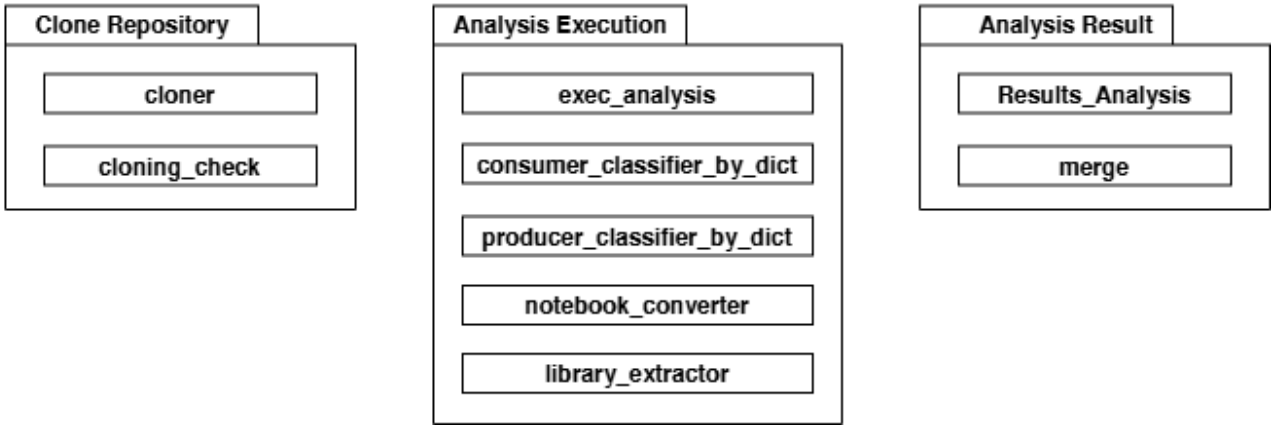
Pipeline Diagram Di seguito è riportato il **pipeline diagram**, con il quale si intende descrivere il flusso di azioni che vanno dalla raccolta dei dati di input fino all'elaborazione dei dati di output.



Nota. Ogni fase è già descritta all'interno del documento **IGES_Mark_CCR**.

Package Diagram Di seguito è riportato il **packages diagram** del sistema. È possibile organizzare il sistema in tre sottosistemi denominati **Clone Repository**, **Analysis Execution** e **Analysis Result**.

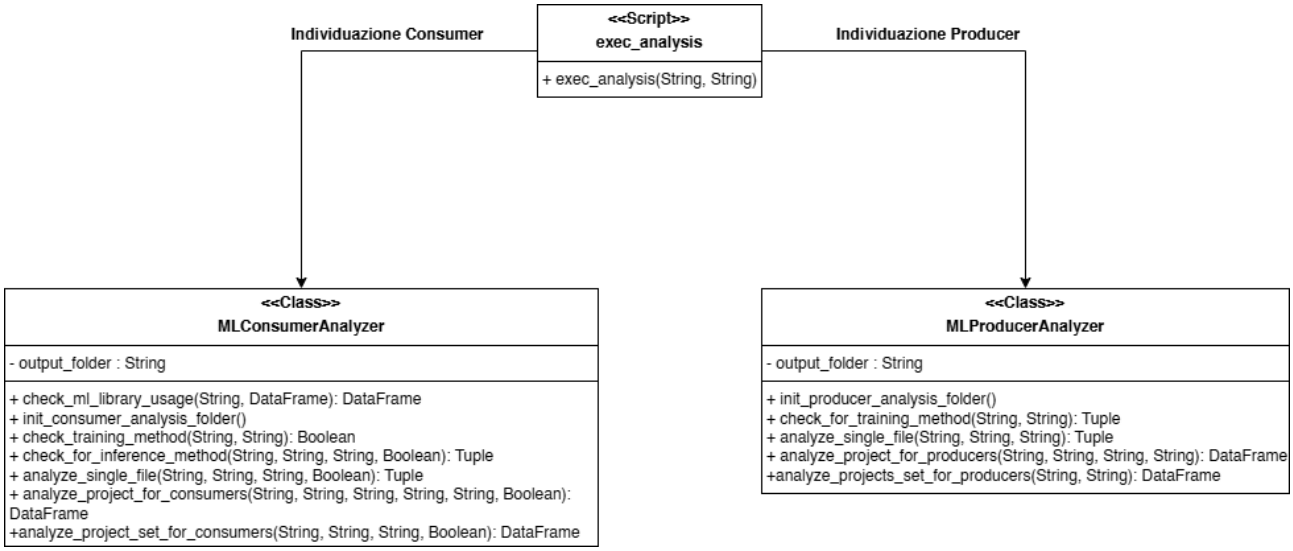
Il primo sottosistema si occupa della gestione delle repository, ovvero dei processi di acquisizione dei dati, clonazione in locale, controllo del processo di clonazione e dell'eventuale eliminazione di alcuni dati.



Il secondo sottosistema di occupa di eseguire l'analisi tramite classificatori dei dati raccolti dal primo sottosistema con l'obiettivo di classificare questi ultimi in ML consumers, ML producers o entrambi.

Il terzo sottosistema di occupa di eseguire l'analisi dei risultati ottenuti in termini di classificazione usufruendo di un oracolo e del calcolo di diverse metriche quali : la *F1-measure*, la *recall*, la *precision* e l'*accuracy*.

Class Diagram Siccome all'interno del sistema sono già presenti delle classi si è ritenuto ragionevole riportare le informazioni inerenti a queste ultime attraverso il seguente **class diagram**.



Nota. Lo script *exec_analysis.py* è un script che istanzia entrambe le classi *MLConsumerAnalyzer* e *MLProducerAnalyze*.

6 Requisiti Funzionali e User Stories

Di seguito sono elencati i requisiti funzionali del sistema definiti a partire dalle informazioni ottenute a livello di codice e diagrammi statici. Questi ultimi sono inoltre dettagliati attraverso delle user stories.

ID	Nome	Descrizione
RF_1	Clonazione Repository	L'utente deve poter indicare un insieme di repository da clonare inserendo le informazioni relative a nome utente e nome della repository per ognuna di esse all'interno di un file denominato selected_projects.csv strutturato allo stesso modo in cui è descritto nel documento **IGES_Mark_CCR**.
RF_2	Controllo Clonazione Repository	L'utente deve poter ispezionare due file .csv che tengono traccia delle repository effettivamente clonate e di quelle non clonate.
RF_3	Conversione File *.ipynb*	L'utente deve poter convertire i file con estensione *.ipynb* in file con estensione *.py*.
RF_4	Avvio Analisi	L'utente deve poter avviare un'analisi automatica delle repository clonate in grado di classificare tali repository in ML consumer, ML producer o entrambi e ispezionare due file .csv denominati nome_repo_ml_producer.csv e nome_repo_ml_consumer.csv strutturati allo stesso modo in cui sono descritti nel documento **IGES_Mark_CCR**.

ID	Descrizione
US_1	**Come** utente **voglio** dare in input il path di selected_projects.csv, dare in input il path della cartella in cui il sistema deve clonare le repository definite nel file .csv e avviare l'esecuzione **così che** possa ottenere una copia delle repository indicate.
US_2	**Come** utente **voglio** dare in input un file denominato applied_projects.csv strutturato allo stesso modo in cui è descritto nel documento **IGES_Mark_CCR**, dare in input il path della cartella in cui si trovano le repository copiate e avviare l'esecuzione **così che** possa controllare due file denominati not_cloned.csv e effective_repos.csv, strutturati allo stesso modo in cui sono descritti nel documento **IGES_Mark_CCR**, per capire quali repository sono state clonate e quali no.
US_3	**Come** utente **voglio** avviare l'esecuzione **così che** possa convertire il file con estensione *.ipynb* in file con estensione *.py*.
US_4	**Come** utente **voglio** dare in input il path della cartella dove si trovano le repository copiate, dare in input il path della cartella in cui voglio che siano create le cartelle *Producers_Final* e *Consumers_Final* **così che** possa ottenere i risultati delle analisi in due file *nome_repo*_ml_producer.csv e *nome_repo*_ml_consumer.csv nelle rispettive cartelle, strutturati allo stesso modo in cui sono descritti nel documento **IGES_Mark_CCR**.