Analisi Dettagliata delle Correlazioni tra Teoria e Implementazione in Ambito AI

Slide 1: Introduzione

- **Titolo**: Correlazioni tra Teoria AI e Implementazione Pratica
- **Sottotitolo**: Analisi dei collegamenti tra modelli teorici (LLM, RAG, Modelli Predittivi) e la loro applicazione nel codice

Slide 2: Architettura del Sistema - Visione d'Insieme

- **Sistema implementato**: Framework di previsione e analisi della qualità dell'aria con interfaccia conversazionale
- **Componenti principali**:
  - Modello predittivo temporale (Prophet)
  - Integrazione con LLM (Large Language Model) via Groq API
  - Sistema di recupero dati e contestualizzazione (elementi RAG)
  - Interfaccia conversazionale con caratteristiche di AI Agent

Slide 3: Large Language Models (LLM) - Implementazione e Teoria

- **Implementazione nel codice**:

```python
data = {
    "model": "llama3-70b-8192",
    "messages": [{"role": "user", "content": prompt}],
    "temperature": 0.7
}
```

- **Correlazione con teoria (smid-mpa-2024-5.pdf)**:
  - Il codice implementa quanto descritto a pg. 11: "*LLM è un tipo di rete neurale chiamata transformer, progettata per elaborare e generare dati in sequenza, come il testo*"
  - Uso del parametro di temperatura (0.7) per controllare la creatività del modello, come discusso nella teoria dei trasformer
  - Dimensione del modello (70B) riflette il concetto di "*centinaia di miliardi di connessioni tra milioni di neuroni*" menzionato a pg. 13

Slide 4: Architettura LLM e Flusso di Elaborazione

- **Implementazione**:

1. Scomposizione dell'input in token
2. Elaborazione del prompt
3. Generazione della risposta token per token

- **Correlazione con teoria**:
  - Riflette il processo descritto a pg. 14-15: "*Per ogni parola (o parte di parola) fornita come input al modello e per ogni parola generata come output*"
  - Conversione da input testuale a token e viceversa
  - Generazione sequenziale che "*suona giusta*" (pg. 16), non basata su ricerca di fatti ma su pattern appresi

Slide 5: Prompting Sofisticato - La Chiave dei LLM

- **Implementazione nel codice**:

```python
prompt = (
    f"Nella regione {region_clean}, l'inquinante {pollutant_name} è {trend} dal {first_year} "
    f"({first_value:.2f} µg/m³) al {last_year} ({last_value:.2f} µg/m³), con un cambiamento "
    f"del {percent:.1f}%. {plausibility} Scrivi un testo su questo andamento in italiano, "
    "parlando dell'impatto sulla salute, sull'ambiente e su cosa potrebbe causare questa evoluz
    "Basandoti solo sui dati forniti dal dataset e da ciò che vedi sul grafico."
)
```

- **Correlazione con teoria**:
  - Applicazione pratica del concetto di "attenzione" dei transformer (pg. 10): i dati numerici forniti dirigono l'attenzione del modello
  - Uso di prompt strutturati che includono:
    - Dati concreti (valori numerici)
    - Contesto (regione, inquinante)
    - Istruzioni specifiche sul formato di output
    - Limitazioni ("*basandoti solo sui dati forniti*")
  - Esempio di come guidare l'output del LLM attraverso input ben progettati

Slide 6: Retrieval Augmented Generation (RAG) - Fondamenti Teorici

- **Teoria da smid-mpa-2024-11.pdf**:
  - "*RAG è un metodo che combina il potere di retriever su larga scala e modelli generativi sequence-to-sequence*" (pg. 1)

- *"Un processo in due fasi: 1) Recupero: quando viene presentata una query, il sistema RAG recupera prima passaggi o documenti rilevanti da un ampio corpus (database) che potrebbero contenere la risposta. 2) Generazione: questi passaggi vengono quindi forniti come contesto a un modello sequence-to-sequence, che genera una risposta coerente e contestualmente appropriata*" (pg. 1)

Slide 7: Implementazione RAG nel Sistema

- **Nel codice**:

```python
# FASE 1: RETRIEVAL - Recupero dati rilevanti
def find_region(query, df):
    """Find the region based on the user's query."""
    query_lower = query.lower()
    # [...codice di recupero...]
    return matched_region, "match diretto"

# FASE 2: ANALISI e PROCESSING dei dati
def train_and_forecast(df_region, pollutant, pollutant_name, region_name):
    # [...processing con Prophet...]

# FASE 3: GENERATION - Generazione basata sui dati recuperati
explanation = generate_ai_explanation(region_name, pollutant_name, first_year, first_value, las
```

- **Correlazione con teoria RAG**:
  - Implementazione completa del ciclo RAG:
    1. Recupero contestuale (find_region)
    2. Elaborazione dei dati recuperati (train_and_forecast)
    3. Generazione di output informato dal contesto (generate_ai_explanation)
  - Esempio concreto del vantaggio RAG citato a pg. 3: *"Accuratezza migliorata: recuperando informazioni rilevanti da un database, RAG può fornire risposte più accurate, specialmente in compiti di recupero di informazioni o risposta a domande*"

Slide 8: Vantaggi RAG - Teoria e Implementazione

- **Teoria (smid-mpa-2024-11.pdf, pg. 4)**:
  - *"Coerenza contestuale: la fase di recupero aiuta a garantire che l'output generato sia contestualmente rilevante e coerente con l'input fornito e il contesto recuperato*"
  - *"Miglioramento dell'accuratezza: recuperando informazioni rilevanti da un database, RAG può fornire risposte più accurate*"
- **Implementazione**:

python

```python
def process_region(df, region_name):
    df_region = df[df['WHO Region'] == region_name]
    for key, col_name in POLLUTANTS.items():
        if col_name in df_region.columns:
            print(generate_ai_chat_response("processing_pollutant", pollutant=col_name))
            path, explanation = train_and_forecast(df_region, col_name, col_name, region_name)
```

- **Correlazione**:
  - Il sistema limita le risposte ai dati effettivamente disponibili
  - Verifica che l'inquinante richiesto esista nei dati della regione
  - Genera spiegazioni contestualizzate basate su dati reali, non su conoscenze generiche

Slide 9: Modelli Predittivi - Dal Generativo al Forecast

- **Implementazione di Prophet**:

python

```python
model = Prophet(yearly_seasonality=True, seasonality_mode='additive')
model.fit(df_model)
future = model.make_future_dataframe(periods=FORECAST_YEARS, freq='YE')
forecast = model.predict(future)
```

- **Correlazione con teoria dei modelli generativi**:
  - Collegamento con il concetto di "modello generativo" descritto in smid-mpa-2024-7.pdf (pg. 1-2):
    - "*I modelli generativi possono generare nuove istanze di dati*"
    - "*Un modello generativo cattura la probabilità congiunta P(X,Y) o solo P(X) se non ci sono etichette*"
  - Prophet è un modello generativo che crea nuove serie temporali basate su pattern appresi dai dati storici
  - Analogia con GAN: "*L'obiettivo di un GAN è trasformare il rumore in dati indistinguibili dai dati di addestramento*" (pg. 7) - Prophet trasforma i dati storici in previsioni future plausibili

Slide 10: Architettura dei Modelli Generativi - Confronto Teorico

- **Teoria (smid-mpa-2024-7.pdf)**:
  - "*I modelli generativi affrontano un compito più difficile rispetto ai modelli discriminativi analoghi. I modelli generativi devono modellare di più*" (pg. 2)
  - "*I modelli generativi cercano di modellare come i dati sono collocati in tutto lo spazio*" (pg. 2)

- **Implementazione in Prophet**:
  - Modellazione della stagionalità temporale: `yearly_seasonality=True`
  - Cattura la distribuzione dei dati nel tempo anziché solo la classificazione
  - Generazione di intervalli di confidenza attorno alle previsioni:

python

```
ax.fill_between(forecast_future['ds'], forecast_future['yhat_lower'], forecast_future['yhat_upp
                color='pink', alpha=0.3, label='Intervallo confidenza')
```

Slide 11: AI Agent - Definizione e Caratteristiche

- **Teoria (smid-mpa-2024-11.pdf, pg. 21-23)**:
  - "*Gli agenti AI, a differenza dei modelli LLM e RAG, sono costruiti appositamente per l'interazione con il mondo reale. Questi agenti possiedono la capacità di percepire il loro ambiente, prendere decisioni ed eseguire azioni autonomamente o semi-autonomamente*" (pg. 21)
  - "*Interazione in tempo reale: gli agenti AI tipicamente operano in scenari in tempo reale o quasi in tempo reale, interagendo con gli utenti o i sistemi in modo dinamico*" (pg. 23)

Slide 12: Implementazione di Caratteristiche Agent-like

- **Nel codice**:

python

```
def chat_loop():
    print(generate_ai_chat_response("greeting"))
    df = load_data(INPUT_FILE)
    # [...]
    while True:
        query = input("🗣️ Tu: ").strip()
        if generate_ai_chat_response("exit_intent", user_message=query):
            print(generate_ai_chat_response("goodbye"))
            break
        matched_region = find_region(query, df)
        # [...esecuzione di azioni basate sull'input...]
```

- **Correlazione con teoria degli agenti**:
  - Il sistema implementa un ciclo di percezione-decisione-azione:
    1. Percezione: analisi dell'input dell'utente
    2. Decisione: identificazione dell'intento (exit_intent) e della regione
    3. Azione: recupero dati, addestramento modello, generazione risposta

- Interattività in tempo reale come descritto nella teoria
- Autonomia nelle decisioni basate sull'input degli utenti

Slide 13: Limitazioni dei LLM - Teoria e Mitigazione

- **Teoria (smid-mpa-2024-5.pdf, pg. 18)**:
  - "*Gli LLM producono testo che suona giusto ma non possono garantire che sia giusto*"
  - "*ChatGPT non può fornire alcuna garanzia che il suo output sia corretto, solo che suoni corretto*"
- **Mitigazione implementata**:

```python
plausibility = ""
if last_value < 5:
    plausibility = ("Tuttavia, questo valore molto basso nel futuro potrebbe essere "
                    "ottimistico: raggiungerlo richiederebbe politiche molto rigorose e duratur
```

- **Correlazione**:
  - Il codice anticipa potenziali inesattezze nelle previsioni molto ottimistiche
  - Aggiunge avvertenze nei prompt per guidare il LLM verso output più cauti
  - Implementa la cautela nella generazione di contenuto speculativo

Slide 14: Future Direzioni - Integrazione Avanzata

- **Teoria (smid-mpa-2024-5.pdf, pg. 19)**:
  - "*Nel prossimo futuro, è probabile che gli LLM saranno integrati in sistemi che combinano la potenza della generazione di testo dell'LLM con un motore computazionale o una base di conoscenza per fornire risposte fattuali in testo naturale convincente*"
- **Potenziali miglioramenti del sistema attuale**:
  - Integrazione di database vettoriali (come discusso in smid-mpa-2024-11.pdf, pg. 13-16)
  - Implementazione di meccanismi di feedback per migliorare l'accuratezza (RLHF)
  - Aggiunta di retriever specializzati per le ricerche sulla qualità dell'aria

Slide 15: Conclusioni

- **Sintesi delle correlazioni teoria-pratica**:
  1. LLM come generatori di testo basati su transformer
  2. Sistema RAG completo con retrieval, processing e generation
  3. Modelli predittivi generativi per previsioni temporali
  4. Funzionalità Agent-like per interazione in tempo reale

5. Consapevolezza e mitigazione delle limitazioni dei LLM

- **Valore didattico**:
  - Il codice fornisce un caso di studio concreto dell'applicazione dei concetti teorici
  - Dimostra l'integrazione di diverse tecnologie AI in un unico sistema funzionale
  - Illustra i compromessi tra accuratezza, efficienza e usabilità