Possono gli LLM Convertire uno Script Python di un Progetto di Machine Learning in una Recipe di MLflow?

# Introduzione

## Obiettivo dello Studio

Fare merge con: { **Obiettivo:** Introduci il contesto del tuo studio e definisci chiaramente il problema.

* **Motivazione:**
  + I chatbot basati su LLM (Large Language Models) come ChatGPT sono sempre più utilizzati per automatizzare e semplificare processi complessi. Esplora il loro ruolo nella transizione di progetti di machine learning verso standard strutturati come le *Recipes di MLflow*.
  + Perché è importante? Standardizzare il codice in una Recipe migliora la replicabilità, l'organizzazione e la collaborazione nei progetti ML.
* **Problema:**
  + I chatbot possono essere considerati strumenti affidabili per trasformare script Python in Recipes MLflow? Quali sono i limiti e le potenzialità?
* **Domande di ricerca:**
  + Qual è la precisione dei chatbot nel comprendere e convertire script complessi?
  + Quali tipi di errori o mancanze emergono durante la conversione?
  + Quali suggerimenti possono essere forniti per migliorare le prestazioni dei chatbot?

Descrivi l'obiettivo del tuo studio, ovvero valutare la capacità dei chatbot di convertire uno script Python di un progetto di machine learning in una recipe di MLflow.

## Importanza dello Studio

Da rivedere

**Obiettivo:** Fornire una base teorica.

* **Tecnologie Coinvolte:**
  + **Chatbot LLM:** Breve spiegazione di come funzionano, con un focus su GPT e modelli simili.
  + **MLflow Recipes:** Introduzione a MLflow e all'importanza delle Recipes per creare pipeline standardizzate.
* **Stato dell'Arte:**
  + Ricerche precedenti sull'uso di chatbot per la generazione di codice.
  + L'evoluzione degli strumenti per la gestione di pipeline ML.
* **Gap Identificato:** Mancanza di studi sull'applicazione specifica dei chatbot per convertire script Python in Recipes di MLflow.

## Problematiche

Illustra le problematiche che hanno spinto a lavorare sull'argomento. Ad esempio:

* **Complessità della conversione manuale**: La conversione manuale di script Python in recipe di MLflow può essere complessa e soggetta a errori.
* **Efficienza e riproducibilità**: La necessità di migliorare l'efficienza e la riproducibilità dei flussi di lavoro di machine learning.
* **Automazione**: L'importanza di automatizzare processi ripetitivi per liberare tempo per attività più creative e strategiche.

# Metodologia

## Strumenti Utilizzati

Elenca gli strumenti e le tecnologie che utilizzerai, come il chatbot (ad esempio, Microsoft Copilot), Python, MLflow, ecc.

## Script di Partenza

Descrivi lo script Python di machine learning che utilizzerai come base per la conversione.

## Criteri di Valutazione

* **Accuratezza**: La precisione con cui il chatbot converte lo script Python in una recipe di MLflow.
  + Si concretizza nell’analisi degli output e nel confronto con la soluzione “funzionante”.
* **Completezza**: Se il chatbot fornisce una conversione completa e funzionale.
  + sono presenti tutti gli artefatti di cui ho bisogno?
* **Numero di Errori/warnings:** importare il codice in un progetto e verificare staticamente il numero di errori rilevati dall’ide o a runtime in un tentativo di esecuzione.
* **Accettazione di File Python**: Se il chatbot è in grado di accettare file Python come input.

# Procedura

## Prompt Engineering

Il prompt engineering rappresenta una pratica essenziale per ottimizzare le risposte dei chatbot, mirata a elaborare e perfezionare i prompt al fine di ottenere risposte più accurate e utili. Questo studio esplora diverse tecniche di prompt engineering, evidenziando la loro importanza e applicazione.

**Metodologia:**

1. **Chiarezza e Specificità:** È fondamentale che i prompt siano chiari e specifici, evitando qualsiasi forma di ambiguità. Questo permette al chatbot di comprendere esattamente la richiesta e rispondere in modo pertinente.
2. **Contesto Dettagliato:** Fornire un contesto dettagliato è cruciale per aiutare il chatbot a comprendere meglio il compito richiesto. Un contesto ben definito guida il modello nella generazione di risposte più accurate.
3. **Esempi Concreti:** Includere esempi concreti all'interno dei prompt può illustrare meglio le richieste, fornendo al chatbot un riferimento chiaro su come strutturare la risposta.
4. **Suddivisione del Problema:** Dividere il problema in passi più piccoli e gestibili facilita il processo di risposta del chatbot, permettendo una gestione più efficiente e accurata delle richieste complesse.

## Prompt engineering utilizzando Chain-of-Thought (CoT) Prompting

L'obiettivo di questo studio è sviluppare una metodologia per incrementare progressivamente il livello di conoscenza di un modello linguistico di grandi dimensioni (LLM) attraverso una serie di interazioni strutturate. Il processo prevede l'introduzione graduale di informazioni specifiche sull'argomento di interesse, culminando con un prompt di "richiesta di traduzione". Questo studio si basa sulla tecnica di prompt engineering del **Chain of Thought**, che consente di migliorare la capacità del modello di ragionare e rispondere in modo coerente e dettagliato.

**Metodologia:**

1. **Introduzione Graduale delle Informazioni:**
   * Ogni fase della metodologia è progettata per fornire al chatbot informazioni incrementali.
   * In prima istanza si recuperano le informazioni sulle tecnologie abilitanti.
2. **Descrizione degli Step:**
   * Per ogni step, viene fornita una descrizione dettagliata di cosa il chatbot deve esaminare nel codice.
   * Viene incluso uno pseudocodice per chiarire come ogni step deve essere progettato.
3. **Richiesta di traduzione:**
   * Viene fornita al chatbot la richiesta di traduzione fornendo contestualmente il codice di cui fare l’elaborazione sotto forma di testo nel caso in cui non sia possibile fornire il file in formato “.py”.

## Prompt Iniziale con Contesto e Conoscenza

Definizione di un prompt per recuperare la conoscenza necessaria ad affrontare il task.

Bisogna nominare: Python, mlflow, recipe, jinja2 yaml, machine learning.

Possibile prompt:

"Sto affrontando un problema di traduzione di uno script Python in una recipe mlflow. Per risolvere questo problema, ho bisogno che tu acquisisca una descrizione dettagliata e delle applicazioni pratiche delle seguenti tecnologie nel contesto del machine learning. Per favore, recupera informazioni pertinenti e dettagliate per ciascun argomento:

1. **Python**:
2. **mlflow**:
3. **recipe**:
4. **jinja2 yaml**:
5. **machine learning**:

Queste tecnologie sono essenziali per affrontare il problema di traduzione dello script Python in una recipe mlflow.”

## Prompt di spiegazione della traduzione

Bisogna fornire all’LLM le informazioni necessarie a tradurre lo script in una recipe

Ad esempio, bisogna affrontare ogni step e spiegare all’llm cosa identificare nello script per dividerlo correttamente.

### Ingest

Possibile prompt:

Per individuare al meglio il codice da inserire nello step di ingest, devi individuare dove nella prima porzione dello script viene creato, recuperato e aggregato il dataset su cui avverrà l’addestramento della rete neurale.

Ti fornisco lo pseudocodice di esempio per lo step di ingest che potrai utilizzare per affrontare il problema di traduzione che ti fornirò in seguito. Ecco lo pseudocodice:

# Pseudocodice per lo Step di Ingest

## Definisci le routine necessarie per un passaggio specifico di un processo (ad esempio, 'ingest' di una ricetta di regressione)

1. Importa i moduli e le librerie necessarie, ad esempio:

- `logging` per la gestione dei log

- `DataFrame` dalla libreria `pandas` per la manipolazione dei dati

2. Configura il logger per registrare messaggi di log:

- Crea un'istanza del logger

3. Definisci una funzione per caricare i dati da un file in un formato specifico (ad esempio, CSV), o dalla fonte dati identificata dallo script che dovrai tradurre:

- Specifica i parametri della funzione (percorso del file, formato del file)

- Documenta la funzione con una docstring che descrive il suo scopo e i parametri

4. Implementa la logica per caricare i dati in base al formato del file:

- Se il formato del file è supportato (ad esempio, CSV):

- Importa la libreria necessaria (ad esempio, `pandas`)

- Registra un avviso se necessario

- Carica il file nel formato appropriato (ad esempio, `pandas.read\_csv`)

- Esegui eventuali trasformazioni sui dati (ad esempio, aggiungere colonne)

- Ritorna il DataFrame risultante

- Se il formato del file non è supportato:

- Solleva un'eccezione appropriata (ad esempio, `NotImplementedError`)

5. Assicurati che la funzione possa essere estesa o modificata facilmente per supportare nuovi formati di file o logiche di caricamento.

6. Crea lo step di ingest per la recipe di MLflow inserendo:

- Tipo del file di ingestione (es. CSV)

- Una location

- Il loader method

# Fine dello pseudocodice

---

Assicurati che la recipe di MLflow e lo script dello step siano due file separati. Di seguito, un esempio del file per la recipe di MLflow:

```yaml

# Esempio di file recipe di MLflow

recipe: "nomedellarecipe/v1"  
target\_col: "labeldellacolonnatarget"  
positive\_class: 1  
primary\_metric: "f1\_score"  
steps:  
 ingest: {{INGEST\_CONFIG}}

#Esempio di file local.yaml

experiment:  
 name: "nome\_experiment"  
 tracking\_uri: "sqlite:///metadata/mlflow/mlruns.db"  
 artifact\_location: "/metadata/mlflow/mlartifacts"  
  
model\_registry:  
 model\_name: "red\_wine\_classifier" #ad esempio

INGEST\_CONFIG:  
 using: csv  
 location: ["/data/winequality-white.csv", "/data/winequality-red.csv"]  
 loader\_method: nome\_del\_metodo

### Split

Possibile prompt:

Per individuare al meglio il codice da inserire nello step di split, devi individuare dove nella prima porzione dello script viene diviso o filtrato il dataset, recuperando anche i parametri di split per inserirli nel file local.yaml come SPLIT\_RATIOS.

Ti fornisco lo pseudocodice di esempio per lo step di split che potrai utilizzare per affrontare il problema di traduzione che ti fornirò in seguito. Questo pseudocodice ti sarà necessario se lo script fornito tratterà il filtraggio del dataset. Ecco lo pseudocodice:

# Definisci le routine necessarie per un passaggio specifico di un processo (ad esempio, 'split' di una ricetta di regressione)

1. Importa i moduli e le librerie necessarie, ad esempio:

- DataFrame e Series dalla libreria pandas per la manipolazione dei dati

2. Configura ad esempio il logger per registrare messaggi di log (opzionale):

- Crea un'istanza del logger

3. Definisci una funzione per filtrare i dati nei dataset di training, validation e test:

- Specifica i parametri della funzione (dataset)

- Documenta la funzione con una docstring che descrive il suo scopo e i parametri

4. Implementa la logica per filtrare i dati:

- Crea una Series booleana che indica se ogni riga deve essere filtrata

- Ritorna la Series risultante

5. Assicurati che la funzione possa essere estesa o modificata facilmente per supportare nuove logiche di filtraggio.

Inoltre, ecco il template corretto dello step di split della recipe di MLflow che puoi utilizzare:

# Template per lo step di split della recipe di MLflow

split:

split\_ratios: {{SPLIT\_RATIOS|default([0.75, 0.125, 0.125])}}

Infine, ecco la specifica del file local.yaml:

# Specifica del file local.yaml

SPLIT\_RATIOS: [0.80, 0.10, 0.10]

### Transform

Per individuare al meglio il codice da inserire nello step di trasformazione, devi individuare dove nella porzione centrale dello script vengono trasformati i dati. È importante recuperare anche i parametri di trasformazione per inserirli nel file local.yaml come TRANSFORM\_PARAMS.

Ti fornisco lo pseudocodice di esempio per lo step di trasformazione che potrai utilizzare per affrontare il problema di traduzione che ti fornirò in seguito. Questo pseudocodice ti sarà necessario se lo script fornito tratterà la trasformazione del dataset. Ecco lo pseudocodice:

1. **Definire una funzione chiamata transformer\_fn**:
   * Questa funzione deve restituire un trasformatore non addestrato.
2. **All'interno della funzione transformer\_fn**:
   * Inizializzare un oggetto trasformatore.
   * Definire un metodo chiamato fit che accetta dati di addestramento e addestra il trasformatore.
   * Definire un metodo chiamato transform che accetta dati di input e li trasforma secondo la logica definita.
3. **Assegnare i metodi fit e transform all'oggetto trasformatore**:
   * Assicurarsi che il trasformatore abbia i metodi fit e transform definiti.
4. **Restituire l'oggetto trasformatore non addestrato o None**:
   * La funzione transformer\_fn deve restituire il trasformatore con i metodi fit e transform pronti per essere utilizzati, oppure None se non è stato utilizzato nello script di partenza il trasformatore.

### Train

Per individuare al meglio il codice da inserire nello step di training, devi individuare dove nella porzione centrale dello script viene addestrato il modello.

Durante il training del modello, lo step di trasformazione è cruciale per garantire che i dati siano preparati correttamente. Ecco un esempio di pseudocodice:

1. **Descrizione del modulo**:
   * Questo modulo definisce una routine utilizzata nel passaggio di 'train' (allenamento) di un modello.
   * La routine principale è ad esempio estimator\_fn, che definisce il tipo di stimatore personalizzabile e i parametri utilizzati durante l'allenamento per produrre un modello.
2. **Importazioni**:
   * Viene importato ad esempio il tipo Dict e Any dal modulo typing per specificare i tipi di dati dei parametri.
3. **Definizione della funzione estimator\_fn**:
   * La funzione estimator\_fn accetta un parametro opzionale estimator\_params di tipo dizionario (Dict[str, Any]).
   * Se estimator\_params non viene fornito (è None), viene inizializzato come un dizionario vuoto.
4. **Importazione della classe SGDClassifier**:
   * All'interno della funzione, viene importata ad esempio la classe SGDClassifier dal modulo sklearn.linear\_model.
5. **Creazione e restituzione dello stimatore**:
   * La funzione crea ad esempio un'istanza di SGDClassifier utilizzando random\_state=42 per garantire la riproducibilità dei risultati.
   * Gli eventuali parametri aggiuntivi forniti in estimator\_params vengono passati al costruttore di SGDClassifier d’esempio.
   * La funzione restituisce lo stimatore non addestrato (unfitted), che ha i metodi fit() e predict() compatibili con gli stimatori di scikit-learn.

### Evaluate, Register, Ingest\_scoring, Predict

Prompt per il LLM:

A partire dallo script che ti fornirò in seguito dovrai prendere i parametri di valutazione da inserire come threshold nel file recipe.yaml insieme agli altri parametri di cui ti do informazioni dettagliate che vanno messi in step separati.

Threshold di validazione: Identifica le metriche utilizzate per valutare le prestazioni del modello (es. f1\_score, precision\_score, recall\_score) e identifica le soglie utilizzate per ciascuna metrica presente nello script che ti fornirò.

Permessi di registrazione: Specifica se un modello che non soddisfa le soglie può essere registrato comunque (imposta un valore booleano per allow\_non\_validated\_model) questo dovrà essere identificato nel codice se viene gestito altrimenti imposta un valore di default come ‘false’.

Configurazioni di ingest: Fornisci eventuali configurazioni per la sezione ingest\_scoring (es. percorso dei dati di input o formato). Nel caso sia presente una sezione di ingest scoring nello script che ti fornirò dovrai individuare i parametri e inserirli nel file local.yaml mentre nel file recipe.yaml è necessario inserire il riferimento all’interno di {{}} per referenziare i parametri

Configurazioni di predizione: Se nello script che ti fornirò sarà presente una sezione di predizione allora dovrà essere analizzata per estrapolare i parametri come il tipo di formato che si utilizza per i dati e la location dei dati, i parametri dovranno essere inseriti nel file local.yaml e referenziati all’interno di {{}} all’interno del recipe.yaml

### Prompt di traduzione

A partire dalle informazioni ottenute in precedenza prendi questo script python e realizza la traduzione in una recipe di MLflow, ricorda di separare script diversi per i vari step e realizzare i file recipe.yaml e local.yaml come indicato. Ricordati anche delle dipendenze di Jinja e degli output precedenti. Dallo script riuscirai a dare il valore corretto a "nomedellarecipe" tra classification e regression. Ecco lo script:

import mlflow

import mlflow.sklearn

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import f1\_score, precision\_score, recall\_score

from sklearn.model\_selection import train\_test\_split

# Configurazione di MLflow

mlflow.set\_tracking\_uri("sqlite:///metadata/mlflow/mlruns.db")

mlflow.set\_experiment("sklearn\_classification\_experiment")

# Funzione per l'ingestione dei dati

def ingest\_data(file\_paths):

# Carica i dati dei vini bianco e rosso con il delimitatore corretto

data\_white = pd.read\_csv(file\_paths[0], delimiter=';')

data\_red = pd.read\_csv(file\_paths[1], delimiter=';')

# Aggiungi la colonna is\_red (0 per bianco, 1 per rosso)

data\_white["is\_red"] = 0

data\_red["is\_red"] = 1

# Unisci i due dataset

data = pd.concat([data\_white, data\_red], ignore\_index=True)

return data

# Funzione per la trasformazione dei dati

def transform\_data(data):

# Esempio di trasformazioni personalizzate

data = data.dropna() # Rimuovi valori nulli

return data

# Funzione per l'addestramento del modello

def train\_model(X\_train, y\_train):

model = RandomForestClassifier()

model.fit(X\_train, y\_train)

return model

# Funzione per valutare il modello

def evaluate\_model(model, X\_test, y\_test):

y\_pred = model.predict(X\_test)

f1 = f1\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

return f1, precision, recall

# Ingestione dei dati

file\_paths = ["./data/winequality-white.csv", "./data/winequality-red.csv"]

data = ingest\_data(file\_paths)

# Separazione in caratteristiche e target

target\_col = "is\_red"

X = data.drop(columns=[target\_col])

y = data[target\_col]

# Divisione dei dati (usando le SPLIT\_RATIOS definite in local.yaml)

split\_ratios = [0.80, 0.10, 0.10]

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=1 - split\_ratios[0], random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp,

test\_size=split\_ratios[2] / (split\_ratios[1] + split\_ratios[2]),

random\_state=42)

# Trasformazione dei dati

X\_train = transform\_data(X\_train)

X\_val = transform\_data(X\_val)

X\_test = transform\_data(X\_test)

# Avvia un run di MLflow

with mlflow.start\_run():

# Addestramento del modello

model = train\_model(X\_train, y\_train)

# Log del modello con MLflow

mlflow.sklearn.log\_model(model, "model")

# Valutazione del modello

f1, precision, recall = evaluate\_model(model, X\_test, y\_test)

# Log dei parametri e delle metriche

mlflow.log\_param("model\_type", "RandomForest")

mlflow.log\_metric("f1\_score", f1)

mlflow.log\_metric("precision\_score", precision)

mlflow.log\_metric("recall\_score", recall)

# Output delle metriche

print("F1 Score:", f1)

print("Precision Score:", precision)

print("Recall Score:", recall)

print("Esperimento completato!")

Oppure utilizzando il linking del file .py

## Valutazione della Risposta:

* Analisi del Codice Generato: Esamina il codice fornito dal chatbot per verificarne l'accuratezza e la completezza. -> stesso un diff
* Verifica della Facilità di Implementazione: Valuta quanto sia facile implementare il codice generato nel contesto del tuo progetto. -> facciamo un diff per vedere cosa manca?

# Analisi dei Risultati

## Comparazione delle Risposte

Confronta le diverse risposte ottenute dal chatbot in base ai criteri di valutazione definiti. Utilizza la seguente tabella per registrare e confrontare i risultati:

| Chatbot | Accuratezza | Completezza | Facilità di Implementazione | Accettazione di File Python | Fornitura del Codice Risultante |
| --- | --- | --- | --- | --- | --- |
| Copilot |  |  |  | No | Si |
| ChatGPT |  |  |  | Si |  |
| CodeGpt |  |  |  | Si |  |
| Gemini |  |  |  | Ni |  |
| Claude |  |  |  |  |  |
| Mistral Ai |  |  |  | Si | Si |

## Discussione

Da rivedere con: { **Obiettivo:** Interpretare i risultati.

* **Potenzialità:**
  + I chatbot possono velocizzare la standardizzazione di progetti ML per pipeline strutturate.
  + Possono supportare sviluppatori meno esperti nella gestione di MLflow.
* **Limiti:**
  + Mancanza di comprensione dei dettagli specifici di alcune librerie Python.
  + Errori ricorrenti nei casi complessi.
  + Necessità di intervento umano per verificare e correggere i risultati.
* **Confronto con le Aspettative:**
  + Sono stati rispettati gli obiettivi iniziali?
  + Come si collocano i risultati rispetto allo stato dell'arte?

}

Questo studio analizza i punti di forza e di debolezza delle risposte generate dal chatbot. L'analisi si concentra su:

1. **Punti di Forza:**
   * **Accuratezza delle Risposte:** Le risposte del chatbot sono risultate più precise e pertinenti grazie ai prompt ottimizzati.
   * **Coerenza:** L'uso di prompt chiari e specifici ha migliorato la coerenza delle risposte, riducendo le ambiguità.
2. **Punti di Debolezza:**
   * **Limitazioni Contestuali:** In alcuni casi, il chatbot ha mostrato difficoltà nel comprendere contesti complessi nonostante i prompt dettagliati.
   * **Variabilità delle Risposte:** La variabilità nelle risposte può indicare una necessità di ulteriori perfezionamenti nei prompt.

# Conclusioni

Da rivedere con: {

**Obiettivo:** Riassumi le scoperte principali e proponi sviluppi futuri.

* **Risultati Principali:**
  + Livello di affidabilità raggiunto dai chatbot.
  + Principali lacune identificate.
* **Implicazioni:**
  + Impatto nell'automazione di processi MLOps.
  + Suggerimenti per migliorare i chatbot, come prompt migliori o fine-tuning specifico.
* **Lavori Futuri:**
  + Estendere lo studio con più script e chatbot.
  + Integrazione diretta di chatbot in piattaforme MLOps per supportare flussi di lavoro automatizzati.

}

## Sintesi dei Risultati

Riassumi i principali risultati del tuo studio.

## Raccomandazioni

Fornisci raccomandazioni su come migliorare ulteriormente l'uso dei chatbot per questo tipo di task.

## Prospettive Future

Suggerisci possibili direzioni per ricerche future.

# Appendici

## Codice e Prompt Utilizzati

Includi tutti i prompt e il codice generato durante lo studio.

## Risorse Aggiuntive

Fornisci link a risorse utili, come documentazione di MLflow e tutorial su Python.

## Stato dell'Arte

### Analisi delle ricerche precedenti sull'uso degli LLM nello sviluppo software

Le ricerche sull'uso degli LLM hanno dimostrato il loro potenziale nella generazione di codice, con esempi applicativi che spaziano dalla scrittura di funzioni a snippet di codice complessi. Tuttavia, pochi studi hanno affrontato l’applicazione specifica degli LLM per standardizzare pipeline ML, come nel caso delle MLflow Recipes. Esistono lacune nel confronto sistematico tra i diversi modelli e nell’analisi delle tecniche di prompt engineering.

### Tecnologie coinvolte

* **Python:** Linguaggio predominante per lo sviluppo di soluzioni ML grazie alla sua semplicità e ricca libreria di strumenti.
* **MLflow Recipes:** Framework per la creazione di pipeline ML standardizzate, che migliora la replicabilità e l'organizzazione dei progetti.
* **Prompt Engineering:** Tecniche per ottimizzare la qualità delle risposte degli LLM attraverso la formulazione di prompt specifici e dettagliati.

## Metodologia

### Dettagli sui dati raccolti

Il dataset utilizzato include risposte generate da vari LLM (Copilot, ChatGPT, Gemini, ecc.) su task di conversione di script Python in MLflow Recipes. Le risposte sono state valutate su:

* Accuratezza: Misura della correttezza rispetto ai requisiti.
* Completezza: Percentuale di artefatti generati rispetto a quelli richiesti.
* Errori: Numero e tipo di problemi riscontrati (sintassi, runtime, logica).

### Tecniche di analisi

* **Differenziazione tra modelli:** Confronto delle performance dei modelli per accuratezza, completezza ed errori.
* **Analisi evolutiva:** Studio delle variazioni nelle risposte con aggiornamenti del modello o prompt ottimizzati.
* **Esperimenti controllati:** Task di complessità crescente per valutare robustezza e scalabilità delle risposte.

## Risultati

### Confronto tra LLM

I risultati mostrano una variazione significativa nelle performance:

1. **Accuratezza:** Varia dal 66% all’85%, con migliori performance di Mistral AI nei task complessi.
2. **Completezza:** Alcuni modelli, come ChatGPT, producono artefatti quasi completi, mentre altri, come Gemini, faticano nei task più articolati.
3. **Errori:** La maggior parte degli errori riguarda la logica del codice e l’omissione di artefatti essenziali.

### Impatto del tempo e del prompt engineering

L’uso di tecniche come Chain-of-Thought ha migliorato l’accuratezza delle risposte del 15-20%. Inoltre, aggiornamenti regolari dei modelli hanno portato a miglioramenti progressivi.

## Discussione

### Limiti e opportunità

**Limiti:**

1. **Mancanza di comprensione di librerie specifiche:** Gli LLM spesso non gestiscono correttamente funzionalità avanzate o casi limite relativi a librerie come MLflow, pandas o scikit-learn, con conseguente necessità di intervento umano.
2. **Intervento umano:** Anche nelle migliori condizioni, è necessario verificare e correggere i risultati, il che limita l’automazione completa del processo.

**Opportunità:**

### Velocizzazione dei workflow ML

1. L'adozione di LLM per automatizzare la standardizzazione delle pipeline ML con strumenti come MLflow consente di:

### Ridurre il tempo di sviluppo: I chatbot generano configurazioni complesse, come file YAML e codice per pipeline, in pochi secondi, eliminando operazioni ripetitive.

### Minimizzare gli errori manuali: La standardizzazione automatica riduce la possibilità di omissioni o errori nel setup delle pipeline, migliorando la replicabilità dei risultati.

### Accelerare l'onboarding: Per team nuovi o junior, gli LLM offrono un punto di partenza chiaro e funzionale per costruire pipeline, velocizzando l'adattamento al flusso di lavoro esistente.

### Supporto a sviluppatori meno esperti

1. Gli LLM possono fungere da strumenti educativi per gli sviluppatori meno esperti grazie a:

### Esempi contestualizzati: I chatbot generano codice accompagnato da spiegazioni dettagliate, facilitando la comprensione di framework come MLflow, scikit-learn, e pandas.

### Suggerimenti guidati: Gli LLM possono suggerire librerie, configurazioni o pratiche ottimali per affrontare specifici task, riducendo la curva di apprendimento.

### Debug assistito: Quando uno sviluppatore fornisce input errati o incompleti, i modelli possono identificare errori e proporre correzioni, migliorando l'autoapprendimento.

### Ampliamento delle funzionalità

1. Gli LLM possono evolversi per offrire:

### Supporto a librerie avanzate: Integrare conoscenze approfondite di librerie come TensorFlow, PyTorch e librerie di analisi dei dati, migliorando le capacità di generare codice per pipeline complesse.

### Suggerimenti contestuali: Basandosi sul codice fornito, gli LLM possono proporre miglioramenti o ottimizzazioni, ad esempio nel preprocessing dei dati o nella scelta di modelli ML.

### Ottimizzazioni specifiche: Attraverso il fine-tuning su task MLOps, i modelli potrebbero generare configurazioni altamente specifiche per ambienti eterogenei, come cluster Kubernetes o ambienti serverless.

### Raccomandazioni per migliorare i processi MLOps con LLM

1. **Migliorare il Prompt Engineering**
2. L'ottimizzazione del prompt engineering è fondamentale per aumentare la precisione e l'affidabilità delle risposte degli LLM. Alcune strategie includono:
3. **Sviluppo di template predefiniti:**
   1. **Template generici:** Definire prompt standardizzati per task comuni, come la creazione di file recipe.yaml o local.yaml, riduce la variabilità nelle risposte.
      1. *Esempio:*
4. yaml
5. Copia codice
6. Genera una MLflow Recipe con i seguenti parametri:  
   - Nome esperimento: "nome\_experiment"  
   - Target: "colonna\_target"  
   - Steps: [ingest, split, transform, train, evaluate]  
   1. **Prompt modulari:** Strutturare i prompt in modo gerarchico, con richieste chiare per ogni step. Ad esempio, un prompt per la configurazione dell'ingest e uno separato per lo split.
7. **Chain-of-Thought (CoT):**
   1. Usare tecniche che guidano il modello a pensare passo dopo passo. Questo approccio consente risposte più coerenti e dettagliate.
      1. *Esempio:*
8. markdown
9. Copia codice
10. Dividi l’analisi in passi:  
    1. Identifica i dati da caricare.  
    2. Specifica il formato e il metodo di caricamento.  
    3. Configura il logger per tracciare gli eventi.
11. **Prompt contestualizzati:** Fornire contesto dettagliato sui requisiti del progetto e sull'ambiente operativo.
    1. *Esempio:*
12. go
13. Copia codice
14. Stiamo lavorando su un progetto ML in cui il dataset è salvato in formato CSV e deve essere utilizzato per allenare un modello di classificazione con MLflow. Genera il file `recipe.yaml`.

### 2. Integrare Validatori Automatici

1. Automatizzare la verifica degli artefatti generati dagli LLM garantisce maggiore affidabilità e riduce l'intervento umano. Le principali proposte includono:
2. **Test unitari automatici:**
   1. Implementare suite di test per verificare la correttezza del codice generato.
      1. *Esempio:* Test per validare che il file YAML prodotto abbia una struttura corretta o che il codice Python importi correttamente le librerie richieste.
   2. Utilizzare framework come pytest per testare la sintassi, la logica e l'integrazione del codice.
3. **Validazione sintattica:**
   1. Strumenti come yamllint o jsonlint possono essere integrati per verificare automaticamente i file YAML generati.
   2. Per il codice Python, l’uso di flake8 o pylint può garantire aderenza agli standard di codifica.
4. **Validazione runtime:**
   1. Implementare script per eseguire pipeline simulate, verificando che ogni step prodotto (ingest, split, ecc.) sia funzionante.
   2. Utilizzare container Docker per creare ambienti isolati e testare l’intera pipeline in modo automatizzato.
5. **Reportistica automatizzata:**
   1. Generare report dettagliati che evidenzino errori o parti mancanti negli artefatti prodotti, fornendo un feedback chiaro per ulteriori miglioramenti.

### 3. Fine-Tuning dei Modelli

1. Adattare gli LLM ai domini specifici come MLOps migliora la comprensione di librerie e flussi di lavoro. Questo processo include:
2. **Raccolta di dataset specifici:**
   1. Creare dataset di training con esempi di configurazioni MLflow, script Python per pipeline ML, e file YAML.
   2. *Esempio:* Dataset con migliaia di esempi di configurazioni recipe.yaml per task di classificazione e regressione.
3. **Adattamento del modello:**
   1. Utilizzare tecniche di fine-tuning come quelle offerte da framework come Hugging Face o OpenAI API.
   2. *Esempio:* Addestrare il modello su task specifici come:
      1. Generazione di pipeline end-to-end.
      2. Configurazione di ambienti di sviluppo MLOps con Kubernetes o Docker.
4. **Introduzione di conoscenze specifiche:**
   1. Integrare nei dataset informazioni avanzate su librerie come MLflow, TensorFlow, scikit-learn, o PyTorch.
   2. *Esempio:* Insegnare al modello le best practice per il logging delle metriche di performance dei modelli.
5. **Validazione del modello fine-tuned:**
   1. Testare il modello aggiornato su task reali per verificare miglioramenti rispetto alla versione generica.
   2. Confrontare le risposte del modello fine-tuned con quelle di versioni predefinite.

### Impatto Previsto

* **Maggiore efficienza:** Le risposte degli LLM diventano più mirate e precise, riducendo il tempo di revisione.
* **Riduzione degli errori:** L’automazione nella verifica e il fine-tuning migliorano l’accuratezza degli artefatti generati.
* **Adozione più ampia:** Migliorare i processi e l’affidabilità degli LLM li rende strumenti più attraenti per i team di sviluppo MLOps.

## Conclusioni e Lavori Futuri

### Sintesi dei risultati principali

Gli LLM hanno dimostrato un potenziale significativo nel supportare lo sviluppo di pipeline ML standardizzate, ma presentano ancora limitazioni nella comprensione del contesto e nella generazione completa degli artefatti richiesti. L’uso di tecniche di prompt engineering e aggiornamenti continui dei modelli ha portato a miglioramenti sostanziali nella qualità delle risposte.

### Prospettive di integrazione degli LLM nei flussi di lavoro reali

1. **Automazione delle pipeline:** L’integrazione di LLM come assistenti interattivi per generare, validare e migliorare pipeline ML potrebbe trasformare i processi di sviluppo.
2. **Ampliamento dello studio:** Estendere l’analisi ad altri contesti di sviluppo software, come DevOps o gestione dei dati.
3. **Collaborazione uomo-macchina:** Progettare workflow che combinino le capacità degli LLM con l’expertise umana per ridurre errori e migliorare l’efficienza.