

**Università della Calabria**  
Dipartimento di Matematica e Informatica

---



**Corso di Laurea Magistrale in Informatica**

Tesi di Laurea

# **L'uso di LLM in scenari di data migration and integration**

Relatori:

Ch.mo Prof. Giorgio Terracina

Dott. Sebastiano Antonio Piccolo

Candidato:

Daniele Avolio

Matricola 242423

---

Anno Accademico 2024/2025

*Perhaps we now understand that not everything has to have an answer.*



# Indice

<b>1</b>	<b>Introduzione</b>	<b>4</b>
1.1	Contesto e motivazioni . . . . .	4
1.2	Attività svolte e risultati ottenuti . . . . .	5
1.3	Principali contributi . . . . .	6
1.4	Problemi noti e limiti . . . . .	7
1.5	Descrizione del contenuto della tesi . . . . .	8
<b>2</b>	<b>Tecnologie e strumenti</b>	<b>9</b>
2.1	TIBCO . . . . .	9
2.1.1	TIBCO ActiveMatrix BusinessWorks . . . . .	10
2.1.2	XSLT (eXtensible Stylesheet Language Transformations) . . . . .	10
2.2	MuleSoft . . . . .	11
2.2.1	DataWeave . . . . .	13
2.3	Large Language Models (LLM) . . . . .	14
2.3.1	Architettura dei LLM . . . . .	15
2.3.2	Punti di forza e debolezza . . . . .	17
2.4	Spectrum . . . . .	18
2.4.1	Regole di Trasformazione . . . . .	20
2.4.2	Post-Processing . . . . .	20
2.5	Streamlit . . . . .	21
2.6	OpenAI API . . . . .	22
2.6.1	Utilizzo per gpt-4o-mini . . . . .	22
2.7	Chatbot Arena (ex. LMSYS) . . . . .	23
<b>3</b>	<b>Implementazione e metodologie</b>	<b>24</b>
3.1	Estensione grafica . . . . .	24
3.2	Necessità di un'intelligenza artificiale . . . . .	29
3.3	Conversione tramite Large Language Model . . . . .	31
3.4	Approccio basato su fine-tuning di modelli pre-addestrati . . . . .	47
3.4.1	Analisi Critica dello Studio Salesforce . . . . .	47

3.4.2	Considerazioni . . . . .	49
3.5	LLM per Micro Task . . . . .	51
3.5.1	Implementazione . . . . .	52
3.5.2	Codice e esempi . . . . .	53
3.5.3	Alternativa con LLM locale . . . . .	58
3.5.4	Considerazioni . . . . .	59
3.6	LLM per Macro-Task . . . . .	59
3.6.1	Complessità del Task e Limiti del Modello . . . . .	60
3.6.2	Analisi Critica . . . . .	61
3.6.3	Considerazioni . . . . .	61
3.6.4	Soluzione algoritmica . . . . .	61
3.7	Chatbot Integrato . . . . .	63
3.7.1	Scopo del Chatbot . . . . .	64
3.7.2	Piattaforma e Tecnologia . . . . .	64
3.7.3	Limiti e Sfide . . . . .	65
3.7.4	Esempi di Utilizzo . . . . .	66
3.7.5	Esempio di task complesso . . . . .	67
3.7.6	Considerazioni . . . . .	70
<b>4</b>	<b>Risultati e Conclusioni</b>	<b>71</b>
4.1	Riepilogo del lavoro . . . . .	71
4.2	Fattori limitanti . . . . .	72
4.3	Sviluppi futuri . . . . .	74
4.4	Un'alternativa potenzialmente promettente: il RAG . . . . .	75
4.5	Conclusioni . . . . .	76

# Capitolo 1

## Introduzione

### 1.1 Contesto e motivazioni

Negli ultimi anni, il settore dell'integrazione di dati ha vissuto una trasformazione radicale, alimentata dalla crescente complessità dei sistemi informatici e dalla necessità di garantire la comunicazione tra applicazioni aziendali. Con l'avvento della digitalizzazione e l'espansione del cloud computing, le aziende si trovano sempre più spesso a dover migrare da una piattaforma tecnologica a un'altra, affrontando sfide legate alla compatibilità, alla sicurezza e all'efficienza dei processi. Una delle sfide principali è quella di passare da sistemi legacy a sistemi più moderni e performanti, richiedendo una conoscenza approfondita di entrambe le tecnologie.

In questo contesto, due delle soluzioni più diffuse per l'integrazione applicativa sono TIBCO e MuleSoft. TIBCO è storicamente riconosciuta per la sua robustezza e affidabilità, mentre MuleSoft si è affermata come una piattaforma innovativa, caratterizzata da un approccio cloud e dalla facilità d'uso in contesti moderni. La migrazione tra queste due tecnologie non è solo una questione tecnica, ma anche strategica, con implicazioni significative per l'efficienza operativa e l'adattabilità delle aziende ai cambiamenti del mercato.

Il problema irrisolto che questa tesi affronta è la mancanza di strumenti automatizzati efficienti per supportare la migrazione da TIBCO a MuleSoft. In particolare, il processo di conversione del codice XSLT (utilizzato da TIBCO) in DataWeave (utilizzato da MuleSoft) è spesso manuale, ripetitivo e soggetto a errori umani. Questo deficit rappresenta un ostacolo significativo per le aziende che cercano di modernizzare i propri sistemi legacy senza compromettere l'efficienza e la qualità del codice risultante.

Nella fattispecie, questa tesi parte da problemi reali – presenti in un con-

testo aziendale tipico della piccola e media impresa – e analizza l’applicabilità dei moderni strumenti di intelligenza artificiale basati sui large language model (LLM) per facilitare le attività di data migration ed integration. Questa tesi è basata sui seguenti casi di studio:

1. Il miglioramento di un software che automatizzi la conversione di risorse da TIBCO a MuleSoft, garantendo coerenza e precisione, sviluppato durante il periodo di tirocinio.
2. L’applicazione di modelli di intelligenza artificiale, i Large Language Models (LLM), per supportare l’automazione e migliorare la qualità delle migrazioni in alcuni contesti.
3. Lo studio e la re-ingegnerizzazione delle risorse per sfruttare al meglio le potenzialità dei LLM, valutando benefici e rischi.

L’obiettivo complessivo è quello di ridurre il più possibile il lavoro a carico del programmatore per quelle task macchinose e spesso ripetitive che non necessitano di abilità avanzate. Questo tipo di attività, se eseguito manualmente da personale esperto, produce uno spreco di tempo notevole degli sviluppatori ed un conseguente spreco di risorse economiche. Nella presente tesi verranno esplorati vari approcci per automatizzare questa tipologia di task, confrontando i risultati ottenuti con e senza l’utilizzo degli LLM, valutando l’efficacia di questi strumenti e tenendo conto del loro rapporto costi/benefici in scenari di utilizzo su larga scala.

## 1.2 Attività svolte e risultati ottenuti

Nel corso di questa tesi, durante il periodo di *tirocinio* svolto presso **Lutech S.p.A.**, sono state intraprese diverse attività per rispondere agli obiettivi prefissati. Il progetto ha seguito un approccio pratico e teorico, articolato in:

1. Analisi delle tecnologie TIBCO e MuleSoft, identificandone le principali caratteristiche, differenze e problematiche, approfondendo quali fossero le principali difficoltà riscontrate dagli sviluppatori durante la fase di migrazione.
2. L’estensione del software Spectrum con nuove funzionalità e componenti grafiche, migliorando l’interfaccia utente (UI) e l’esperienza utente (UX).
3. La risoluzione di bug noti che rallentavano il processo di conversione e rendevano il software meno affidabile.

4. L'introduzione di un nuovo modulo che sfrutta le API di OpenAI per generare nomi descrittivi per le variabili, migliorando la leggibilità e la manutenibilità del codice.
5. Lo studio e l'applicazione di tecniche di Prompt Engineering per ottimizzare l'interazione con i modelli LLM.
6. L'implementazione di un chatbot integrato per supportare gli sviluppatori nella risoluzione di problemi comuni durante il processo di migrazione.

I contributi tecnici riflettono un approccio bilanciato tra miglioramenti pratici e ricerca. Ad esempio, l'aggiunta di un chatbot integrato, descritto nel Capitolo 3, è stata ispirata dalle richieste degli sviluppatori che necessitavano di uno strumento dinamico per risolvere problemi specifici, come il cambio di struttura di un codice complesso, risultante dalla conversione tramite il software Spectrum. Inoltre, l'uso di LLM per generare nomi descrittivi ha dimostrato di rendere più comprensibile e manutenibile il codice, come evidenziato nei risultati del Capitolo 4.

## 1.3 Principali contributi

Il contributo principale di questa tesi è rappresentato dall'estensione di un software che era esistente all'interno dell'azienda, che si occupa di tradurre dal linguaggio legacy XSLT utilizzato da TIBCO al linguaggio di MuleSoft, DataWeave. Questo software è stato migliorato con l'aggiunta di nuove funzionalità e componenti grafiche e un miglioramento generale della UI e UX, rendendo il software più user-friendly e facile da utilizzare.

Oltre a delle migliorie dovute all'interfaccia grafica, sono stati risolti dei problemi che erano già presenti, come alcuni bug noti che rendevano il processo più lento e meno affidabile. Questi hanno richiesto un lavoro di refactoring e di ottimizzazione del codice, che ha occupato una parte significativa del tempo di sviluppo.

Inoltre è stato aggiunto un nuovo modulo che sfrutta le potenzialità dell'intelligenza artificiale tramite l'utilizzo delle API di OpenAI, il quale utilizzo è stato notevolmente apprezzato e utilizzato dagli sviluppatori, che hanno potuto beneficiare di un supporto aggiuntivo rendendo il codice più pulito e affidabile. Gli utilizzi di questa tecnologia sono stati vari, facendo particolare attenzione alle task basate sul codice, ma che non sono state totalmente riportate o finalizzate nell'applicativo finale per motivi di costi e rischi, che verranno discussi nel dettaglio nei capitoli successivi.

## 1.4 Problemi noti e limiti

Un fattore chiave del progetto è stato la necessità di fare delle scelte strategiche per ottimizzare le risorse disponibili e concentrarsi sulle funzionalità più critiche e di maggiore impatto per il business.

Inoltre, un fattore cruciale quando si parla di queste tecnologie è senza ombra di dubbio il *fattore economico*, che ha limitato l'utilizzo di alcune strategie che avrebbero potuto portare a dei risultati migliori, ma che sono state scartate per via di costi e rischi giudicati non sostenibili. A questi si aggiungono anche i rischi connessi all'adozione di quest'ultime, che, data la natura intrinsecamente complessa dei Large Language Models, potrebbero portare a risultati meno idonei o addirittura insoddisfacenti rispetto alle aspettative.

I principali fattori limitanti riscontrati durante lo svolgimento della presente tesi sono i seguenti:

- La complessità del codice XSLT, che spesso include strutture annidate e regole di trasformazione intricate, rendendo difficile la conversione automatica in DataWeave.
- La limitata disponibilità di dataset specifici per addestrare modelli LLM nel contesto della conversione da XSLT a DataWeave.
- I costi elevati associati all'utilizzo di modelli LLM avanzati e alle chiamate API, che hanno richiesto un'attenta valutazione del rapporto costo-beneficio.
- Le preoccupazioni relative alla privacy e alla sicurezza dei dati sensibili contenuti nel codice sorgente, che hanno portato all'esplorazione di soluzioni basate su modelli locali o anonimizzazione dei dati.

Come discusso nel Capitolo 3, l'approccio basato su LLM ha mostrato risultati promettenti, ma ha anche evidenziato la necessità di una revisione manuale per garantire la correttezza del codice finale. Inoltre, i costi associati all'uso di modelli avanzati impongono limiti pratici all'adozione di queste soluzioni su larga scala, come sottolineato nel Capitolo 4.

Infine, un ulteriore limite è rappresentato dalla notevole complessità del software da espandere a causa della sua articolata architettura; tuttavia, tramite il supporto del team di sviluppo, è stato possibile contare su una guida e un supporto costante.

La sfida principale di questo lavoro di tesi è stata quella di dover creare soluzioni *out of the box* per problemi concreti, soggetti ai fattori limitanti di cui sopra. Il che ha richiesto una costante valutazione del *trade-off* tra



le possibili soluzioni tecniche, i loro costi, gli annessi rischi e i conseguenti benefici.

## 1.5 Descrizione del contenuto della tesi

Questa tesi è organizzata in quattro capitoli principali:

- **Capitolo 1 - Introduzione:** Presenta il contesto, gli obiettivi, i contributi principali e i problemi tecnici affrontati durante lo sviluppo del progetto.
- **Capitolo 2 - Tecnologie e strumenti:** Descrive le tecnologie utilizzate nel progetto, inclusi TIBCO, MuleSoft, DataWeave, XSLT, Large Language Models (LLM) e il software Spectrum. Vengono inoltre discusse le architetture e le metodologie di base.
- **Capitolo 3 - Implementazione e metodologie:** Descrive le implementazioni effettuate, le modifiche apportate al software Spectrum e le metodologie utilizzate per risolvere i problemi tecnici. Vengono presentati casi di studio pratici e analizzati i risultati ottenuti.
- **Capitolo 4 - Risultati e conclusioni:** Riassume i risultati ottenuti, discute i problemi riscontrati e propone possibili sviluppi futuri per migliorare ulteriormente il processo di migrazione.

## Capitolo 2

# Tecnologie e strumenti

In questo Capitolo 2 verranno discusse le tecnologie che sono state utilizzate ed analizzate. In particolare, verranno presentate le tecnologie di integrazione **TIBCO** e **MuleSoft**, con un focus sulle principali caratteristiche, differenze e problematiche connesse alla migrazione. Verranno inoltre presentate le tecnologie di intelligenza artificiale, in particolare i **Large Language Models (LLM)**, e una discussione generale su **Spectrum**, il software di migrazione che è stato esteso durante il periodo di tirocinio realizzato presso l'azienda.

Principalmente, gli sviluppi si sono basati meno sulle tecnologie come TIBCO e Mulesoft stesse, ma più sul miglioramento del software di supporto Spectrum e sulle metodologie di aiuto alla migrazione tramite algoritmi e tecniche di intelligenza artificiale.

**Definizione 2.1** (API (Application Programming Interface)). É un insieme di definizioni e protocolli che consentono a software e applicazioni di comunicare tra loro. Le API definiscono le regole e i metodi per l'interazione tra i diversi componenti software.

### 2.1 TIBCO

TIBCO rappresenta una delle soluzioni più consolidate e robuste nel panorama dell'integrazione aziendale, particolarmente apprezzata per la sua affidabilità in contesti complessi, fondata ufficialmente nel 1997<sup>1</sup>. L'azienda ha sviluppato una vasta gamma di prodotti e soluzioni per l'integrazione di sistemi e la gestione di processi aziendali. Uno dei prodotti più utilizzati è **TIBCO BusinessWorks**, una piattaforma di integrazione che consente

---

<sup>1</sup>[https://en.wikipedia.org/wiki/TIBCO\\_Software](https://en.wikipedia.org/wiki/TIBCO_Software)

di progettare, sviluppare e gestire flussi di lavoro e processi di integrazione tra applicazioni e sistemi eterogenei.

### 2.1.1 TIBCO ActiveMatrix BusinessWorks

TIBCO ActiveMatrix BusinessWorks è una suite di prodotti per l'integrazione di applicazioni aziendali. Questa piattaforma consente di creare servizi e integrare applicazioni attraverso un ambiente di sviluppo visivo e basato su modelli, per poi distribuirli nell'ambiente suo ambiente di runtime.

#### Caratteristiche Principali

TIBCO ActiveMatrix BusinessWorks utilizza l'interfaccia grafica (GUI) di Eclipse fornita da *TIBCO Business Studio™ for BusinessWorks* per definire i processi aziendali e generare artefatti distribuiti sotto forma di archivio. Questi artefatti possono essere distribuiti ed eseguiti nell'ambiente runtime.

L'integrazione viene proprio gestita attraverso l'uso di **processi di business**, che possono essere definiti come un insieme di attività correlate che consentono di raggiungere un obiettivo aziendale specifico. Questi processi possono essere implementati utilizzando un'ampia varietà di **servizi e componenti**. Principalmente, per le trasformazioni di dati strutturati in formato XML, vengono utilizzati gli **XSLT** (eXtensible Stylesheet Language Transformations).

### 2.1.2 XSLT (eXtensible Stylesheet Language Transformations)

Gli **XSLT** costituiscono una componente chiave nei sistemi basati su TIBCO, essendo utilizzati per trasformare e manipolare dati strutturati in formato XML. Questo è stato lo standard fino a qualche anno fa, ma con l'avvento di nuove tecnologie e nuovi approcci, poiché la maggior parte delle API comunicavano tramite protocollo **SOAP**, si è reso necessario un cambiamento per modernizzare i sistemi e adottare nuove tecnologie con **API RESTful**.

Analizziamo un esempio di codice XML e XSLT per comprendere meglio il funzionamento di questa tecnologia.

```
1 <root>
2   <element>
3     <name>John</name>
4     <surname>Doe</surname>
5   </element>
6 </root>
```

---

Listing 2.1: Esempio di codice XML

```
1 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/
  XSL/Transform">
2   <xsl:template match="/">
3     <html>
4       <body>
5         <h2>My CD Collection</h2>
6         <table border="1">
7           <tr bgcolor="#9acd32">
8             <th>Name</th>
9             <th>Surname</th>
10          </tr>
11          <tr>
12            <td><xsl:value-of select="root/element/name"/></td>
13            <td><xsl:value-of select="root/element/surname"/></td>
14          </tr>
15        </table>
16      </body>
17    </html>
18  </xsl:template>
19 </xsl:stylesheet>
```

Listing 2.2: Esempio di codice XSLT

Quello che fa l’XSLT è trasformare il codice XML in un formato diverso, in questo caso in un codice HTML. Il risultato sarà una tabella HTML con i valori di nome e cognome presi dal codice XML.

Name	Surname
John	Doe

Tabella 2.1: Risultato della trasformazione XSLT

## 2.2 MuleSoft

MuleSoft è una piattaforma che fa parte della famiglia *Salesforce* dal 2018, che consente alle aziende di automatizzare processi, integrare sistemi e dati, utilizzando un approccio low-code. Questa piattaforma offre agli sviluppatori strumenti per costruire integrazioni, automazioni e workflow in modo sicuro e scalabile, con tecnologie e sistemi modernizzati e in rapido aggiornamento, con l’utilizzo di un’interfaccia grafica basata su Eclipse. Grazie alla sua

natura modulare, MuleSoft accelera la consegna dei progetti, trasformando ogni risorsa digitale in un prodotto riutilizzabile, senza richiedere conoscenze tecniche approfondite.

### **Integrazione dei dati con MuleSoft**

MuleSoft si distingue per la sua capacità di integrare applicazioni e dati distribuiti in numerosi sistemi aziendali. Ogni organizzazione moderna gestisce una vasta quantità di applicazioni che contengono dati frammentati; MuleSoft risolve questo problema con un approccio *composable*, basato su API riutilizzabili e principi di integrazione cloud (iPaaS). Con iPaaS (Integration Platform as a Service) si intende che MuleSoft offre una piattaforma cloud per l'integrazione di applicazioni e dati, consentendo alle aziende di connettere sistemi legacy con tecnologie moderne.

Il punto di forza di Mulesoft è quello di offrire una piattaforma di integrazione e della creazione delle API semplice e scalabile. Particolarmente, utilizzando un paradigma chiamato **API-led connectivity**, che permette di creare API in modo modulare e riutilizzabile, semplifica di gran lunga la gestione duratura nel tempo delle API stesse.

### **API-led connectivity**

L'API-led connectivity è un approccio alla progettazione e all'integrazione delle API che consente di creare API modulari, riutilizzabili e scalabili. Questo approccio si basa su tre livelli di API:

- **System APIs:** API che consentono di accedere ai sistemi sottostanti, come database, applicazioni e servizi.
- **Process APIs:** API che consentono di orchestrare i processi aziendali, combinando più System APIs per creare nuove funzionalità.
- **Experience APIs:** API che consentono di creare esperienze utente personalizzate, combinando più Process APIs per offrire servizi completi.

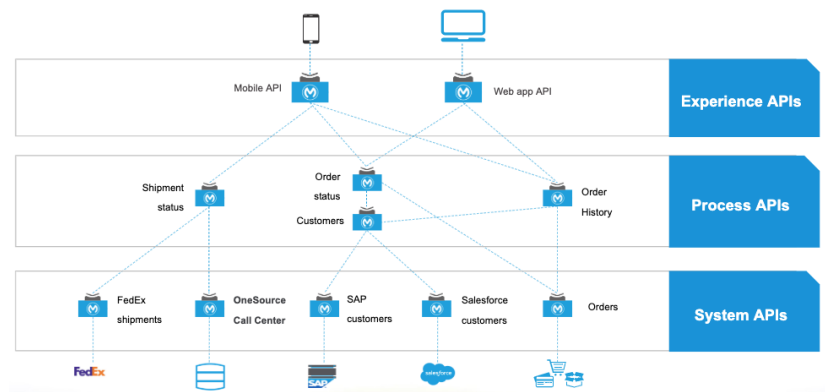


Figura 2.1: API-led connectivity

2

### 2.2.1 DataWeave

DataWeave è un linguaggio di trasformazione dati sviluppato da MuleSoft, progettato per convertire e manipolare dati tra diversi formati, come JSON, XML, CSV e altri. È un linguaggio dichiarativo e funzionale, il che significa che si concentra su **cosa** deve essere fatto piuttosto che su **come** farlo.

DataWeave è integrato nell'ecosistema MuleSoft Anypoint Platform, ma può essere utilizzato anche al di fuori di esso. La sintassi è semplice, ricordando vagamente quella di **Javascript**, non è fortemente tipizzata, ma la sua sintassi funzionale potrebbe essere un po' più complessa per chi non è abituato a questo tipo di programmazione.

### Un esempio di codice DataWeave

Il seguente esempio mostra come creare un oggetto JSON utilizzando DataWeave:

```
%dw 2.0
output application/json
---
{
    nome: "Mario",
    cognome: "Rossi",
    eta: 30
}
```

```
1 {  
2     "nome": "Mario",  
3     "cognome": "Rossi",  
4     "eta": 30  
5 }
```

Listing 2.3: Output del codice DataWeave

La direttiva `%dw 2.0` specifica la versione di DataWeave, mentre `output application/json` indica che il risultato sarà in formato JSON. Il corpo del codice definisce la struttura dell'output.

## Differenze tra DataWeave e XSLT

Per capire le differenze tra DataWeave e XSLT, possiamo partire da alcune caratteristiche chiave. Innanzitutto, entrambi i linguaggi sono dichiarativi, ma DataWeave ha una sintassi più moderna e funzionale, mentre XSLT è strettamente legato alla struttura XML. Questo rende DataWeave più adatto a scenari moderni, dove è necessario lavorare con diversi formati di dati come JSON, XML o CSV. XSLT, d'altra parte, è principalmente focalizzato su XML ed è uno standard W3C, il che lo rende molto diffuso in contesti tradizionali.

Un'altra differenza importante è il contesto di utilizzo: DataWeave è integrato nell'ecosistema MuleSoft Anypoint, mentre XSLT è più generico e può essere utilizzato in una varietà di ambienti. La sintassi di DataWeave è anche più compatta e facile da leggere, mentre quella di XSLT può risultare più verbosa a causa della sua natura basata su modelli XML.

Il fatto che DataWeave sia un linguaggio proprietario di MuleSoft può essere visto come un vantaggio o uno svantaggio a seconda del contesto. Infatti, nel mondo *open-source*, XSLT è più diffuso e supportato da una vasta comunità di sviluppatori, mentre DataWeave ha meno visibilità e meno estensioni disponibili, se non per quelle sviluppate direttamente da MuleSoft e Salesforce.

## 2.3 Large Language Models (LLM)

I Large Language Models sono una classe di modelli di intelligenza artificiale che si focalizzano sull'elaborazione del linguaggio naturale. Questi modelli, grazie alla loro architettura e dimensione, sono in grado di comprendere, generare e manipolare testo in modo estremamente sofisticato.

### 2.3.1 Architettura dei LLM

**Definizione 2.2.** (Rete Neurale) Una rete neurale è un modello computazionale che tramite l'uso di algoritmi di apprendimento automatico cerca di emulare il funzionamento del cervello umano. Le reti neurali sono composte da unità di calcolo chiamate neuroni, organizzate in strati (layers) e connesse tra loro da pesi sinaptici. Con una grande mole di dati di addestramento, le reti neurali possono apprendere a riconoscere pattern e relazioni complesse nei dati.

Gli LLM si basano principalmente su architetture di reti neurali (2.2) di tipo Transformer, che sono state introdotte nel 2017 nel lavoro "Attention is All You Need"[4]. I Transformer si distinguono per l'uso del meccanismo di *self-attention*, che permette al modello di pesare l'importanza di ogni parola in una sequenza rispetto alle altre, a prescindere dalla loro distanza relativa.

In un Transformer, ogni parola (o token) viene trasformata in un vettore numerico (embedding) che rappresenta il suo significato semantico. La rete poi applica il meccanismo di attenzione, che valuta come ogni parola influenza le altre in un dato contesto. Questo approccio consente al modello di gestire lunghe sequenze di testo in modo più efficace rispetto a modelli precedenti come le reti neurali ricorrenti (RNN) o le Long Short-Term Memory (LSTM), che avevano difficoltà con le dipendenze a lungo termine.



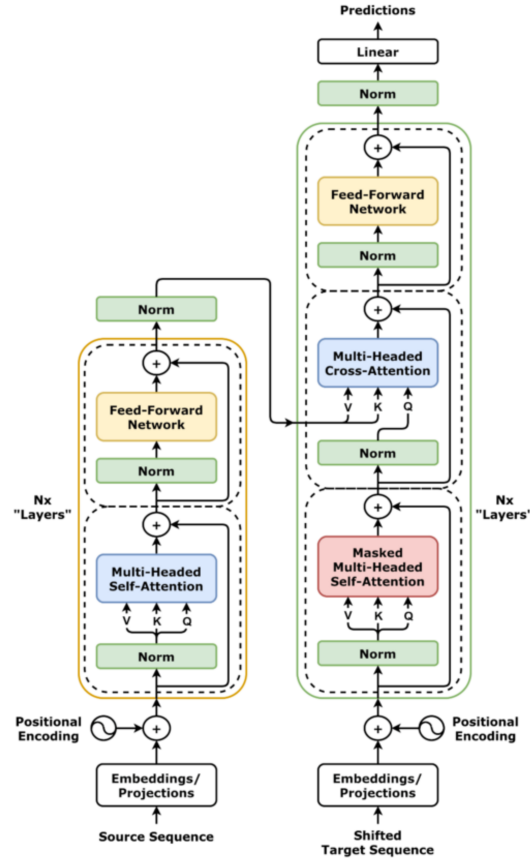


Figura 2.2: Architettura di un Transformer

Un LLM è generalmente costruito con molteplici strati (layers) di Transformer, che gli consentono di apprendere rappresentazioni sempre più complesse del linguaggio. La dimensione dei modelli, espressa in termini di parametri, è uno degli aspetti che determina la potenza e la capacità di generalizzazione del modello ed è ciò che rende difficile l'utilizzo privato o personale di questi modelli, poiché richiedono unità di calcolo con elevata efficienza e quantità di memoria, che difficilmente è accessibile a tutti; banalmente, le schede grafiche più performanti per gli utenti medi riescono a far partire i modelli di medie dimensioni, mentre quelli più parametri sono ancora fuori scala. Per comprendere un po' meglio la scala di questi ultimi, in tabella 2.2 sono riportati alcuni esempi di LLM con il numero di parametri associati.

Modello	Parametri
LLaama 3.1 405B	405 miliardi
GPT-3 (Generative Pre-trained Transformer 3)	175 miliardi
Mixtral	45 miliardi
BERT Large	334 milioni
RoBERTa (Robustly optimized BERT approach)	355 milioni
T5 (Text-to-Text Transfer Transformer)	220 milioni

Tabella 2.2: Esempi di Large Language Models e numero di parametri

Non ci sono state dichiarazioni da parte di OpenAI, ma si stima che GPT-4 avesse un numero di parametri di circa 1.7T (trilioni), ma non ci sono conferme ufficiali. Si pensava a questo numero per un utilizzo dell'architettura MOE (Mixture of Experts), come fa Mixtral[2].

**Definizione 2.3.** (Mixture of Experts) La tecnica *Mixture of Experts* (MoE) consiste nell'uso di più reti neurali esperte, ognuna delle quali è responsabile per un sottoinsieme di compiti. In un modello MoE, l'output per un dato input è determinato dalla somma ponderata degli output delle reti esperte, dove i pesi sono forniti da una rete di gating. Una rete di gating è una rete neurale che determina quali esperti (o sotto-modelli) devono essere attivati in un modello di Mixture of Experts, utilizzando una funzione di assegnazione dei pesi per ogni esperto. Se il vettore di gating è sparso, è possibile evitare di calcolare gli output degli esperti il cui gate è zero. La tecnica consente di aumentare il numero di esperti nel modello senza aumentare esponenzialmente i costi computazionali, poiché viene utilizzato solo un sottoinsieme degli esperti per ogni token elaborato. [2]

### 2.3.2 Punti di forza e debolezza

Gli LLM potrebbero sembrare la soluzione a molti problemi di elaborazione del linguaggio naturale e non solo, estendendo questo anche ad altri task come la generazione di codice, ed infatti è proprio quello di cui si discuterà in seguito. Tuttavia, questi modelli presentano anche alcune criticità e limitazioni che è importante considerare, come possono essere i costi elevati di utilizzo, manutenzione, addestramento privato e del problema delle allucinazioni.

**Definizione 2.4.** (Allucinazioni) Nel campo dell'intelligenza artificiale (IA), un'allucinazione o allucinazione artificiale è una risposta generata dall'IA che contiene informazioni false o fuorvianti presentate come fatti. [5]

Un esempio è una generazione di codice in un linguaggio di programmazione che è completamente inventato.

Esempio: Genera una funzione Python che somma due numeri.

```
1      def somma(a, b){
2          a + b
3      }
4
5      cout << somma(1,2) << endl;
```

Il codice potrebbe sembrare corretto, ma in realtà non lo è, poiché il codice sta utilizzando costrutti di **C++** e non di **Python**.

## 2.4 Spectrum

SPECTRUM è un software progettato per facilitare la conversione di codici **XSLT** nel formato di trasformazione di Mulesoft, **DataWeave**. L'obiettivo è quello di automatizzare quasi in modo totale la conversione di codice legacy, andando ad analizzare e cambiare la struttura e la sintassi del codice **XSLT** in modo da renderlo compatibile con il nuovo formato; questo con una complessa architettura software divisa in vari moduli, ognuno con un compito specifico.

Spectrum è basato sull'utilizzo di tecniche di *code conversion* tramite la rappresentazione del codice in un **AST** (Abstract Syntax Tree), e il supporto alla conversione tramite **BNF** (Backus-Naur Form) e **regole di trasformazione**.

**Definizione 2.5.** (AST) Un Abstract Syntax Tree (AST) è una struttura di dati utilizzata in informatica per rappresentare la struttura di un programma o di un frammento di codice. È una rappresentazione ad albero della struttura sintattica astratta di un testo (spesso codice sorgente) scritto in un linguaggio formale. Ogni nodo dell'albero denota un costrutto presente nel testo. A volte viene chiamato semplicemente albero della sintassi.<sup>3</sup>

---

<sup>3</sup>[https://en.wikipedia.org/wiki/Abstract\\_syntax\\_tree](https://en.wikipedia.org/wiki/Abstract_syntax_tree)

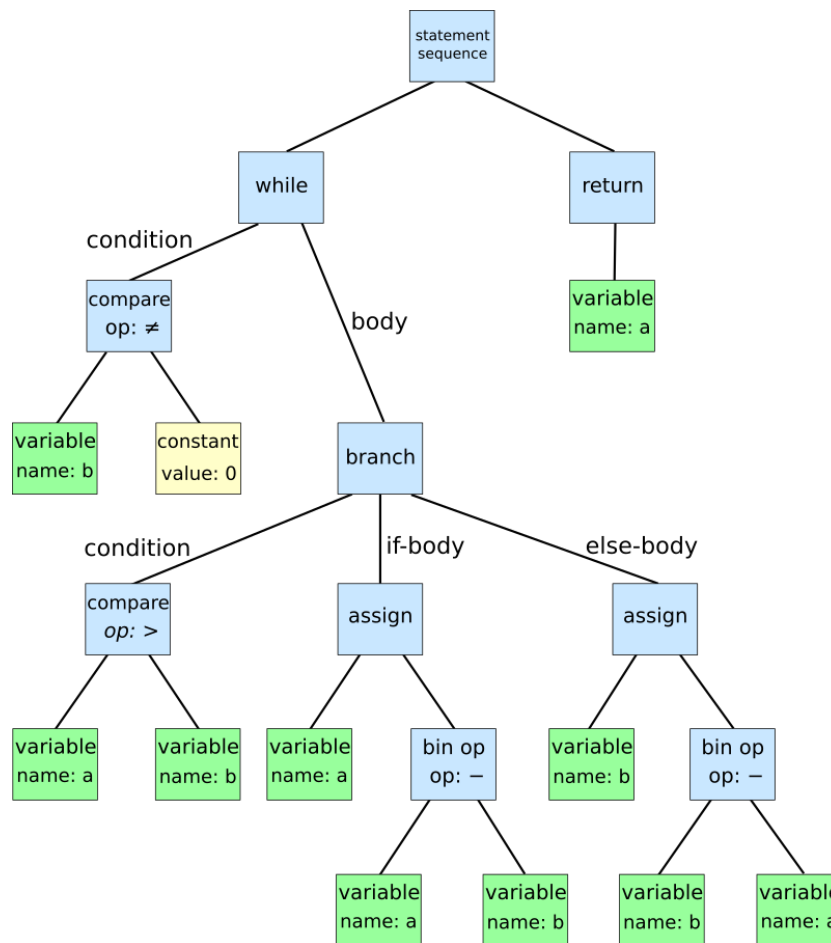


Figura 2.3: Esempio di un AST

**Definizione 2.6.** (BNF) In informatica, la forma Backus-Naur (BNF, Backus normal form) è una notazione utilizzata per descrivere la sintassi dei linguaggi di programmazione o di altri linguaggi formali. La BNF può essere descritta come una notazione metasintattica per le grammatiche libere dal contesto. La forma Backus-Naur viene applicata ovunque siano necessarie descrizioni esatte dei linguaggi, ad esempio nelle specifiche ufficiali dei linguaggi, nei manuali e nei libri di testo sulla teoria dei linguaggi di programmazione.

4

```

1      <expression> ::= <term> | <expression> "+" <term> | <
expression> "-" <term>
2      <term> ::= <factor> | <term> "*" <factor> | <term> "/" <
factor>
3      <factor> ::= <number> | "(" <expression> ")"

```

<sup>4</sup>[https://en.wikipedia.org/wiki/Backus%E2%80%93Naur\\_form](https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form)

---

Questo rappresenta un'espressione matematica, con le operazioni di somma, sottrazione, moltiplicazione e divisione.

### 2.4.1 Regole di Trasformazione

Una volta ottenuta la rappresentazione intermedia, SPECTRUM applica un insieme di **regole di trasformazione** per convertire i nodi dell'AST in normalizzazioni DataWeave. Queste regole includono:

1. *Gestione della sintassi*: adeguamento della struttura sintattica ai requisiti del linguaggio di destinazione.
2. *Inserimento di keyword*: utilizzo delle parole chiave specifiche di DataWeave.
3. *Adattamento dei nomi delle funzioni*: mappatura dei nomi delle funzioni XSLT alle loro controparti equivalenti.
4. *Generazione di variabili*: qualora fossero presenti variabili, verranno generate e inizializzate; se generiche, verranno generate usando un nome standard incrementale.

Questo processo è complesso, poiché non esiste una corrispondenza diretta tra molte funzioni e costrutti di XSLT e DataWeave. Inoltre, la sintassi e la semantica dei due linguaggi sono molto diverse, richiedendo un'attenta analisi e una rielaborazione del codice che non sempre viene effettuata nel modulo di trasformazione, bensì nel modulo di **post-processing**.

Spectrum, genera l'Abstract Syntax Tree (AST) a partire dalle Backus-Naur Form (BNF). In pratica, le BNF definiscono le regole grammaticali del linguaggio, specificando come le varie componenti sintattiche possono essere combinate. A partire da queste regole, Spectrum costruisce l'AST.

### 2.4.2 Post-Processing

Questo modulo è stato principalmente esteso durante il periodo di tirocinio, nonostante la sua esistenza già in precedenza, ma con funzionalità limitate. La fase di **post-elaborazione** si occupa di:

- Arricchire il codice risultante.
- Applicare modifiche al codice in base alle esigenze dell'utente

- Applicare trasformazioni qualora non siano state applicate correttamente durante la fase di trasformazione, per impossibilità di riconoscimento.

La discussione su Spectrum occuperebbe gran parte del documento vista la sua complessa architettura e funzionalità, ma è importante sottolineare che il progetto è stato esteso con l'aggiunta di nuove funzionalità e miglioramenti, e che la sua base di funzionamento è quella descritta ad *altissimo* livello nei paragrafi precedenti.

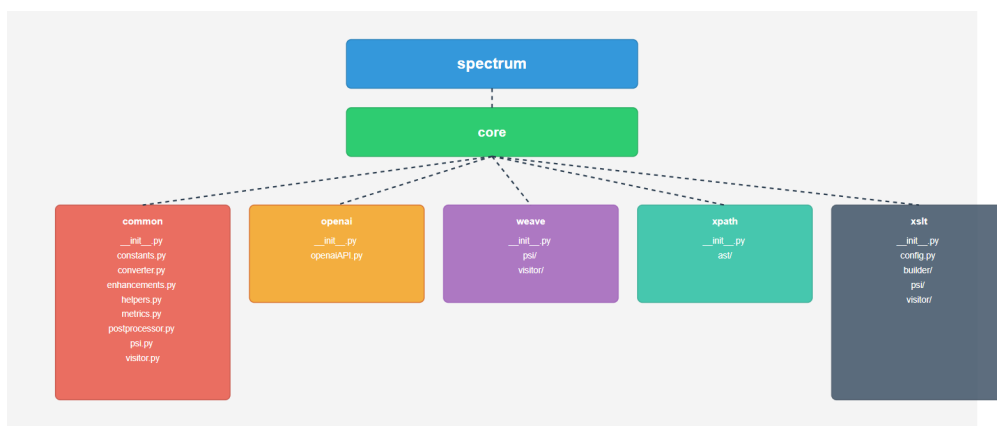


Figura 2.4: Architettura di Spectrum

Spectrum è stato sviluppato in Python e funziona senza l'utilizzo di una libreria grafica; Tuttavia, per migliorare l'utilizzo e semplificare la vita ai programmatori è stato pensato di rendere l'interfaccia grafica più user-friendly, con l'aggiunta di un'interfaccia grafica tramite l'utilizzo di **Streamlit**, che era già presente all'interno del progetto aziendale, ma è stata ulteriormente estesa e rivista per adattarsi alle nuove funzionalità durante il periodo di tirocinio.

## 2.5 Streamlit

Streamlit è una libreria open-source per Python progettata per creare applicazioni web interattive, particolarmente utili nel campo della data science e del machine learning e viene utilizzata principalmente per la creazione di dashboard e applicazioni di visualizzazione dei dati. Tuttavia, la sua semplicità e flessibilità lo rendono uno strumento ideale per una vasta gamma di applicazioni, e non richiedendo una conoscenza approfondita di HTML, CSS o JavaScript, consente a chiunque di creare applicazioni web in modo

rapido e intuitivo. Ovviamente questo comporta dei limiti legati ai vincoli che la libreria impone, ma per la maggior parte delle applicazioni è più che sufficiente.

Ciò che rende Streamlit particolarmente adatto per la creazione di applicazioni web è la sua capacità di integrare facilmente funzionalità interattive come slider, input di testo, selettori e checkbox. Inoltre, Streamlit supporta la condivisione e la distribuzione delle applicazioni attraverso funzionalità di deployment sul cloud. Infine, Streamlit offre la possibilità di personalizzare l'aspetto e il layout dell'applicazione tramite variabili predefinite, anche se molto limitate.

Il suo funzionamento è **top-down**, quindi ogni azione che viene eseguita all'interno del programma comporta una riesecuzione dell'intero codice, e questo può comportare dei problemi di performance in caso di applicazioni molto complesse o con molte chiamate a funzioni esterne, oltre ad un funzionamento errato dell'applicazione, perciò è stato richiesto un lavoro di ottimizzazione e di refactoring per evitare problemi di questo tipo.<sup>5</sup>

## 2.6 OpenAI API

Le API di OpenAI sono un'interfaccia di programmazione progettata per consentire l'accesso a modelli avanzati di intelligenza artificiale sviluppati da OpenAI. Si basano su un'interazione general-purpose denominata "*text in, text out*", permettendo di utilizzare i modelli per una vasta gamma di task, dipendentemente dal modello utilizzato.

### 2.6.1 Utilizzo per gpt-4o-mini

Un esempio concreto di utilizzo delle API di OpenAI è la possibilità di chiamare modelli come gpt-4o-mini per compiti di generazione di testo avanzati. Questi modelli sono progettati per gestire input complessi e fornire risposte coerenti, rendendoli ideali per applicazioni che richiedono un linguaggio naturale avanzato. Nel progetto, le API chiamano principalmente *gpt-4o-mini* per la generazione di testo e codice, e le caratteristiche che lo rendono un modello ottimo per il task sono:

- Finestra di contesto: 128.000 token, ovvero quanti token, che sono le unità di testo, il modello può accettare in **input**.
- Max Output Tokens: 16.384 token, ovvero quanti token il modello può generare in **output**.

---

<sup>5</sup><https://streamlit.io/>

Secondo le *docs* di OpenAI, il modello *gpt-4o-mini-2024-07-18* ha come prezzi per le chiamate su 1 Milione di token: <sup>6</sup>

Input Tokens	Cached Input Tokens	Output Tokens
\$0.15	\$0.0075	\$0.60

## 2.7 Chatbot Arena (ex. LMSYS)

Chatbot Arena <sup>7</sup> è una piattaforma online progettata per confrontare e testare diversi modelli di intelligenza artificiale (AI) per la generazione di testo e immagini. La piattaforma consente agli utenti di interagire in stile *chatbot* con una vasta gamma di LLM, come ChatGPT, Gemini, Claude, Llama e altri, che siano *open source* o meno.

La piattaforma è stata utilizzata per mettere a confronto i vari LLM per la generazione di codice, in particolare per effettuare test in modo uniforme sui vari task e per valutare le prestazioni dei modelli in base alla correttezza della risposta, senza considerare metriche numeriche come BLEU o ROUGE, che sono più adatte per la valutazione di traduzioni o generazioni di testo più generiche, poiché i test non erano condotti su un dataset standardizzato.

---

<sup>6</sup><https://platform.openai.com/docs/pricing>

<sup>7</sup><https://lmarena.ai/>



## Capitolo 3

# Implementazione e metodologie

In questo capitolo si discuterà delle implementazioni e delle modifiche apportate al progetto Spectrum 2.4, dei suoi limiti iniziali e dei problemi che erano presenti prima delle modifiche. Si parlerà inoltre delle metodologie utilizzate per la risoluzione di tali problemi e delle scelte progettuali che hanno portato il software al suo stato attuale. Verranno analizzati in dettaglio i vari passaggi che hanno portato alla risoluzione dei problemi, con un focus particolare sulle tecniche utilizzate e sui risultati ottenuti. Oltre a questo si discuterà dell'implementazione dell'intelligenza artificiale, di quali sono stati i limiti raggiunti e i vincoli imposti dal progetto, delle tecniche di prompting e di come sono state implementate e di alcune strategie che sono state scartate per la complessità o per la scarsa efficacia. Verranno inoltre presentati alcuni casi di studio che illustrano l'applicazione pratica delle metodologie discusse, con esempi concreti e dati empirici che dimostrano l'efficacia delle soluzioni adottate. Infine, si farà una riflessione sulle possibili evoluzioni future del progetto e sulle potenziali aree di miglioramento, con suggerimenti per ulteriori ricerche e sviluppi.

### 3.1 Estensione grafica

Il software **Spectrum**, nella sua prima implementazione su Streamlit, presentava un'interfaccia molto semplice. Questa permetteva di caricare un file XSLT come input, effettuare la conversione automatica e visualizzare il codice risultante direttamente sulla piattaforma, oltre alla possibilità di scaricare il file generato. Tuttavia, questa configurazione iniziale presentava alcune limitazioni operative.

Uno dei principali problemi riscontrati riguardava il funzionamento del software, progettato per lavorare in modalità **top-down**. Ciò significava che

ogni componente presente nel codice e ogni funzione chiamata venivano rielaborate in modo sistematico dal basso verso l'alto a ogni azione compiuta dall'utente. Questo comportamento comportava un aumento del tempo necessario per eseguire anche semplici operazioni, rendendo l'interfaccia meno fluida e l'esperienza utente più macchinosa.

Per risolvere questo problema, è stato introdotto un primo miglioramento significativo. Si è intervenuti sulle componenti grafiche dell'interfaccia, modificandole e ottimizzandole per renderle più **accessibili** e intuitive, andando a cambiare la logica di esecuzione del codice. Questa parte del software non era problematica o complessa, ma era comunque necessario apportare delle modifiche per poter rendere più scalabile e personalizzabile il progetto con le nuove implementazioni che ci sarebbero state.

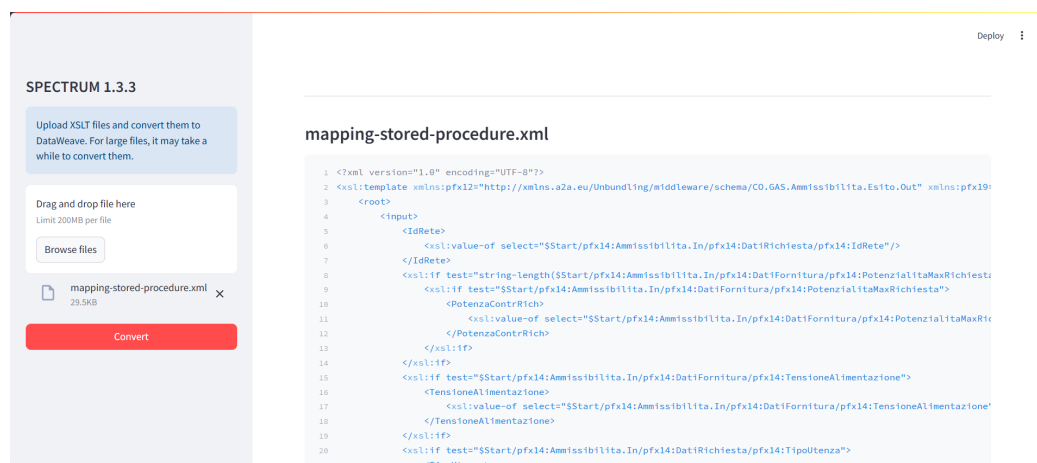


Figura 3.1: Interfaccia grafica di Spectrum prima dell'estensione grafica

## Sidebar aggiornata

La sidebar dell'applicazione è stata aggiornata con alcune modifiche importanti per migliorarne l'usabilità e garantire un utilizzo conforme alle policy aziendali. Tra i primi cambiamenti apportati, è stato aggiunto un *disclaimer* che specifica che il software è destinato esclusivamente a un utilizzo interno all'azienda *Lutech S.p.A* e riservato al personale autorizzato.


Un'altra funzionalità introdotta è stata la possibilità per l'utente di inserire una propria *API Key* di **OpenAI**. Questa chiave è necessaria per alcune funzionalità del programma che verranno discusse successivamente.

Inoltre, è stata aggiunta un'opzione per selezionare il formato di output desiderato tra XML e JSON. Questa scelta è motivata dal fatto che DataWeave

supporta vari formati di output, ma XML e JSON risultano essere i più utilizzati nella maggior parte dei casi, mentre gli altri vengono raramente richiesti.

A parte queste modifiche, non sono stati apportati ulteriori cambiamenti significativi alla struttura e alle funzionalità della sidebar, che è rimasta relativamente simile nel design e nella funzionalità.


ver 1.3.5



Spectrum is property of Lutech S.p.A. Only authorized personnel can use this software and the data provided is confidential. Unauthorized use is prohibited.

Developed by Lutech S.p.A. - 2025


Upload XSLT files and convert them to DataWeave. For large files, it may take a while to convert them.

OpenAI API Key 

Drag and drop file here

Limit 200MB per file

Browse files

Output type  
XML 

Convert

Figura 3.2: Sidebar aggiornata di Spectrum

## Code visualizer

Nella sua versione precedente, l'interfaccia permetteva di visualizzare il codice XSLT o il codice DataWeave convertito, ma esclusivamente uno per volta. Questa limitazione rendeva difficile confrontare direttamente i due linguaggi o passare rapidamente da uno all'altro, soprattutto in contesti in cui era necessario analizzare entrambe le implementazioni.

Per risolvere a questo problema è stata introdotta un'implementazione più flessibile e intuitiva. L'interfaccia è stata suddivisa in tre aree distinte:

- Una sezione dedicata esclusivamente alla visualizzazione del codice XSLT di base, utile per chi deve lavorare direttamente con questo linguaggio.
- Una sezione riservata al codice DataWeave, che consente di esaminare la trasformazione dei dati in questo formato.
- Una terza sezione che offre una visione *side by side*, permettendo di confrontare direttamente il codice XSLT e DataWeave in modo simultaneo.

Questo non solo migliora l'esperienza utente, ma rende anche più efficiente il lavoro di sviluppo e debugging, offrendo una panoramica completa e immediata delle due rappresentazioni del codice, che hanno spesso bisogno di un confronto diretto per individuare eventuali errori o discrepanze.



Figura 3.3: Interfaccia grafica di Spectrum dopo l'estensione grafica

Mettendo a paragone la Figura 3.3 con Figura 3.1, è facilmente intuibile come l'interfaccia sia più pulita e utile per uno sviluppatore che ha bisogno

di controllare se la conversione sia andata a buon fine e, in caso di problemi, individuarli più facilmente.

## Cambiamenti e migliorie minori

Oltre alle modifiche principali, sono state apportate diverse migliorie minori che hanno contribuito a rendere l'interfaccia più funzionale. Una di queste è stata la rimozione di alcuni messaggi statici permanenti. Al loro posto, è stato introdotto un sistema di notifiche più dinamico e contestuale, basato su messaggi *toast*.

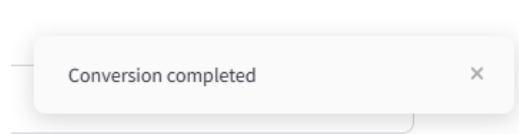


Figura 3.4: Esempio di notifica toast

Questo sistema fornisce un feedback immediato all'utente in base alle azioni svolte; in caso di operazioni completate con successo, viene visualizzata una conferma per la correttezza dell'operazione, e viceversa in caso di errori o problemi.

Oltre alle migliorie già citate, sono state introdotte ulteriori modifiche e aggiunte che verranno approfondite nel momento in cui si affronteranno i singoli problemi trattati. Queste modifiche includono l'implementazione di nuovi componenti all'interno del programma, ciascuno progettato per affrontare specifiche esigenze o criticità emerse durante lo sviluppo e che hanno richiesto uno sforzo maggiore rispetto alle modifiche grafiche.

## 3.2 Necessità di un'intelligenza artificiale

Il progetto *Spectrum*, nella sua versione iniziale, era basato su un approccio puramente **rule-based**. Questo significa che il software si affidava a regole e pattern predefiniti per analizzare e trasformare il codice da un linguaggio all'altro. Sebbene questo approccio funzionasse adeguatamente per i casi d'uso più conosciuti e definiti, si sono presto evidenziati i suoi limiti per alcuni costrutti più complessi.

Come requisito fondamentale per la nuova versione del software, è stato esplicitamente richiesto di integrare un'intelligenza artificiale all'interno del programma. Questa scelta nasce dall'esigenza di superare le limitazioni del sistema *rule-based*, introducendo un approccio più dinamico e adattabile,

capace di comprendere e gestire strutture complesse. L'obiettivo principale era quello di semplificare ulteriormente il lavoro degli sviluppatori, riducendo al minimo gli interventi manuali e garantendo risultati più precisi e coerenti, indipendentemente dalla complessità del codice di partenza.

L'integrazione dell'intelligenza artificiale ha comportato una serie di sfide, sia di natura tecnica che metodologica. È stato necessario sviluppare un'infrastruttura che consentisse l'integrazione dell'AI nel flusso di lavoro del software senza compromettere le prestazioni. Dal punto di vista metodologico, sono stati definiti criteri per valutare l'efficacia del sistema. Principalmente, il sistema sarebbe stato valutato in base alla sua capacità di generare codice DataWeave che avrebbero dovuto avere bisogno di meno modifiche manuali possibili per essere funzionante.

Questa sfida non è banale poichè la *generazione di codice* da parte di un LLM può portare a dei problemi: se da un lato permette di automatizzare e semplificare il lavoro degli sviluppatori, dall'altro può generare codice non ottimale o non conforme agli standard, richiedendo ulteriori interventi manuali per renderlo funzionante, introducendo *bug* da risolvere e *debug* da effettuare.

### 3.3 Conversione tramite Large Language Model

Un'idea preliminare, se non la prima che è stata proposta, è stata quella di utilizzare un **Large Language Model** (LLM) per la conversione del codice XSLT in DataWeave. Questi modelli, come **GPT-4o** di **OpenAI**, sono in grado di generare codice partendo da un prompt testuale ma sono limitati da ciò che viene chiamata *finestra di contesto*, ovvero la quantità di testo che il modello è in grado di "vedere" per generare il testo successivo. Questo significa che, se il modello non è in grado di vedere l'intero codice XSLT, potrebbe non essere in grado di generare un codice DataWeave corretto. Questi non sono gli unici problemi, ma analizziamo in ordine quali possono essere le problematiche legate a questo approccio.

Questa idea è stata proposta anche prima che il progetto Spectrum prendesse forma, quindi partendo direttamente da un codice XSLT e generando un codice DataWeave. L'idea era stata proposta avendo in mente che lo sviluppatore dovesse fare comunque qualche modifica al codice, ma che il modello avrebbe potuto generare una base di partenza da cui partire. Questo approccio è stato scartato inizialmente per diversi motivi:

- **Complessità del codice:** il codice XSLT è spesso molto complesso e articolato, con strutture annidate e regole di trasformazione complesse. Un modello LLM potrebbe avere difficoltà a comprendere e replicare queste strutture in modo accurato, generando codice DataWeave non funzionante o non conforme agli standard.
- **Limiti del modello:** i modelli LLM hanno una capacità limitata di generare codice in base al contesto fornito. Se il modello non è in grado di "vedere" l'intero codice XSLT, potrebbe non essere in grado di generare un codice DataWeave corretto. Questo potrebbe comportare la generazione di codice non funzionante o non ottimale, richiedendo ulteriori interventi manuali per renderlo operativo.
- **Allucinazioni:** la generazione di codice richiede una comprensione approfondita del linguaggio di partenza e di quello di destinazione. I modelli LLM possono generare codice "allucinato", come ad esempio *sintassi* errata, *variabili* non definite o *funzioni* non esistenti.
- **Scarsa affidabilità:** le motivazioni riportate in precedenza portano ad un risultato finale poco affidabile, che richiede ulteriori interventi manuali per renderlo funzionante. Questo comporta un aumento del tem-



po e delle risorse necessarie per completare la conversione, annullando i vantaggi dell'automazione.

Nonostante le problematiche evidenziate, l'idea di sfruttare un modello LLM per la conversione automatica del codice XSLT in **DataWeave** è stata comunque presa in considerazione. L'impiego di modelli di linguaggio avanzati rappresentava infatti un'opportunità interessante per esplorare nuove soluzioni e approcci che potessero ridurre il carico di lavoro degli sviluppatori e rendere il processo di conversione più veloce ed efficiente.

L'obiettivo principale era verificare se un LLM potesse generare un codice DataWeave corretto e funzionale partendo da un semplice codice XSLT, evitando errori sintattici e rispettando la logica della trasformazione. Per testare questa ipotesi, è stato condotto un esperimento preliminare in cui diversi modelli sono stati interrogati per eseguire la conversione sulla base di uno stesso input.

Per eseguire il test, è stata utilizzata la piattaforma **Chatbot Arena** (vedi 2.7), un'applicazione che consente di confrontare le prestazioni di diversi modelli di LLM all'interno di un contesto di *chat*. Questa piattaforma è risultata particolarmente utile perché permette di interagire con diversi modelli contemporaneamente e analizzare i risultati ottenuti con la stessa richiesta.

### Primo esperimento

Il primo esperimento è stato condotto interrogando quattro LLM con lo stesso prompt, chiedendo loro di convertire un codice XSLT relativamente semplice in DataWeave. I modelli selezionati per il test sono stati:

- **GPT-4o**
- **Claude-3-Opus**
- **DeepSeek-R1**
- **Llama-3.1-405B-Instruct-bf16**

Questi modelli sono stati scelti in quanto, al momento del test, risultavano essere tra i migliori nella classifica di Chatbot Arena per quanto riguarda le *task* legate alla programmazione. Inoltre, ciascuno di essi è stato sviluppato con approcci differenti, il che ha permesso di confrontare come modelli con diverse architetture affrontassero il problema della conversione del codice.

Il prompt utilizzato per interrogare i modelli è stato il seguente:

*You are a helpful assistant. I will send you some XSLT code and you should provide me with the corresponding converted DataWeave code, being careful not to make syntax errors.*

L'obiettivo del test era valutare diversi aspetti della generazione automatica del codice, tra cui:

- La correttezza sintattica del codice generato.
- L'aderenza alla logica del codice XSLT originale.
- La presenza di eventuali omissioni o errori concettuali nella conversione.
- La necessità di interventi manuali per correggere il codice prodotto.

I risultati ottenuti hanno permesso di trarre importanti conclusioni sull'efficacia dell'approccio basato su LLM, evidenziando sia i vantaggi che le criticità di questa soluzione.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:template xmlns:xsl="http://www.w3.org/1999/XSL/
  Transform" xmlns:pd="http://xmlns.tibco.com/bw/process/2003"
  xmlns:BW="java://com.tibco.pe.core.JavaCustomXPathFunctions"
  xmlns:ns="http://www.tibco.com/pe/EngineTypes" xmlns:xsd="
  http://www.w3.org/2001/XMLSchema" xmlns:CustomBWFunction="
  java://CustomBwFunction" xmlns:ns0="http://www.tibco.com/pe/
  GenerateErrorActivity/InputSchema" xmlns:tib="http://www.
  tibco.com/bw/xslt/custom-functions" xmlns:ns2="http://www.ibm
  .com/maximo" xmlns:ns1="http://www.ibm.com/maximo/wsdl/
  ZOESB_ZOMXOPERLOC_BYMX">
3 <xsl:choose>
4   <xsl:when test="count($QueryZOMXOPERLOC/
  ns1:QueryZOMXOPERLOCResponse/parameters/
  ns2:QueryZOMXOPERLOCResponse/ns2:ZOMXOPERLOCSet/ns2:LOCATIONS
  /ns2:LOCATIONSPEC[ns2:ASSETATTRID = 'A2A000000546'][string-
  length(ns2:ALNVALUE) > 0 or string-length(ns2:ZODATEVALUE) >
  0]) > 0">
5     <jdbcUpdateActivityInput>
6       <PSRTYPE>
7         <xsl:value-of select="'Cabina secondaria con
  telecontrollo'" />
8       </PSRTYPE>
9       <FEEDEROBJECTTYPE>
10        <xsl:value-of select="'ServiceSubstation'" />
11      </FEEDEROBJECTTYPE>
12      <MRID>
13        <xsl:value-of select="$current/Record/MRID" /
  >
```

```

14         </MRID>
15         <CIRCUIT>
16             <xsl:value-of select="$Start/root/circuit"/>
17         </CIRCUIT>
18     </jdbcUpdateActivityInput>
19 </xsl:when>
20 <xsl:otherwise>
21     <jdbcUpdateActivityInput>
22         <PSRTYPE>
23             <xsl:value-of select="'Cabina secondaria
senza telecontrollo'"/>
24         </PSRTYPE>
25         <FEEDEROBJECTTYPE>
26             <xsl:value-of select="'Vista'"/>
27         </FEEDEROBJECTTYPE>
28         <MRID>
29             <xsl:value-of select="$current/Record/MRID"/
>
30         </MRID>
31         <CIRCUIT>
32             <xsl:value-of select="$Start/root/circuit"/>
33         </CIRCUIT>
34     </jdbcUpdateActivityInput>
35 </xsl:otherwise>
36 </xsl:choose>
37 </xsl:template>

```

I risultati ottenuti, per ogni modello, sono i seguenti:

#### **GPT-4o:**

```

1 %dw 2.0
2 output application/xml
3 ns ns1 http://www.ibm.com/maximo/wsd1/Z0ESB_Z0MXOPERLOC_BYMX
4 ns ns2 http://www.ibm.com/maximo
5
6 var QueryZ0MXOPERLOC = payload.QueryZ0MXOPERLOC
7
8
9 {
10     jdbcUpdateActivityInput :
11         if (
12             (QueryZ0MXOPERLOC.ns1#QueryZ0MXOPERLOCResponse.
parameters.ns2#QueryZ0MXOPERLOCResponse.ns2#Z0MXOPERLOCSet.
ns2#LOCATIONS.ns2#LOCATIONSPEC filter ((item) ->
13                 item.ns2#ASSETATTRID == "A2A000000546" and
14                 (sizeof(item.ns2#ALNVALUE default "") > 0 or
sizeof(item.ns2#Z0DATEVALUE default "") > 0)
15             )).sizeof() > 0

```

```

16     )
17     {
18         PSRTYPE: "Cabina secondaria con telecontrollo",
19         FEEDEROBJECTTYPE: "ServiceSubstation",
20         MRID: vars.current.Record.MRID,
21         CIRCUIT: vars.Start.root.circuit
22     }
23     else
24     {
25         PSRTYPE: "Cabina secondaria senza telecontrollo",
26         FEEDEROBJECTTYPE: "Vista",
27         MRID: vars.current.Record.MRID,
28         CIRCUIT: vars.Start.root.circuit
29     }
30 }

```

### Claude-3-Opus:

```

1  %dw 2.0
2  output application/xml
3  ---
4  {
5      jdbcUpdateActivityInput: {
6          PSRTYPE: if (sizeof(QueryZ0MXOPERLOC.
7              QueryZ0MXOPERLOCResponse.parameters.QueryZ0MXOPERLOCResponse.
8              Z0MXOPERLOCSet.LOCATIONS.LOCATIONSPEC filter ($.ASSETATTRID
9              == 'A2A000000546' and (sizeof($.ALNVALUE) > 0 or sizeof($.
10             Z0DATEVALUE) > 0))) > 0)
11              then 'Cabina secondaria con telecontrollo'
12              else 'Cabina secondaria senza telecontrollo',
13          FEEDEROBJECTTYPE: if (sizeof(QueryZ0MXOPERLOC.
14              QueryZ0MXOPERLOCResponse.parameters.QueryZ0MXOPERLOCResponse.
15              Z0MXOPERLOCSet.LOCATIONS.LOCATIONSPEC filter ($.ASSETATTRID
16              == 'A2A000000546' and (sizeof($.ALNVALUE) > 0 or sizeof($.
17              Z0DATEVALUE) > 0))) > 0)
18              then 'ServiceSubstation'
19              else 'Vista',
20          MRID: payload.Record.MRID,
21          CIRCUIT: vars.Start.root.circuit
22      }
23  }

```

### DeepSeek-R1:

```

1  %dw 2.0
2  output application/xml
3  ns ns1 http://www.ibm.com/maximo/wsd1/Z0ESB_Z0MXOPERLOC_BYMX

```

```

4 ns ns2 http://www.ibm.com/maximo
5 ns pd http://xmlns.tibco.com/bw/process/2003
6 ns ns0 http://www.tibco.com/pe/GenerateErrorActivity/InputSchema
7
8
9 if (sizeof(
10     vars.QueryZ0MXOPERLOC.ns1#QueryZ0MXOPERLOCResponse.
11     parameters.ns2#QueryZ0MXOPERLOCResponse.ns2#Z0MXOPERLOCSet.
12     ns2#LOCATIONS.ns2#LOCATIONSPEC
13     filter (($.ns2#ASSETATTRID == "A2A000000546")
14         and ((($.ns2#ALNVALUE default "") != "") or (($.ns2#
15     Z0DATEVALUE default "") != "")))
16 ) > 0)
17 {
18     jdbcUpdateActivityInput: {
19         PSRTYPE: "Cabina secondaria con telecontrollo",
20         FEEDEROBJECTTYPE: "ServiceSubstation",
21         MRID: vars.current.Record.MRID,
22         CIRCUIT: vars.Start.root.circuit
23     }
24 }
25 else {
26     jdbcUpdateActivityInput: {
27         PSRTYPE: "Cabina secondaria senza telecontrollo",
28         FEEDEROBJECTTYPE: "Vista",
29         MRID: vars.current.Record.MRID,
30         CIRCUIT: vars.Start.root.circuit
31     }
32 }
33 }

```

### Llama-3.1-405B-Instruct-bf16:

```

1 %dw 2.0
2 output application/xml
3
4 if (sizeof(payload.QueryZ0MXOPERLOC.QueryZ0MXOPERLOCResponse
5     .parameters.QueryZ0MXOPERLOCResponse.Z0MXOPERLOCSet.LOCATIONS
6     .LOCATIONSPEC[?($.ASSETATTRID == 'A2A000000546' and (
7     stringLength($.ALNVALUE) > 0 or stringLength($.Z0DATEVALUE) >
8     0))]) > 0)
9     jdbcUpdateActivityInput: {
10         PSRTYPE: 'Cabina secondaria con telecontrollo',
11         FEEDEROBJECTTYPE: 'ServiceSubstation',
12         MRID: payload.current.Record.MRID,
13         CIRCUIT: payload.Start.root.circuit
14     }
15 else
16     jdbcUpdateActivityInput: {

```

```

13         PSRTYPE: 'Cabina secondaria senza telecontrollo',
14         FEEDEROBJECTTYPE: 'Vista',
15         MRID: payload.current.Record.MRID,
16         CIRCUIT: payload.Start.root.circuit
17     }

```

Tutti i modelli testati hanno generato un codice DataWeave che, a prima vista, potrebbe sembrare corretto o addirittura funzionante. Tuttavia, un'analisi più approfondita ha rivelato diverse problematiche che ne compromettono l'esecuzione. In molti casi, il codice presentava errori di sintassi che impedivano la corretta interpretazione da parte del compilatore, mentre in altri si riscontravano omissioni di parti essenziali, rendendo il risultato inutilizzabile.

Un aspetto particolarmente critico si manifesta quando il codice DataWeave ha come output un file XML. In questo caso, è fondamentale importare tutti i *namespace* presenti nel codice XSLT originale, qualora essi siano utilizzati. Questa difficoltà è stata riscontrata in particolare nei modelli **Llama** e **Claude**, i quali hanno generato codice privo delle dichiarazioni necessarie. Al contrario, i modelli **GPT-4o** e **DeepSeek** sono riusciti a includere correttamente i *namespace*, evitando questo errore specifico.

Tra i modelli analizzati, **DeepSeek** ha prodotto il codice che ha richiesto il minor numero di correzioni manuali. Nonostante ciò, il risultato ottenuto non era completamente funzionante e ha comunque richiesto un intervento per poter essere eseguito correttamente.

Questo esperimento ha dimostrato che l'utilizzo di un modello LLM per la conversione da XSLT a DataWeave è un'opzione percorribile, ma con limitazioni significative. In particolare, il tempo necessario per il *debugging* del codice generato può superare quello richiesto per scrivere il codice manualmente. Inoltre, sebbene vi sia una probabilità non nulla che il codice generato risulti eseguibile al primo tentativo, questo non garantisce che la sua struttura sia semanticamente corretta, introducendo un ulteriore livello di incertezza nell'automazione della conversione.

Questa strada è stata, per l'appunto, scartata per i motivi sopra citati. Il test successivo, invece, non partiva da un codice XSLT, bensì da un codice *DataWeave* che era stato già convertito da **Spectrum** in precedenza.

## Secondo esperimento

Analizziamo il prompt che è stato utilizzato per questo test:

*You will receive dataweave code that has comments inside and a structure to be compiled without giving errors, however, semantics*

*is not respected for this reason. You need to delete the comments, reset the if statements correctly, and prefix payload where needed to access the input value.*

Il prompt ha questa struttura e questa richiesta molto specifica perché Spectrum genera codice DataWeave seguendo una struttura ben definita e con delle regole che devono essere rispettate. Un esempio sono i commenti all'interno del codice che vengono riportati per fare in modo che il codice, una volta caricato sulla piattaforma di Mulesoft, possa essere compilato senza errori. Questo non significa che il codice sia funzionante, bensì che non ci siano errori di sintassi nella conversione delle funzioni e delle variabili. Lo sviluppatore successivamente andrà ad effettuare una serie di modifiche e correzioni per riportare la logica del codice originale accedendo alle variabili di input e output correttamente. Questa non è l'unica correzione da effettuare, ma ci sono una serie di modifiche che da apportare per ogni tipologia di file che viene caricato, quindi è richiesta una buona conoscenza di entrambi i linguaggi.

Il codice DataWeave dato in input al modello insieme al prompt è il seguente:

```
1 %dw 2.0
2 import substringBeforeLast from dw::core::Strings
3 ns ns4 http://www.tibco.com/pe/EngineTypes
4 ns ns0 http://www.tibco.com/namespaces/tnt/plugins/file
5 ns xsi http://www.w3.org/2001/XMLSchema-instance
6 ns ns7 http://www.tibco.com/schemas/FlussoPoste_root/
  Business/SharedResources/SchemaDefinitions/XMLSchema/Schema.
  xsd
7 ns pfx http://www.tibco.com/xmlns/ae2xsd/2002/05/ae/731/
  basic/functionModules
8 ns xsd http://www.w3.org/2001/XMLSchema
9 ns ns5 http://www.eutile.eu/standard/soa/messageStandard
10 ns ns6 http://www.tibco.com/pe/GenerateErrorActivity/
  InputSchema
11 ns pd http://xmlns.tibco.com/bw/process/2003
12 ns ns1 http://www.tibco.com/pe/DeployedVarsType
13 ns tib http://www.tibco.com/bw/xslt/custom-functions
14 ns ns3 http://www.tibco.com/pe/WriteToLogActivitySchema
15 ns xsl http://www.w3.org/1999/XSL/Transform
16 ns ns2 http://www.tibco.com/xmlns/ae2xsd/2002/05/ae/46C/
  basic/functionModules
17
18 fun getTargetValue_COMPL_1(payload) =
19   (if (true)
20     /* Adapter-Subscriber.ProcessStarterOutput.body.ns#EUMC.
  additionalMsgData.item[name == 'lastItem'].value == 'true' */
```

```

21         ("X")
22     else
23         {}
24     var elem_2 = "Adapter-Subscriber.ProcessStarterOutput.body.
ns#EUMC.ancestorBusinessId"
25     var elem_3 = "Adapter-Subscriber.ProcessStarterOutput.body.
ns#EUMC.additionalMsgData.item[name == 'TIPPAG '].value"
26     var elem_4 = "_globalVariables.nsl#GlobalVariables.Client"
27     var elem_5 = "Adapter-Subscriber.ProcessStarterOutput.body.
ns#EUMC.ancestorBusinessId"
28     var elem_6 = "Adapter-Subscriber.ProcessStarterOutput.body.
ns#EUMC.additionalMsgData.item[name == 'idRiga '].value"
29     var elem_7 = "Adapter-Subscriber.ProcessStarterOutput.body.
ns#EUMC.payload"
30     output application/xml
31     _____
32     (
33         root:
34         ns2#
35         caret_request_caret_Z__BAPI__LP__LOTTI_caret_Z__BAPI__LP__LOTTI
: {
36             COMPL: getTargetValue_COMPL_1(payload),
37             DATA: "tib:format-date('yyyyMMdd', current-date())",
38             ID__TBC: elem_2,
39             (
40                 (TIPPAG: elem_3) if (/* elem_3 / true), ITEM : item
: {MANDT : elem_4, ID__TBC : elem_5, (if (
41                 / Adapter-Subscriber.ProcessStarterOutput.body.ns#EUMC.
additionalMsgData.item[name == 'lastItem '].value == 'true' /
true) (NUM__RIGA : "substringBeforeLast(Adapter-Subscriber.
ProcessStarterOutput.body.ns#EUMC.additionalMsgData.item[name
== 'idRiga '].value,'-')") else (((NUM__RIGA : elem_6) if
42                 / elem_6 / true))), STATO : "TBC", ((RECORD : elem_7) if
43                 / elem_7 */
true)
44             )
45         }
46     )

```

*Nota:* Il codice fornito qui nel documento è stato formattato tramite la piattaforma di testing di Mulesoft: <https://dataweave.mulesoft.com/learn/dataweave>, poiché non ci sono strumenti open-source che permettano di formattare correttamente il codice DataWeave. Sono stati effettuati dei tentativi per utilizzare un formattatore di un altro linguaggio, vista la somiglianza a *Javascript*, ma questo portava a formattazioni errate del codice e scrittura errata.



I modelli che sono stati testati sono gli stessi del test precedente. Analizziamo i risultati ottenuti:

#### GPT-4o:

```

1 %dw 2.0
2 output application/xml
3
4 import substringBeforeLast from dw::core::Strings
5
6 fun getTargetValue_COMPL_1(payload) =
7     if (payload.AdapterSubscriber.ProcessStarterOutput.body.ns#
EUMC.additionalMsgData.item[name == 'lastItem'].value == '
true')
8         "X"
9     else
10        ""
11
12 var elem_2 = payload."AdapterSubscriber.ProcessStarterOutput.
body.ns#EUMC.ancestorBusinessId"
13 var elem_3 = payload."AdapterSubscriber.ProcessStarterOutput.
body.ns#EUMC.additionalMsgData.item[name == 'TIPPAG'].value"
14 var elem_4 = payload."_globalVariables.ns1#GlobalVariables.
Client"
15 var elem_5 = payload."AdapterSubscriber.ProcessStarterOutput.
body.ns#EUMC.ancestorBusinessId"
16 var elem_6 = payload."AdapterSubscriber.ProcessStarterOutput.
body.ns#EUMC.additionalMsgData.item[name == 'idRiga'].value"
17 var elem_7 = payload."AdapterSubscriber.ProcessStarterOutput.
body.ns#EUMC.payload"
18
19 ns ns4 http://www.tibco.com/pe/EngineTypes
20 ns ns0 http://www.tibco.com/namespaces/tnt/plugins/file
21 ns xsi http://www.w3.org/2001/XMLSchema-instance
22 ns ns7 http://www.tibco.com/schemas/FlussoPoste_root/Business/
SharedResources/SchemaDefinitions/XMLSchema/Schema.xsd
23 ns pfx http://www.tibco.com/xmlns/ae2xsd/2002/05/ae/731/basic/
functionModules
24 ns xsd http://www.w3.org/2001/XMLSchema
25 ns ns5 http://www.eutile.eu/standard/soa/messageStandard
26 ns ns6 http://www.tibco.com/pe/GenerateErrorActivity/InputSchema
27 ns pd http://xmlns.tibco.com/bw/process/2003
28 ns ns1 http://www.tibco.com/pe/DeployedVarsType
29 ns tib http://www.tibco.com/bw/xslt/custom-functions
30 ns ns3 http://www.tibco.com/pe/WriteToLogActivitySchema
31 ns xsl http://www.w3.org/1999/XSL/Transform
32 ns ns2 http://www.tibco.com/xmlns/ae2xsd/2002/05/ae/46C/basic/
functionModules
33
34 {

```

```

35     root: ns2#
36     caret_request_caret_Z__BAPI__LP__LOTTI_caret_Z__BAPI__LP__LOTTI
37     : {
38         COMPL: getTargetValue_COMPL_1(payload),
39         DATA: "tib:format-date('yyyymmdd', current-date())",
40         ID__TBC: elem_2,
41         TIPPAG: if (elem_3 != null) elem_3,
42         ITEM: {
43             MANDT: elem_4,
44             ID__TBC: elem_5,
45             NUM__RIGA: if (payload.Adapter-Subscriber.
46 ProcessStarterOutput.body.ns#EUMC.additionalMsgData.item[name
47 == 'lastItem'].value == 'true')
48                 substringBeforeLast(payload.Adapter-Subscriber.
49 ProcessStarterOutput.body.ns#EUMC.additionalMsgData.item[name
50 == 'idRiga'].value, '-')
51                 else
52                 if (elem_6 != null) elem_6,
53             STATO: "TBC",
54             RECORD: if (elem_7 != null) elem_7
55         }
56     }
57 }

```

### Claude-3-Opus:

```

1
2 %dw 2.0
3 output application/xml
4
5 import substringBeforeLast from dw::core::Strings
6
7 fun getTargetValue_COMPL_1(payload) =
8 (if (payload.Adapter-Subscriber.ProcessStarterOutput.body.ns#
9 EUMC.additionalMsgData.item[name == 'lastItem'].value == '
10 true') ("X") else {})
11
12 var elem_2 = payload.Adapter-Subscriber.ProcessStarterOutput.
13 body.ns#EUMC.ancestorBusinessId
14 var elem_3 = payload.Adapter-Subscriber.ProcessStarterOutput.
15 body.ns#EUMC.additionalMsgData.item[name == 'TIPPAG'].value
16 var elem_4 = payload._globalVariables.ns1#GlobalVariables'.
17 Client
18 var elem_5 = payload.Adapter-Subscriber.ProcessStarterOutput.
19 body.ns#EUMC.ancestorBusinessId
20 var elem_6 = payload.Adapter-Subscriber.ProcessStarterOutput.
21 body.ns#EUMC.additionalMsgData.item[name == 'idRiga'].value

```

```

15 var elem_7 = payload. 'Adapter-Subscriber '. ProcessStarterOutput .
    body. 'ns#EUMC'. payload
16
17 ns ns4 http://www.tibco.com/pe/EngineTypes
18 ns ns0 http://www.tibco.com/namespaces/tnt/plugins/ file
19 ns xsi http://www.w3.org/2001/XMLSchema-instance
20 ns ns7 http://www.tibco.com/schemas/FlussoPoste_root/Business/
    SharedResources/SchemaDefinitions/XMLSchema/Schema.xsd
21 ns pfx http://www.tibco.com/xmlns/ae2xsd/2002/05/ae/731/basic/
    functionModules
22 ns xsd http://www.w3.org/2001/XMLSchema
23 ns ns5 http://www.eutile.eu/standard/soa/messageStandard
24 ns ns6 http://www.tibco.com/pe/GenerateErrorActivity/InputSchema
25 ns pd http://xmlns.tibco.com/bw/process/2003
26 ns ns1 http://www.tibco.com/pe/DeployedVarsType
27 ns tib http://www.tibco.com/bw/xslt/custom-functions
28 ns ns3 http://www.tibco.com/pe/WriteToLogActivitySchema
29 ns xsl http://www.w3.org/1999/XSL/Transform
30 ns ns2 http://www.tibco.com/xmlns/ae2xsd/2002/05/ae/46C/basic/
    functionModules
31
32 {
33     root: {
34         "ns2#
35         caret_request_caret_Z__BAPI__LP__LOTTI_caret_Z__BAPI__LP__LOTTI
36         ": {
37             COMPL: getTargetValue_COMPL_1(payload),
38             DATA: tib#format-date('yyyyMMdd', current-date()),
39             ID__TBC: elem_2,
40             (TIPPAG: elem_3) if (elem_3 != null),
41             ITEM: {
42                 item: {
43                     MANDT: elem_4,
44                     ID__TBC: elem_5,
45                     NUM__RIGA: if (payload. 'Adapter-Subscriber '.
46                     ProcessStarterOutput.body. 'ns#EUMC'. additionalMsgData.item[
47                     name == 'lastItem'].value == 'true')
48                     substringBeforeLast(payload. 'Adapter-
49                     Subscriber '. ProcessStarterOutput.body. 'ns#EUMC'.
50                     additionalMsgData.item[name == 'idRiga'].value, '-')
51                     else
52                     elem_6,
53                     STATO: "TBC",
54                     (RECORD: elem_7) if (elem_7 != null)
55                 }
56             }
57         }
58     }
59 }

```

## DeepSeek-R1:

```
1      %dw 2.0
2      output application/xml
3      import substringBeforeLast from dw::core::Strings
4
5      // Namespaces realmente utilizzati nell output
6      ns ns2 http://www.tibco.com/xmlns/ae2xsd/2002/05/ae/46C/basic/
       functionModules
7      ns tib http://www.tibco.com/bw/xslt/custom-functions
8      ns ns1 http://www.tibco.com/pe/DeployedVarsType
9
10     // Funzione correttamente implementata
11     fun getTargetValue_COMPL_1(payload) =
12         if (payload.Adapter-Subscriber.ProcessStarterOutput.body.ns#
           EUMC.additionalMsgData.item[?($.name = 'lastItem')].value[0]
           == 'true')
13             "X"
14         else
15             null
16
17     // Variabili convertite in riferimenti diretti al payload
18     var elem_2 = payload.Adapter-Subscriber.ProcessStarterOutput.
       body.ns#EUMC.ancestorBusinessId
19     var elem_3 = payload.Adapter-Subscriber.ProcessStarterOutput.
       body.ns#EUMC.additionalMsgData.item[?($.name = 'TIPPAG')].
       value[0]
20     var elem_4 = vars._globalVariables.ns1#GlobalVariables.Client
21     var elem_5 = payload.Adapter-Subscriber.ProcessStarterOutput.
       body.ns#EUMC.ancestorBusinessId
22     var elem_6 = payload.Adapter-Subscriber.ProcessStarterOutput.
       body.ns#EUMC.additionalMsgData.item[?($.name = 'idRiga')].
       value[0]
23     var elem_7 = payload.Adapter-Subscriber.ProcessStarterOutput.
       body.ns#EUMC.payload
24
25     —
26     ns2#
       caret_request_caret_Z_BAPI_LP_LOTTI_caret_Z_BAPI_LP_LOTTI
       @"xmlns:ns2": "http://www.tibco.com/xmlns/ae2xsd/2002/05/ae
       /46C/basic/functionModules"): {
27         COMPL: getTargetValue_COMPL_1(payload),
28         DATA: tib:format-date('yyyyMMdd', now()),
29         ID_TBC: elem_2,
30         (TIPPAG: elem_3) if elem_3 != null,
31         ITEM: {
```

```

32     MANDT: elem_4,
33     ID__TBC: elem_5,
34     (NUM__RIGA: substringBeforeLast(elem_6, '-')) if (
payload.Adapter-Subscriber.ProcessStarterOutput.body.ns#EUMC.
additionalMsgData.item[?($.name == 'lastItem')].value[0] == '
true'),
35     (NUM__RIGA: elem_6) if (payload.Adapter-Subscriber.
ProcessStarterOutput.body.ns#EUMC.additionalMsgData.item[?($.
name == 'lastItem')].value[0] != 'true'),
36     STATO: "TBC",
37     (RECORD: elem_7) if elem_7 != null
38 }
39 }

```

### Llama-3.1-405B-Instruct-bf16:

```

1
2 %dw 2.0
3   output application/xml
4
5   import substringBeforeLast from dw::core::Strings
6
7   fun getTargetValue_COMPL_1(payload) =
8     (if (payload.Adapter-Subscriber.ProcessStarterOutput.body.ns
#EUMC.additionalMsgData.item[?($.name == 'lastItem')].value
== 'true') "X" else "")
9
10    var elem_2 = payload.Adapter-Subscriber.ProcessStarterOutput
.body.ns#EUMC.ancestorBusinessId
11    var elem_3 = payload.Adapter-Subscriber.ProcessStarterOutput
.body.ns#EUMC.additionalMsgData.item[?($.name == 'TIPPAG')].
value
12    var elem_4 = payload._globalVariables.ns1#GlobalVariables.
Client
13    var elem_5 = payload.Adapter-Subscriber.ProcessStarterOutput
.body.ns#EUMC.ancestorBusinessId
14    var elem_6 = payload.Adapter-Subscriber.ProcessStarterOutput
.body.ns#EUMC.additionalMsgData.item[?($.name == 'idRiga')].
value
15    var elem_7 = payload.Adapter-Subscriber.ProcessStarterOutput
.body.ns#EUMC.payload
16
17    ns ns4 http://www.tibco.com/pe/EngineTypes
18    ns ns0 http://www.tibco.com/namespaces/tnt/plugins/file
19    ns xsi http://www.w3.org/2001/XMLSchema-instance
20    ns ns7 http://www.tibco.com/schemas/FlussoPoste_root/
Business/SharedResources/SchemaDefinitions/XMLSchema/Schema.
xsd

```

```

21 ns pfx http://www.tibco.com/xmlns/ae2xsd/2002/05/ae/731/
basic/functionModules
22 ns xsd http://www.w3.org/2001/XMLSchema
23 ns ns5 http://www.eutile.eu/standard/soa/messageStandard
24 ns ns6 http://www.tibco.com/pe/GenerateErrorActivity/
InputSchema
25 ns pd http://xmlns.tibco.com/bw/process/2003
26 ns ns1 http://www.tibco.com/pe/DeployedVarsType
27 ns tib http://www.tibco.com/bw/xslt/custom-functions
28 ns ns3 http://www.tibco.com/pe/WriteToLogActivitySchema
29 ns xsl http://www.w3.org/1999/XSL/Transform
30 ns ns2 http://www.tibco.com/xmlns/ae2xsd/2002/05/ae/46C/
basic/functionModules
31
32 root : ns2#
caret_request_caret_Z__BAPI__LP__LOTTI_caret_Z__BAPI__LP__LOTTI
: {
33     COMPL : getTargetValue_COMPL_1(payload),
34     DATA : tib:format-date('yyyyMMdd', current-date()),
35     ID__TBC : elem_2,
36     (TIPPAG : elem_3) if (elem_3 != null),
37     ITEM : item : {
38         MANDT : elem_4,
39         ID__TBC : elem_5,
40         (NUM__RIGA : substringBeforeLast(elem_6, '-')) if (
elem_6 != null and payload.Adapter-Subscriber.
ProcessStarterOutput.body.ns#EUMC.additionalMsgData.item[?($.
name == 'lastItem')].value == 'true'),
41         (NUM__RIGA : elem_6) if (elem_6 != null),
42         STATO : "TBC",
43         (RECORD : elem_7) if (elem_7 != null)
44     }
45 }

```

Tra i quattro modelli analizzati, **DeepSeek-R1** ha prodotto il codice DataWeave più corretto e strutturato. Questo modello ha dimostrato una migliore comprensione della sintassi DataWeave, utilizzando correttamente i namespace e gestendo in modo appropriato le condizioni e le trasformazioni dei dati.

Ad esempio, DeepSeek-R1 ha utilizzato correttamente la funzione `substringBeforeLast` con la sintassi `item[?$(.name == 'lastItem')]`, mentre **GPT-4o** ha commesso l'errore di usare una sintassi non valida come `item[name == 'lastItem']`, che potrebbe causare errori di runtime. Inoltre, **Claude-3-Opus** ha introdotto un uso improprio delle virgolette, ad esempio scrivendo `Adapter-Subscriber` invece di gestire correttamente

i riferimenti ai campi.

Llama ha prodotto un codice che potrebbe avvicinarsi a quello di DeepSeek, ma ha commesso errori di sintassi che potrebbero causare errori di compilazione.

Anche in questo caso, si ha effettivamente un codice che potrebbe essere compilato senza errori, e il suo output potrebbe essere corretto, rischiando meno rispetto ad una conversione diretta del codice XSLT. Tuttavia, il codice generato dai modelli LLM non è ancora pronto per essere utilizzato direttamente in un progetto, poiché richiede comunque una serie di correzioni manuali per garantire la compilazione corretta o il corretto funzionamento della logica.

Questo approccio è molto più applicabile rispetto al precedente, ma comunque richiede l'utilizzo di modelli più avanzati e costosi, rispetto a quelli citati nel Capitolo 2, dove si cita Gpt-4o-mini 2.6.1.

Si è optato per abbandonare l'utilizzo di modelli LLM per la conversione da codice XSLT a DataWeave, poiché, seppur i risultati fossero promettenti, la fase di revisione del codice generato continua a richiedere un intervento manuale notevole.

## 3.4 Approccio basato su fine-tuning di modelli pre-addestrati

Un secondo approccio che è stato preso in considerazione è l'utilizzo di modelli pre-addestrati successivamente addestrati su un dataset specifico per la conversione di codice XSLT in DataWeave. Questo approccio è basato sul fine-tuning, una tecnica di apprendimento automatico che consiste nell'addestrare un modello pre-addestrato su un dataset specifico per adattarlo a un compito particolare.

**Definizione 3.1. (Fine-Tuning):**

Il *fine-tuning* in ambito LLM (Large Language Models) rappresenta una tecnica di transfer learning dove un modello pre-addestrato su dati generici viene riaddestrato su un dominio specifico utilizzando un dataset specializzato [1].

La procedura di fine-tuning consiste nel minimizzare una funzione di *loss* che misura l'errore tra le predizioni del modello e i target del dataset di addestramento. Formalmente, il fine-tuning può essere definito come un problema di ottimizzazione:

$$\min_{\theta} \sum_{(x,y) \in D} \mathcal{L}(f_{\theta}(x), y) \quad (3.1)$$

dove  $D$  è il dataset di input-output-code triples,  $f_{\theta}$  il modello linguistico, e  $\mathcal{L}$  la funzione di loss per token prediction.

Poiché un task come il *fine-tuning* richiede una grande quantità di dati, una conoscenza approfondita del dominio e risorse computazionali significative, si è preso in esame inizialmente uno studio di Salesforce [3].

Lo studio Salesforce [3] utilizza una variante efficiente detta *LoRA* (Low-Rank Adaptation).

LoRA è un metodo di fine-tuning che sfrutta la bassa dimensionalità del task-specific embedding space per ridurre il numero di parametri adattati, garantendo una rapida convergenza e una maggiore generalizzazione [3].

### 3.4.1 Analisi Critica dello Studio Salesforce

Lo studio si basa sul voler mettere a confronto le performance dei modelli *closed-source*, come GPT-4, Claude, usando le relative API, con modelli *open-source*, più piccoli e utilizzando la tecnica di fine-tuning LoRA.

La metrica per valutare i modelli è la **pass@k**



**Definizione 3.2.** (**pass@k**): La metrica **pass@k** è un estimatore non distorto della probabilità che almeno uno tra i primi  $k$  codici generati superi i test. Data una generazione di  $n$  campioni ( $n \geq k$ ) con  $c$  risposte corrette.

$$pass@k := E\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \quad (3.2)$$

con:

- $\binom{n}{k}$  il numero di combinazioni di  $n$  elementi presi  $k$  alla volta
- $\binom{n-c}{k}$  il numero di modi per scegliere campioni tutti errati
- $\frac{\binom{n-c}{k}}{\binom{n}{k}}$  la probabilità che tutti i campioni siano errati
- complemento a 1 della probabilità di fallimento

Un esempio numerico, per capire meglio:

- $n = 5$  campioni generati
- $c = 2$  risposte corrette
- $k = 3$  campioni considerati

$$pass@3 = 1 - \frac{\binom{3}{3}}{\binom{5}{3}} = 1 - \frac{1}{10} = 0.9 \quad (3.3)$$

Questo ci dice che c'è il 90% di probabilità che almeno uno dei primi 3 campioni generati sia corretto.

**Definizione 3.3** (N-Shot). Il termine **n-shot** si riferisce al numero di esempi forniti a un modello linguistico durante il processo di inferenza. In una configurazione **zero-shot**, il modello deve generare una risposta senza alcun esempio precedente. In una configurazione **one-shot**, viene fornito un solo esempio precedente come contesto, mentre in una configurazione **two-shot**, vengono forniti due esempi. Questi contesti aiutano il modello a comprendere meglio il compito richiesto.

L'analisi confronta le prestazioni di diversi modelli linguistici, tra cui GPT-3.5, GPT-4, Claude 1.3 e Claude 2, nella generazione di codice DataWeave. I risultati mostrano che GPT-4 è il modello più affidabile, con un punteggio medio di Pass@1 del 0.434 in configurazione a two-shot. In

generale, i modelli tendono a migliorare nelle configurazioni one-shot rispetto a zero-shot, ma non si osservano miglioramenti significativi passando da one-shot a two-shot.

GPT-3.5 e GPT-4 superano notevolmente Claude 1.3 e Claude 2, soprattutto in termini di percentuali di compilazione, indicando una maggiore capacità di generare codice eseguibile. Anche se GPT-3.5 e Claude 2 ottengono risultati simili per il codice compilato, GPT-4 eccelle nettamente in questo aspetto. È interessante notare che la versione di GPT-4 rilasciata nel giugno 2023 ha mostrato un miglioramento significativo rispetto alla versione di marzo 2023, mentre GPT-3.5 è rimasto stabile.

Prima di parlare del processo utilizzato per il fine-tuning, è importante citare che tutti i modelli *open-source* utilizzati avevano una percentuale  $<1\%$  di pass@k prima del fine-tuning.

Il **dataset** utilizzato per questa analisi è stato creato utilizzando discussioni sul forum di Mulesoft e dalla documentazione di DataWeave.

I 3 modelli presi in considerazione per il fine-tuning sono:

- Bloomz-7b1: Modello multilingue generalista
- XGen-4k: Specializzato in codice con contesto a 4k token
- XGen-Instruct-8k: Variante instruction-tuned

Il processo di fine-tuning ha portato un miglioramento sostanziale a tutti i modelli open-source, passando dal  $<1\%$  di pass@1 ad avere almeno un 15% di pass@1. Il modello migliore è stato il XGen-4k che ha raggiunto un pass@1 del 0.316 e un pass@2 del 0.364. [3]

Tuttavia, questo processo ha comunque dei costi elevati e dei vincoli di tempo dipendenti dalla potenza di calcolo disponibile. Inoltre, il fine-tuning richiede un dataset di input-output-code triples, che potrebbe non essere disponibile o difficile da ottenere, come nel caso di codice XSLT a DataWeave.

I risultati finali mostrano che nonostante il processo di fine-tuning, il modello di OpenAI GPT-4 rimane il più affidabile per la generazione di codice DataWeave, con un pass@1 del 0.434 in configurazione two-shot, secondo questa analisi.

### 3.4.2 Considerazioni

Nel contesto della nostra analisi, è fondamentale considerare diversi aspetti che rendono il nostro caso diverso dagli studi condotti sulla generazione di codice condotta da Salesforce. In particolare, la conversione da XSLT a

DataWeave implica una serie di sfide specifiche che non sono sempre ben rappresentate nei modelli pre-addestrati o nelle tecniche di fine-tuning standard.

I modelli linguistici generalisti come GPT-3.5 e GPT-4, pur mostrando buone prestazioni in generale, potrebbero non essere sufficientemente specializzati per gestire le peculiarità di questa conversione. Anche quando adottiamo tecniche di *fine-tuning*, il modello deve essere raffinato su un dataset altamente specifico, il che aggiunge ulteriori livelli di complessità.

Inoltre, i codici XSLT possono essere estremamente complessi, con regole di trasformazione complesse e dipendenze multiple. La traduzione precisa di queste regole in DataWeave richiede non solo una comprensione approfondita dei due linguaggi, ma anche la capacità di gestire le ambiguità e le variazioni nel codice sorgente. Questo livello di dettaglio è difficile da raggiungere anche con modelli più grandi sottoposti a fine-tuning. Pertanto, l'efficacia di tali modelli nella nostra applicazione specifica potrebbe essere limitata.

Un altro aspetto importante da considerare è la poca utilità, nel nostro specifico caso, di fine-tunare modelli più piccoli, come quelli open-source esaminati nello studio Salesforce. Nonostante il fine-tuning abbia portato miglioramenti significativi, questi modelli hanno comunque ottenuto risultati inferiori rispetto ai modelli closed-source più grandi e sofisticati come GPT-4.

Per quanto riguarda la conversione da XSLT a DataWeave, il dataset dovrebbe coprire una vasta gamma di casi d'uso e configurazioni possibili, il che aumenta ulteriormente la complessità e i costi del processo.

Quando si parla di modelli grandi come GPT-4o, i costi associati al fine-tuning diventano ancora più evidenti. Il modello di OpenAI ha un'enorme quantità di parametri, il che lo rende estremamente potente ma anche estremamente costoso da addestrare e mantenere. I costi computazionali per fine-tunare un modello di questa scala sono proibitivi se non adeguatamente stimati e gestiti con attenzione.

Oltre ai costi finanziari, ci sono anche vincoli temporali e di risorse. Il fine-tuning di un modello grande richiede una potenza di calcolo considerevole e un tempo di elaborazione prolungato, con possibile rischio di reiterazione se non accuratamente pianificato e monitorato. Inoltre, anche se fosse possibile fine-tunare un modello grande come GPT-4o, non c'è garanzia che le prestazioni migliorino in modo sostanziale rispetto alla versione originale del modello, specialmente per task altamente specifici come nel caso della nostra conversione. Questo è stato il fattore che ha ufficialmente decretato questa soluzione come non ottimale per il nostro caso di studio.

Infine, è importante considerare il contesto dell'attuale strumento disponibile per la conversione da XSLT a DataWeave, ovvero Spectrum. Spectrum è già in grado di convertire il codice in modo corretto e funzionale, soddisfacendo le esigenze della maggior parte degli utenti. Quindi, qual è il senso di

investire risorse significative per sviluppare nuovi modelli o migliorare quelli esistenti, se il risultato finale non supera o almeno eguaglia le prestazioni di Spectrum? Se un nuovo modello non riesce a produrre codice funzionante o migliore di quello generato da Spectrum, il suo valore pratico rimane limitato.

### 3.5 LLM per Micro Task

Abbiamo parlato di task particolarmente complessi per quanto riguarda la generazione di codice DataWeave da codice XSLT. Tuttavia, esistono anche micro task all'interno di questo processo che potrebbero beneficiare dell'utilizzo di modelli di LLM. Spectrum durante il processo di conversione, genera rappresentazioni specifiche in DataWeave per vari costrutti presenti nel codice XSLT originale. Ad esempio, quando ci sono riferimenti a variabili in XSLT, questi vengono tradotti in definizioni di variabili in DataWeave come:

```
1 var elem_1 = "Start.definizione.Pagamento.POS.1"  
2 var elem_2 = "Start.costi.fine.periodo"
```

Questo schema continua fino alla  $n$ -esima variabile. Benché questa traduzione sia funzionale, i nomi delle variabili risultanti possono essere poco descrittivi e difficili da interpretare per gli sviluppatori che devono leggere e mantenere il codice.

Un'idea interessante per dare un valore aggiunto a questa tecnologia è l'utilizzo di LLM per generare nomi contestuali per le variabili. L'obiettivo è migliorare la leggibilità, manutenibilità e comprensione del codice generato da Spectrum. Questo approccio si basa sull'idea di passare una lista di variabili raccolte da uno script all'API di un modello come GPT-4o-mini, che restituisce una lista corrispondente con nomi di variabili più significativi.

Ad esempio, se lo script estrae le seguenti variabili dal codice convertito:

```
1 var elem_1 = "customer_name"  
2 var elem_2 = "order_id"  
3 var elem_3 = "product_price"
```

L'API di GPT-4 potrebbe restituire nomi di variabili più descrittivi come:

```
1 var customerName = "customer_name"  
2 var orderId = "order_id"  
3 var productPrice = "product_price"
```

Questo processo non solo rende il codice più leggibile, ma anche facilita la manutenzione futura, poiché i nomi delle variabili riflettono meglio il loro contenuto e scopo, rispetto ad un nome generico come "elem\_1" o "elem\_2".

Le attività di assegnazione di nomi alle variabili sono meno strict e meno propense agli errori rispetto ad altre operazioni di generazione di codice. Tuttavia, è importante notare che anche in queste micro task, il prompt fornito al modello è cruciale per garantire output accurati e coerenti. Un prompt ben progettato può guidare il modello verso risultati desiderati, mentre un prompt mal formulato può portare a confusioni o output non ottimali.

**Definizione 3.4** (Prompt Engineering). Il Prompt Engineering consiste nel concepire e strutturare input precisi per gli LLM. Questo si basa su seguire alcune linee guida e best practices per ottenere risultati coerenti e affidabili, dipendentemente dal tipo di task che si vuole svolgere o dal modello che si sta utilizzando.

### 3.5.1 Implementazione

Per implementare la funzionalità di generazione di nomi di variabili descrittivi, è stato necessario sviluppare un modulo aggiuntivo dedicato alla gestione delle chiamate alle API di OpenAI. Questo modulo, denominato `openai_api.py`, è stato progettato inizialmente per supportare esclusivamente un singolo prompt relativo alla conversione dei nomi delle variabili. Tuttavia, considerando l'obiettivo futuro di estendere le funzionalità del modulo e integrarlo con altre feature di Spectrum, si è deciso di rendere la sua architettura più modulare e flessibile.

Il modulo è stato organizzato in modo tale da utilizzare un dizionario per memorizzare i diversi prompt che possono essere richiamati in base alle esigenze specifiche. Inoltre, è stato definito un *prompt base* statico, utile per fornire un contesto generale alle richieste inviate al modello. Il prompt base è formulato come segue:

"I'll provide a DataWeave code, but my question may not relate to it. Answer my question regardless of the code unless I specify otherwise."

Questa formulazione è stata pensata per garantire che il modello possa gestire richieste diverse dal solo codice DataWeave, lasciando aperta la possibilità di espandere il modulo con nuove funzionalità.

Quando si interagisce con le API di OpenAI, è fondamentale distinguere tra due tipologie di prompt: *system prompt* e *user prompt*. Il *system prompt* è definito dagli sviluppatori e serve a stabilire il comportamento generale del

modello, mentre il *user prompt* contiene l'input specifico fornito dall'utente. Nel nostro caso, il *system prompt* è stato configurato per guidare il modello verso il compito desiderato, mentre il *user prompt* spesso include il codice DataWeave, anche se non potrebbe essere diverso in base al task richiesto.

Un esempio di prompt specifico per la generazione di nomi di variabili è il seguente:

```
"variable_names": """"You are an assistant that takes
generic variable names and provides meaningful names
in JSON format based on their content. Respond only
with a JSON object where the keys are the original
variable names, and the values are the suggested
meaningful names in English, with no spaces and
camelCase and be quite short. Avoid generic terms
like 'element' and avoid the ```json format, just the
object itself."""""
```

Questo prompt è stato accuratamente progettato per garantire che il modello restituisca un output strutturato e conforme alle specifiche richieste. In particolare, il formato JSON è stato scelto per facilitare l'integrazione con il resto dell'applicazione, mentre le indicazioni relative al formato *camelCase* e alla brevità dei nomi mirano a migliorare la leggibilità del codice generato.

Inizialmente, si era considerato l'utilizzo di un prompt più semplice, privo di vincoli rigidi sul formato dell'output. Tuttavia, durante le fasi di testing, è emerso che un prompt meno strutturato poteva portare a risposte non conformi alle aspettative, compromettendo la coerenza del risultato finale. Per questo motivo, si è optato per una formulazione più dettagliata, che definisse chiaramente lo schema atteso dal modello.

Inoltre, si era valutata la possibilità di adottare un approccio *few-shot* o *one-shot*, fornendo al modello alcuni esempi di input-output per guidarne il comportamento. Tuttavia, i test condotti hanno dimostrato che il modello era in grado di fornire risposte accurate anche senza esempi, grazie alla chiarezza e alla precisione del prompt.

### 3.5.2 Codice e esempi

Solitamente quando si utilizzano le API di OpenAI lo si fa attraverso la libreria apposita *openai* che fa da wrapper per le chiamate da fare ai vari servizi. Tuttavia, per un problema legato all'architettura e al modo in cui viene distribuito l'applicativo, non è stato possibile utilizzare questa libreria, poiché internamente aveva bisogno di librerie che non erano compatibili con il processo di creazione dell'eseguibile di Spectrum. Questo perché l'applicativo

*Streamlit* viene convertito in eseguibile utilizzando *stlite*, che esegue il codice *Streamlit* in *Pyodide* nel browser, per poi essere convertito in eseguibile utilizzando *Electron*. I dettagli di queste librerie sono stati volontariamente omessi poiché non rilevanti. L'unico dettaglio importante è che la libreria *openai* non era compatibile con *Pyodide* per i motivi precedentemente citati.

Per risolvere questo problema è stato necessario implementare la chiamate alle API tramite un'altra libreria, *urllib3*. Questa libreria permette di avere uno *user-friendly* client HTTP, che permette di fare chiamate senza troppi problemi. Il codice del modulo `openai_api.py` per la gestione delle chiamate e delle risposte alle API di OpenAI è il seguente:

```

1  def call_openai_api(api_key, message, prompt):
2      """
3      Call the OpenAI API with the given message and return the
4      JSON result.
5
6      Parameters:
7          api_key (str): The API key to use for the request.
8          message (str): The message to send to the API.
9
10     Returns:
11         dict: The JSON result from the API, or an error message
12         if an error occurred.
13     """
14     url = "https://api.openai.com/v1/chat/completions"
15     headers = {
16         "Authorization": f"Bearer {api_key}",
17         "Content-Type": "application/json",
18     }
19
20     base_prompt = """ I ll provide a DataWeave code, but my
21     question may not relate to it. Answer my question regardless
22     of the code unless I specify otherwise." """
23     payload = {
24         "model": "gpt-4o-mini",
25         "messages": [
26             {
27                 "role": "system",
28                 "content": f"""{base_prompt} \n {prompt}""",
29             },
30             {
31                 "role": "user",
32                 "content": f"""{message}""",
33             },
34         ],
35     }
36
37     http = urllib3.PoolManager()

```

```

34
35     try:
36         response = http.request(
37             "POST",
38             url,
39             body=json.dumps(payload),
40             headers=headers,
41         )
42
43         if response.status == 200:
44             # Decodifica il risultato JSON
45             result = json.loads(response.data.decode("utf-8"))
46
47             # Verifica che la risposta contenga "choices"
48             if "choices" in result and len(result["choices"]) >
0:
49                 message_content = result["choices"][0]["message"]
50                 ]["content"]
51
52                 # Verifica che il contenuto non sia vuoto
53                 if message_content:
54                     try:
55                         # Restituisce il JSON decodificato
56                         return json.loads(message_content) if
prompt == get_prompt("variable_names") else message_content
57                     except json.JSONDecodeError as e:
58                         return {
59                             "error": f"Errore nel parsing del
JSON: {e}",
60                             "raw_content": message_content,
61                         }
62                     else:
63                         return {"error": "Nessuna risposta trovata"}
64                     else:
65                         return {"error": "Nessuna scelta trovata nella
risposta"}
66                     else:
67                         return {
68                             "error": f"Errore nell'API, status code: {
response.status}",
69                             "response_text": response.data.decode("utf-8"),
70                         }
71
72     except Exception as e:
73         return {"error": f"Errore nella richiesta: {e}"}

```

Un esempio di *input* e *output* per la generazione di nomi di variabili è il



seguinte:

```
1 {  
2 {  
3   {  
4     "elem_1": "$GetRIRE_NOMECODICE/response/group[current() /  
PDF=PDF]/NOMECODICE",  
5     "elem_2": "current.Record.SERVICESTATE",  
6     "elem_3": "current.Record.MRID",  
7     "elem_5": "GetMeterLocation.response.MeterLocation",  
8     "elem_6": "SIDES.resultSet.*Record"  
9   }  
10 }
```

```
1 {  
2 {  
3   {  
4     "elem_1": "pdfNomCodicePath",  
5     "elem_2": "serviceState",  
6     "elem_3": "meterId",  
7     "elem_5": "meterLocation",  
8     "elem_6": "recordSet"  
9   }  
10 }
```

Qui, ad esempio, nello *user prompt* non viene passato l'intero codice DataWeave, ma semplicemente la lista di variabili che vengono dichiarate all'interno del codice. In altri casi, invece, tutto il codice viene mandato come input al modello. Quello che succede successivamente a questo step è la sostituzione nel codice DataWeave originale dei nomi delle variabili con quelli generati dal modello.



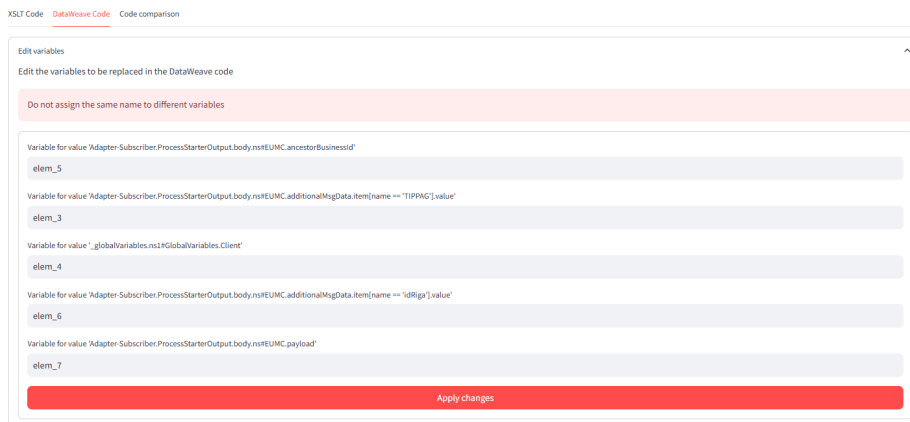


Figura 3.7: UI per la modifica manuale dei nomi delle variabili

### 3.5.3 Alternativa con LLM locale

In un tentativo di esplorare alternative ai modelli cloud-based, è stato valutato l'utilizzo di un modello linguistico locale per la generazione di codice. In particolare, è stato testato il modello **Yi-Coder-1.5B**<sup>1</sup>, disponibile su Hugging Face, che è stato addestrato specificamente per compiti di generazione di codice.

Il test ha riguardato la generazione di un semplice algoritmo in Python, come il *bubble sort*, scelto poiché si presume che il modello abbia avuto accesso a numerosi esempi di questo tipo di codice durante il training. Tuttavia, anche per una generazione apparentemente semplice come questa, il tempo richiesto è stato notevole: circa 10 minuti sulla macchina fornita dall'azienda, che era dotata di CPU ma priva di GPU. Questo tempo di elaborazione rende l'utilizzo del modello poco pratico per task più complessi, come la generazione di codice DataWeave, che richiede un livello di precisione e comprensione del contesto superiore.

Quando si è tentato di utilizzare il modello per generare codice DataWeave, le prestazioni sono ulteriormente peggiorate, con output contenenti errori critici che ne compromettevano l'utilizzabilità.

Sebbene l'uso di una GPU avrebbe potuto ridurre i tempi di generazione, il modello stesso, essendo relativamente piccolo, non sembra essere sufficientemente robusto per gestire task complessi come quelli richiesti dal nostro caso d'uso. Pertanto, considerando sia i tempi di esecuzione non ottimali sia la qualità insoddisfacente del codice generato, si è deciso di scartare questa alternativa.

<sup>1</sup><https://huggingface.co/01-ai/Yi-Coder-1.5B>

### 3.5.4 Considerazioni

L'utilizzo di LLM per generare nomi di variabili descrittivi è un esempio di come queste tecnologie possano essere applicate a task specifici all'interno di un processo più ampio. Questo approccio mira a migliorare piccole parti del codice generato da Spectrum, rendendolo più leggibile e comprensibile per gli sviluppatori, senza dover ricorrere a modifiche manuali. Il funzionamento di questo modulo è legato al fatto che qui il problema legato alla correttezza e alla compilazione del codice non è rilevante, perciò il grado di accettabilità è molto più alto rispetto ad altri task più ampi.

## 3.6 LLM per Macro-Task

Uno dei principali ostacoli che emergono durante il processo di conversione da XSLT a DataWeave riguarda la gestione delle espressioni XPath. Queste espressioni sono fondamentali per accedere a nodi o attributi all'interno di un documento XML, e rappresentano un elemento chiave nella logica del codice generato da Spectrum. Tuttavia, uno dei limiti intrinseci di Spectrum è la sua incapacità di distinguere tra particolari stringhe XPath. Di conseguenza, tutte queste espressioni vengono trattate come stringhe generiche, lasciando all'utente il compito di identificare manualmente quali richiedono l'aggiunta del prefisso `payload..`.

Questo processo manuale è particolarmente oneroso e soggetto a errori, poiché può coinvolgere decine o addirittura centinaia di espressioni all'interno di un singolo file DataWeave. Gli sviluppatori devono analizzare attentamente ogni espressione per determinare se necessita del prefisso `payload..`. Questo lavoro ripetitivo non solo richiede un notevole dispendio di tempo, ma aumenta anche il rischio di errori umani, specialmente quando si lavora su file di grandi dimensioni o con logiche complesse. Inoltre, la mancanza di strumenti automatizzati per supportare questa fase rende il processo ancora più frustrante e inefficiente.

L'obiettivo principale dell'utilizzo di un modello linguistico (LLM) in questo contesto è quello di automatizzare l'individuazione e la modifica delle espressioni XPath che richiedono l'aggiunta del prefisso `payload..`. L'idea è quella di delegare al modello il compito di analizzare il codice DataWeave generato da Spectrum, identificando automaticamente le espressioni XPath e classificandole in base alla necessità del prefisso. In particolare, il modello deve essere in grado di:

- **Identificare le Espressioni XPath:** Queste espressioni possono assumere diverse forme, come percorsi assoluti (ad esempio,

`/root/element`), percorsi relativi (ad esempio, `./child`) o funzioni che accedono a nodi specifici (ad esempio, `node()[1]`).

- **Classificare le Espressioni:** Ad esempio, un'espressione come `/order/id` richiede il prefisso `payload.`, mentre un'espressione già contenuta in una funzione DataWeave che ha come parametro il `payload` non necessita di ulteriori modifiche.
- **Applicare le Modifiche Necessarie:** Il modello deve essere in grado di applicare le modifiche direttamente al codice, aggiungendo il prefisso `payload.` solo dove appropriato. Questo passaggio può anche essere delegato successivamente a Spectrum, ma anche questa idea potrebbe avere dei limiti.

L'automazione di questi passaggi mira a ridurre significativamente il carico di lavoro sugli sviluppatori, migliorando la qualità del codice finale e riducendo il rischio di errori umani. Inoltre, l'uso di un LLM consente di scalare il processo a file di dimensioni moderate, garantendo una maggiore efficienza rispetto all'approccio manuale.

### 3.6.1 Complessità del Task e Limiti del Modello

Nonostante l'ampia finestra di contesto offerta da modelli come GPT-4o-mini, che permette di elaborare grandi quantità di codice in una singola chiamata API, il task di individuazione e modifica delle espressioni XPath presenta diverse sfide tecniche. Innanzitutto, la complessità del codice DataWeave generato da Spectrum può variare notevolmente, influenzando la capacità del modello di interpretare correttamente le espressioni XPath. File con strutture annidate, logiche condizionali complesse o funzioni personalizzate possono confondere il modello, portando a risultati incoerenti.

Inoltre, abbiamo riscontrato diversi problemi durante i test:

- **Selezione di Stringhe Sbagliate:** Il modello talvolta identifica erroneamente espressioni che non richiedono il prefisso `payload.`, aggiungendolo in modo improprio.
- **Selezione di Stringhe Incoerenti:** In alcuni casi, il modello classifica in modo inconsistente espressioni simili, trattandole diversamente in contesti analoghi.
- **Mancanza di Alcune Stringhe Importanti:** Infine, il modello talvolta trascura completamente alcune espressioni XPath che invece richiedono il prefisso `payload.`

Questi limiti dimostrano che, nonostante l'automazione, il modello non è ancora completamente affidabile e richiede una revisione manuale del codice, anche se in misura ridotta rispetto al processo originale.

### 3.6.2 Analisi Critica

L'uso di un LLM per automatizzare l'aggiunta del prefisso `payload`. presenta vantaggi e limiti che devono essere attentamente valutati. Anche se il modello non è perfetto, riduce significativamente il numero di modifiche manuali richieste. Inoltre l'approccio è facilmente scalabile a file di dimensioni moderate, grazie alla capacità del modello di elaborare grandi quantità di testo in una singola chiamata. Tuttavia, ci sono anche importanti limitazioni da considerare. Nonostante l'automazione, il modello commette errori che richiedono comunque una revisione manuale. Questo solleva dubbi sulla convenienza dell'approccio, soprattutto in termini di costi computazionali e di accesso alle API. La creazione di un prompt efficace è cruciale, e anche con esempi few-shot non è garantito che il modello produca risultati ottimali.

Inoltre, è importante considerare che l'uso di un LLM per questo tipo di task non è privo di costi. Le chiamate alle API di modelli come GPT-4o-mini possono diventare significative, soprattutto quando si lavora su file di grandi dimensioni o su progetti con numerosi file da convertire. Pertanto, è essenziale valutare attentamente il rapporto costo-beneficio prima di adottare questa soluzione su larga scala.

### 3.6.3 Considerazioni

L'utilizzo di un LLM per automatizzare questo task rappresenta un approccio innovativo ma non completamente affidabile. Mentre il modello riesce a ridurre il carico di lavoro sugli sviluppatori, i problemi legati alla selezione errata o incompleta delle espressioni XPath dimostrano che questa soluzione non è ancora completamente matura per sostituire completamente il processo manuale. Tuttavia, l'automazione rimane un'opzione interessante per ridurre i tempi in modo significativo, ma probabilmente richiederebbe dei costi maggiori per essere implementata con piena efficacia, magari utilizzando dei modelli più grandi e sofisticati come GPT-4o o o1, che è il modello di punta di OpenAI.

### 3.6.4 Soluzione algoritmica

Uno degli aspetti critici nella migrazione di API da XSLT a DataWeave è la necessità di testare il corretto funzionamento della nuova API utilizzando un

payload di input che mimetizzi un comportamento tipico del sistema. Questo payload, che può essere in formato XML o JSON, rappresenta i dati che l'API riceverà durante l'esecuzione reale. Gli sviluppatori spesso dispongono già di file di payload utilizzati nei sistemi legacy, e si è pensato di sfruttare questa opportunità per estrarre ulteriori informazioni utili dal contenuto di questi file.

Nel caso specifico del payload XML, è stata implementata una procedura automatizzata per esplorare e navigare la struttura gerarchica del documento, estraendo tutti i possibili path dei nodi. Questi path vengono rappresentati come stringhe separate da punti, che riflettono la gerarchia dei nodi all'interno del documento XML. Ad esempio, consideriamo il seguente frammento di XML:

```
<order>
  <id>12345</id>
  <customer>
    <name>John Doe</name>
    <address>
      <city>New York</city>
      <zip>10001</zip>
    </address>
  </customer>
  <items>
    <item>
      <product>Widget</product>
      <quantity>3</quantity>
    </item>
  </items>
</order>
```

Dall'analisi di questo documento XML, è possibile estrarre i seguenti path:

- order.id
- order.customer.name
- order.customer.address.city
- order.customer.address.zip
- order.items.item.product

- `order.items.item.quantity`

Questi path rappresentano i percorsi necessari per accedere ai valori dei nodi XML e sono fondamentali per garantire che il codice DataWeave generato sia in grado di interagire correttamente con il payload.

Una volta estratti i path dal payload XML, è possibile utilizzarli per identificare le variabili di accesso nel codice XSLT originale. Queste variabili corrispondono alle espressioni XPath utilizzate per accedere ai nodi XML. Il processo prevede di scansionare il codice XSLT convertito in DataWeave e individuare tutte le occorrenze di queste espressioni e modificarle in modo che includano il prefisso `payload..`.

Ad esempio, se il codice DataWeave contiene l'espressione `/order/id`, questa viene modificata in `payload.order.id`.

Questo metodo presenta numerosi vantaggi rispetto all'utilizzo di modelli linguistici (LLM) per automatizzare lo stesso processo. Innanzitutto, poiché i path vengono estratti direttamente dal payload XML, siamo certi che nessun valore verrà trascurato. Questo garantisce una copertura completa e una maggiore affidabilità rispetto a un approccio basato su LLM, che potrebbe occasionalmente omettere alcune espressioni o generare risultati incoerenti. In caso di mancanza del valore nel payload, il problema non sarà legato al modello, ma alla mancanza di informazioni nel payload stesso.

Il beneficio in termini di correttezza e affidabilità supera ampiamente i costi iniziali. Inoltre, questo approccio riduce drasticamente il tempo che uno sviluppatore dovrebbe impiegare per eseguire manualmente queste operazioni, migliorando l'efficienza complessiva del processo di migrazione.

Oltre alla gestione dei path XML, abbiamo implementato un ulteriore miglioramento per semplificare ulteriormente il lavoro degli sviluppatori. Durante la conversione da XSLT a DataWeave, Spectrum genera spesso porzioni di codice che servono a rendere il codice compilabile ma che non sono necessarie per il funzionamento finale dell'API.

Con l'approccio basato sull'analisi del payload XML, abbiamo automatizzato anche questa fase del processo. Il sistema è in grado di identificare e rimuovere automaticamente queste parti di codice, garantendo che il codice DataWeave risultante sia *quasi* pronto per l'uso.

## 3.7 Chatbot Integrato

Un'altra applicazione pensata per migliorare il processo di migrazione da XSLT a DataWeave è l'integrazione di un chatbot all'interno dell'applicativo Spectrum. Questa scelta espande le possibilità di interazione con il sistema, offrendo molteplici vantaggi sotto diversi aspetti.



Il chatbot integrato rappresenta uno strumento innovativo progettato per supportare gli sviluppatori interni dell'azienda durante il processo di migrazione da XSLT a DataWeave. L'obiettivo principale di questo sistema è quello di semplificare le attività ripetitive e fornire assistenza tecnica mirata, riducendo il carico di lavoro sugli sviluppatori e migliorando l'efficienza complessiva del processo. Il chatbot è stato implementato direttamente all'interno dell'applicativo Spectrum, garantendo un'integrazione fluida e un'esperienza utente senza interruzioni.

### 3.7.1 Scopo del Chatbot

Il chatbot è stato concepito principalmente per rispondere alle esigenze quotidiane degli sviluppatori che lavorano con la conversione da XSLT a DataWeave. Tra le sue funzionalità principali, si evidenzia il supporto nella risoluzione di problemi comuni, come l'aggiunta del prefisso `payload` alle espressioni XPath, qualora l'utente volesse comunque provare a farlo nonostante le criticità espresse in precedenza, o la generazione di payload XML o JSON di test compatibili con il codice DataWeave generato. Inoltre, il sistema offre la possibilità di utilizzare pulsanti preconfigurati che inviano richieste specifiche al modello linguistico, consentendo agli utenti di ottenere risposte rapide e mirate senza dover formulare domande complesse. Questi prompt statici sono stati progettati per coprire le attività più frequenti e ripetitive, come la spiegazione del codice, l'identificazione di potenziali bug o la generazione di codice citato in precedenza.

Un'altra caratteristica importante del chatbot è la sua capacità di gestire domande generiche sul codice DataWeave. Gli sviluppatori possono porre domande aperte relative alla logica del codice o richiedere suggerimenti per migliorarne la qualità, ma principalmente avendo la possibilità di fare domande l'espansione del modulo è dipendente dall'input dell'utente, quindi richiede meno manutenzione rispetto ad un modulo con funzionalità fisse.

### 3.7.2 Piattaforma e Tecnologia

Dal punto di vista tecnologico, il chatbot è stato integrato direttamente nell'applicativo Spectrum, separando il codice generato da un modulo interamente dedicato alla chat. Per quanto riguarda il modello linguistico (LLM), è stato inizialmente scelto `GPT-4o-mini`, poiché offre un buon equilibrio tra prestazioni e costi. Tuttavia, l'architettura del sistema è stata progettata per essere facilmente espandibile, permettendo l'integrazione di altri modelli, co-

me `o1-mini`<sup>2</sup>, che ha un costo superiore a `GPT-4o-mini`, almeno di 8 volte, ma offrendo un miglioramento notevole nelle capacità di linguaggio, secondo i *benchmark* di OpenAI.

Le chiamate alle API del modello sono state implementate utilizzando la libreria `urllib3`, questo sempre legato al problema citato in precedenza nel capitolo 3.5.2, anche se ha comportato la mancanza di alcune funzionalità avanzate, come lo streaming delle risposte. Nonostante ciò, il chatbot è in grado di fornire risposte rapide e precise, tenendo a mente che comunque il costo delle chiamate è dipendente da quanto il modello viene utilizzato.

Tra le funzionalità principali del chatbot, si evidenziano la conversione del codice in una struttura ben definita, richiesta dagli sviluppatori, la generazione di payload di test compatibili con il codice DataWeave e l'analisi critica del codice per individuare potenziali problemi o errori.

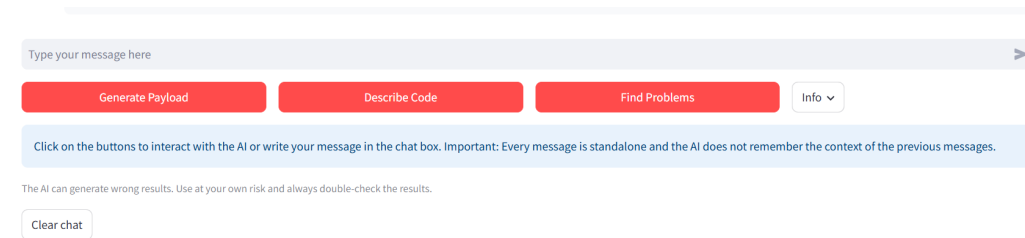


Figura 3.8: Interfaccia del Chatbot

### 3.7.3 Limiti e Sfide

Nonostante le sue numerose funzionalità, il chatbot presenta alcuni limiti che devono essere attentamente considerati. Innanzitutto, la comprensione di domande complesse o ambigue può rappresentare una sfida per modelli più piccoli. In questi casi, il sistema potrebbe fornire risposte incomplete o errate, richiedendo agli utenti di riformulare le domande o di verificare attentamente i risultati ottenuti.

Prima di parlare di un altro limite, è importante definire il concetto di contesto.

**Definizione 3.5** (Contesto). Il contesto è l'insieme delle informazioni rilevanti per comprendere e rispondere a una determinata richiesta. Nel contesto di un chatbot, si riferisce alle informazioni fornite dall'utente durante la conversazione, che vengono utilizzate per generare risposte mirate e coerenti. Il contesto può includere domande precedenti, risposte date e informazioni specifiche sull'utente o sull'ambiente circostante.

<sup>2</sup><https://openai.com/index/openai-o1-mini-advancing-cost-efficient-reasoning/>

Un altro limite significativo riguarda i costi associati all'utilizzo dei token. Per motivi di efficienza economica, il **contesto** della conversazione non viene mantenuto tra una richiesta e l'altra. Ciò significa che ogni volta è necessario inviare l'input completo della conversazione, aumentando il numero di token utilizzati e limitando la capacità del chatbot di gestire dialoghi prolungati. Questa scelta, sebbene pragmatica, può risultare frustrante per gli utenti che desiderano interagire con il sistema in modo più dinamico.

Infine, è importante considerare le implicazioni di privacy e sicurezza legate all'invio di codice e dati a modelli esterni. Sebbene i file utilizzati durante i test non contengano dati sensibili, è fondamentale valutare attentamente i rischi associati alla condivisione di informazioni aziendali con servizi cloud-based. In futuro, potrebbe essere necessario implementare meccanismi di anonimizzazione o utilizzare modelli locali per garantire la protezione dei dati.

### 3.7.4 Esempi di Utilizzo

Per illustrare il funzionamento del chatbot, consideriamo alcuni esempi pratici di interazione. Supponiamo che uno sviluppatore debba aggiungere il prefisso **payload.** a tutte le espressioni XPath presenti nel codice DataWeave. Invece di eseguire questa operazione manualmente, l'utente può inviare una richiesta al chatbot, che analizza il codice e restituisce una versione modificata pronta per l'uso. Questo processo riduce significativamente il tempo necessario per completare l'attività e minimizza il rischio di errori umani.

Un altro esempio riguarda la generazione di payload XML di test. Se un utente fornisce un frammento di codice DataWeave che accede a nodi XML specifici, il chatbot genera automaticamente un payload compatibile, consentendo all'utente di testare il codice senza doverlo creare manualmente. Questa funzionalità è particolarmente utile per accelerare il processo di verifica e garantire che il codice funzioni correttamente.

Infine, consideriamo il caso di uno sviluppatore che desidera comprendere meglio la logica di un frammento di codice DataWeave complesso. Invocando il prompt **describe\_code**, il chatbot fornisce una spiegazione dettagliata in italiano, coprendo tutti gli aspetti del codice e aiutando l'utente a familiarizzare rapidamente con la sua struttura e funzionamento.

È importante sottolineare che, mentre il chatbot è uno strumento potente, l'accuratezza delle sue risposte dipende dalla qualità dell'input fornito e dal modello utilizzato. Pertanto, gli utenti devono sempre verificare attentamente i risultati prima di applicarli al codice finale.

In termini quantitativi, il tempo medio di risposta del chatbot è di circa 3-5 secondi per query semplici, mentre le richieste più complesse possono richiedere fino a 10 secondi.

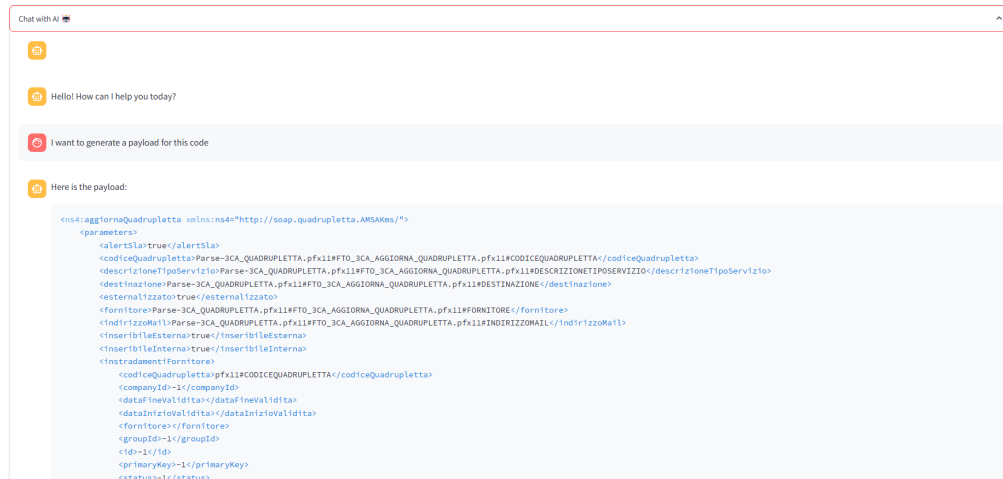


Figura 3.9: Esempio di generazione di payload XML

### 3.7.5 Esempio di task complesso

Un task complesso è quello che richiede un tipo specifico di operazione intermedia nota come *mapping*. Questo processo si occupa di leggere dati spesso provenienti da un database e da un payload per trasformarli in un unico oggetto JSON, che verrà successivamente utilizzato all'interno del flusso applicativo. Si tratta di un passaggio strettamente legato alla fase di trasformazione di un'operazione ETL, sebbene qui ci si concentri esclusivamente sulla generazione di un JSON coerente e strutturato. La sfida principale di questa attività risiede nella mancanza di una struttura standardizzata per il mapping: ogni sviluppatore tende ad adottare un approccio personalizzato, seguendo metodi che gli risultano più familiari o pratici. Per affrontare questa variabilità, è stato necessario collaborare con gli sviluppatori per identificare un metodo comune e strutturare un prompt altamente dettagliato in grado di guidare il processo di trasformazione. Il prompt, denominato **advanced\_refactoring**, fornisce istruzioni precise per ristrutturare script DataWeave, integrando logiche condizionali direttamente nella struttura del mapping e garantendo la coerenza del formato JSON finale.

Il prompt **advanced\_refactoring** è formulato come segue:

```
1 ## Instructions: ##
```

```

2 You are an assistant tasked with refactoring the given DataWeave
  script by inlining function logic directly within its
  mapping structure. For each key in the jsonObject:
3 - If a corresponding variable exists in the payload (even if
  nested), assign its value using payload.variableName. Do not
  insert the entire jsonObject path when referencing the
  payload variable.
4 - Replace separate functions with inline conditional expressions
  following this format:
5   key: if () else
6 Do not include the word "then" after the "if" condition. Do not
  use && as a logical operator, use and instead.
7 - Ensure that a default value is provided when necessary, while
  preserving the original transformation logic.
8 - The final output should be a single JSON object (after a '—'
  delimiter) mapping each key to its transformed value.
9 - If a key contains a hyphen (e.g., foo-bar), enclose the key in
  quotes (e.g., "foo-bar").
10 - Add inside jsonObject some missing keys and add the prefix "
  jsonObject." to the existing keys that are missing it.
11 - Do not remove any import statements
12 - Do not say // More mappings can go here as required... or
  similar comments, always provide the full transformation.
13 Complete the full transformation in one step without asking for
  further input.
14 ## Example: ##
15 Payload: { "name": " John Doe ", "birthDate": "1990-05-15
  00:00:00", "fiscalCode": "" }
16 DataWeave code:
17 %dw 2.0 output application/json
18 import substring, isEmpty from dw::core::Strings
19 var jsonInput = {
20   "Soap": {
21     "Request": {
22       "Name": payload.name,
23       "BirthDate": payload.birthDate,
24       "FiscalCode": payload.fiscalCode
25     }
26   }
27 }
28 —
29 (
30   jsonObject: {
31     "Name": jsonInput.S Soap.Request.Name,
32     "BirthDate": jsonInput.S Soap.Request.BirthDate,
33     "FiscalCode": if (isEmpty(jsonInput.S Soap.Request.FiscalCode)
34   ) "N/A" else jsonInput.S Soap.Request.FiscalCode
35   }
36 )

```

```

36 ## Output: ##
37 %dw 2.0 output application/json
38 var jsonObject = {
39     "Name": payload.name,
40     "BirthDate": payload.birthDate ,
41     "FiscalCode": if (isEmpty(payload.fiscalCode)) "N/A" else
        payload.fiscalCode
42 }
43
44 {
45     "Name": jsonObject.Name,
46     "BirthDate": jsonObject.BirthDate ,
47     "FiscalCode": jsonObject.FiscalCode
48 }
49 Refactor the provided code accordingly.

```

Questo prompt è stato progettato per essere estremamente dettagliato, fornendo indicazioni chiare su come gestire ogni aspetto del mapping, dalle condizioni logiche alle convenzioni sintattiche specifiche di DataWeave. Nonostante la sua complessità, questo approccio non è privo di limiti. Il modello utilizzato `gpt-4o-mini` spesso fallisce nell'eseguire il task senza introdurre errori sintattici o logici, oppure spesso interrompe prematuramente la generazione del codice, lasciando allo sviluppatore il compito di completarlo manualmente. Inoltre, la scarsa familiarità della maggior parte dei modelli con il linguaggio DataWeave può portare all'inserimento di keyword errate o alla violazione delle convenzioni sintattiche. Nonostante queste criticità, l'utilizzo di un tale strumento rappresenta un notevole passo avanti in termini di efficienza. Senza questo supporto, gli sviluppatori potrebbero impiegare ore per elaborare manualmente file complessi, mentre il processo automatizzato, anche se imperfetto, consente di ridurre drasticamente i tempi di lavoro. Tra le possibili soluzioni future, si possono considerare un fine-tuning mirato su task di questo tipo, l'adozione di modelli specializzati o la creazione di prompt ancora più articolati, eventualmente basati su esempi di file convertiti manualmente da sviluppatori esperti. Queste strategie, già discusse in precedenza, potrebbero ulteriormente migliorare le prestazioni e l'affidabilità del sistema.

Questo esempio, sebbene complesso, dimostra il potenziale del chatbot integrato per supportare gli sviluppatori in attività che difficilmente sarebbero automatizzabili da un algoritmo tradizionale.

### 3.7.6 Considerazioni

Pensando ad implementazioni future, ci sono diversi piani per migliorare ed espandere il chatbot. Innanzitutto, si sta valutando l'integrazione di modelli più grandi e sofisticati, come o1-mini, per migliorare la qualità delle risposte e la capacità di gestire domande complesse tramite meccanismo di *reasoning*. Inoltre, si prevede di implementare funzionalità avanzate, come lo streaming delle risposte, che consentiranno agli utenti di visualizzare i risultati in tempo reale, migliorando ulteriormente l'esperienza utente. Questo aspetto è strettamente legato al modo in cui l'applicativo viene distribuito, come accennato in precedenza.

Un'altra area di sviluppo riguarda l'implementazione di meccanismi di *context management*, che consentiranno al chatbot di mantenere lo stato della conversazione tra le richieste, migliorando la coerenza e la continuità della comunicazione. Questa funzionalità è invece legata ai costi e all'utilizzo che si vorrà fare del modello, poiché mantenere il contesto tra le richieste comporta un aumento significativo del numero di token utilizzati e l'utilità di tale funzionalità deve essere attentamente valutata.

# Capitolo 4

## Risultati e Conclusioni

Il lavoro svolto durante questa tesi rappresenta un'importante automazione del processo di migrazione da **XSLT** a **DataWeave** nell'ambito aziendale, un compito che fino ad oggi è stato spesso eseguito manualmente dagli sviluppatori con notevole dispendio di tempo. Il progetto **Spectrum**, su cui si è basata gran parte della ricerca e dello sviluppo, è stato significativamente migliorato sia dal punto di vista funzionale che tecnologico. Attraverso l'integrazione di tecniche di Intelligenza Artificiale, in particolare i Large Language Models (LLM), sono stati introdotti nuovi moduli per automatizzare alcune delle attività più ripetitive e soggette a errori umani.

Di seguito, si riassume il lavoro svolto ed i risultati ottenuti, in riferimento agli obiettivi della tesi.

### 4.1 Riepilogo del lavoro

Il primo obiettivo principale del progetto era quello di estendere e migliorare il software, rendendolo più user-friendly e accessibile. Questo è stato raggiunto attraverso una serie di modifiche all'interfaccia grafica, come l'introduzione di una visualizzazione side-by-side del codice **XSLT** e **DataWeave**, nonché l'aggiunta di funzionalità come la possibilità di selezionare il formato di output desiderato (XML o JSON) e l'inserimento di una propria API Key di OpenAI per accedere ai modelli LLM.

Parallelamente, sono state esplorate diverse strategie per integrare l'intelligenza artificiale nel processo di conversione. Inizialmente, si è valutato l'utilizzo diretto di modelli LLM per convertire il codice **XSLT** in **DataWeave**. Tuttavia, questa idea è stata scartata a causa delle limitazioni intrinseche dei modelli LLM, come la loro incapacità di gestire correttamente strutture complesse e non comuni, la tendenza a generare codice *allucinato* (sintassi erra-



ta, variabili non definite, ecc.) e la scarsa affidabilità complessiva. Anche se i risultati ottenuti con alcuni modelli avanzati come GPT-4o e Claude-3-Opus erano promettenti, il codice generato richiedeva comunque una revisione manuale sostanziale, annullando in parte i vantaggi dell'automazione.

Successivamente, si è passati a un approccio più mirato, utilizzando i modelli LLM per task specifici all'interno del processo di conversione, come la generazione di nomi descrittivi per le variabili o l'identificazione delle espressioni XPath che richiedono il prefisso payload.. Questi micro-task sono risultati più adatti all'utilizzo di LLM, poiché richiedono meno comprensione contestuale e sono meno soggetti a errori critici. Ad esempio, il modulo dedicato alla generazione di nomi descrittivi ha dimostrato di poter migliorare significativamente la leggibilità del codice, rendendolo più facile da mantenere e comprendere. Tuttavia, anche in questo caso, la qualità del risultato dipende fortemente dalla formulazione del prompt e dalla capacità del modello di interpretare correttamente le istruzioni fornite.

Un altro aspetto interessante del progetto è stata l'implementazione di un *chatbot* integrato all'interno dell'applicativo. Questo strumento innovativo è stato progettato per supportare gli sviluppatori nella risoluzione di problemi comuni, come la generazione di payload di test compatibili con il codice DataWeave o l'analisi critica del codice per individuare potenziali bug. Nonostante le sue numerose funzionalità, il chatbot presenta alcuni limiti, soprattutto in termini di costi associati all'utilizzo dei token e di comprensione di domande complesse o ambigue. Inoltre, la mancanza di un sistema di *context management* efficace rende difficile gestire conversazioni prolungate, costringendo gli utenti a reinserire ogni volta l'input completo.

In sostanza, gli obiettivi della tesi sono stati raggiunti ed in certi casi anche superati, specialmente in considerazione di tutti i fattori limitanti con i quali ci si deve scontrare in un contesto aziendale.

## 4.2 Fattori limitanti

Durante lo sviluppo del progetto, sono emersi diversi fattori che hanno limitato l'efficacia delle soluzioni proposte. Uno dei principali ostacoli è stato il costo elevato associato all'utilizzo di modelli LLM avanzati. Questi modelli offrono prestazioni *nettamente* superiori rispetto ai modelli open-source, ma richiedono investimenti significativi in termini di budget, soprattutto quando si lavora su file di grandi dimensioni o su progetti con numerosi file da convertire. Inoltre, l'utilizzo di servizi cloud-based per l'elaborazione del codice solleva importanti questioni di *privacy* e *sicurezza*. Sebbene i file utilizzati durante i test non contenessero dati sensibili, è fondamentale valutare at-

tentamente i rischi associati alla condivisione di informazioni aziendali con servizi esterni. In futuro, potrebbe essere necessario implementare meccanismi di anonimizzazione o utilizzare modelli locali per garantire la protezione dei dati.

Infatti, il codice che viene analizzato e trasformato durante il processo di migrazione è spesso di proprietà di terze parti, aziende clienti che hanno affidato il proprio software a società di consulenza per la sua modernizzazione. Questo codice può contenere informazioni sensibili, come logiche proprietarie, strutture dati specifiche o dettagli implementativi che rappresentano un vantaggio competitivo per l'azienda. In questo contesto, l'invio diretto di tale codice a modelli esterni, come quelli offerti da OpenAI o altri provider cloud, solleva seri dubbi sulla sicurezza e la protezione delle informazioni. Le aziende sono sempre più preoccupate del fatto che i loro dati possano essere utilizzati per addestrare ulteriormente i modelli linguistici, violando così la proprietà intellettuale e le normative sulla privacy.

Per mitigare questi rischi, è stato proposto l'uso di **pipeline di anonimizzazione** o **generalizzazione dei dati**, in cui il codice viene modificato per rimuovere informazioni sensibili o proprietarie prima di essere inviato ai modelli linguistici. Ad esempio, nomi di variabili specifici potrebbero essere sostituiti con identificatori generici, mentre strutture dati complesse potrebbero essere semplificate per preservarne la forma ma non il contenuto originale. Tuttavia, anche questa soluzione presenta limiti significativi. Innanzitutto, l'anonimizzazione non è sempre sufficiente a garantire la completa protezione dei dati, poiché alcuni pattern o logiche implementative potrebbero ancora essere riconoscibili da esperti. In secondo luogo, il processo di anonimizzazione richiede tempo e risorse, aumentando i costi operativi e riducendo l'efficienza complessiva del sistema. Una cosa del genere, però, potrebbe compromettere effettivamente la qualità del codice generato, poiché le informazioni rimosse potrebbero essere cruciali per la corretta trasformazione del codice.

Un'altra possibile soluzione consiste nell'utilizzo di **modelli linguistici locali**, che possono essere eseguiti all'interno dell'infrastruttura dell'azienda di consulenza senza dover inviare dati esternamente. Questo approccio offre un maggiore controllo sulla governance dei dati e riduce il rischio di violazioni della privacy. Tuttavia, l'implementazione di modelli locali richiede investimenti significativi in termini di hardware e competenze tecniche, rendendolo poco pratico per molte organizzazioni. Inoltre, i modelli locali spesso non raggiungono le stesse prestazioni dei loro omologhi cloud-based, limitando la qualità delle risposte e l'efficacia del sistema.

Le preoccupazioni delle aziende non si limitano solo alla privacy, ma si estendono anche alle **implicazioni etiche e legali** dell'uso di modelli lingui-

stici. Molte organizzazioni temono che i dati inviati ai modelli esterni possano essere utilizzati per addestrare ulteriormente le IA, contribuendo involontariamente allo sviluppo di tecnologie concorrenti o dannose. Di conseguenza, molte aziende preferiscono evitare del tutto l'uso di modelli esterni, optando per soluzioni manuali o semi-automatiche che, sebbene meno efficienti, garantiscono un maggiore controllo sui dati.

In buona sostanza, il problema principale non riguarda solo l'efficacia tecnica degli strumenti basati su LLM, ma anche la fiducia che le aziende ripongono nel processo di migrazione. Fino a quando non saranno disponibili soluzioni che garantiscano un livello adeguato di sicurezza, anonimizzazione e conformità normativa, molte organizzazioni continueranno a mostrare resistenza all'adozione di queste tecnologie. È quindi fondamentale investire nello sviluppo di framework e metodologie che affrontino queste criticità, garantendo che i dati aziendali vengano trattati in modo sicuro e responsabile.

Infine, un altro fattore limitante riguarda la complessità del codice **XSLT** e **DataWeave**. Le espressioni XPath, ad esempio, rappresentano un elemento chiave nella logica del codice generato da Spectrum, ma la loro individuazione e modifica automatica si è rivelata particolarmente impegnativa. I modelli LLM spesso commettono errori nella selezione delle stringhe o nella classificazione delle espressioni, richiedendo comunque una revisione manuale del codice. Questo dimostra che, nonostante l'automazione, il modello non è ancora completamente affidabile e deve essere affiancato da un controllo umano. Questo è comunque un problema comune vista la scarsità di dataset contenenti esempi di codice **XSLT** e **DataWeave**, proprio per il fatto che il linguaggio **DataWeave** è relativamente nuovo e proprietario di MuleSoft.

## 4.3 Sviluppi futuri

Guardando al futuro, ci sono diverse direzioni in cui il progetto potrebbe evolvere per superare i limiti attuali e sfruttare pienamente il potenziale dell'intelligenza artificiale. Una delle strade più promettenti è quella del fine-tuning di un modello pre-addestrato su un dataset specifico per la conversione da **XSLT** a **DataWeave**. Questo approccio richiede la creazione di un dataset completo e ben strutturato, contenente esempi di codice scritti manualmente dagli sviluppatori. Un dataset di alta qualità consentirebbe di addestrare un modello già performante sul codice, come **GPT-4o**, a diventare ancora più specializzato e preciso nel contesto specifico della migrazione da **XSLT** a **DataWeave**. Tuttavia, questa soluzione comporta costi elevati, sia in termini di risorse computazionali che di tempo necessario per raccogliere e preparare i dati.

Per quanto riguarda i problemi di privacy, sarebbe opportuno esplorare l'utilizzo di modelli locali o di soluzioni ibride che combinino l'uso di servizi cloud con processi di anonimizzazione dei dati. Ad esempio, si potrebbero rimuovere o mascherare parti sensibili del codice prima di inviarlo al modello, garantendo così la protezione delle informazioni aziendali. In alternativa, si potrebbe considerare l'adozione di modelli open-source, come Llama o Mixtral, che possono essere addestrati e utilizzati localmente, riducendo i rischi associati alla condivisione dei dati. Tuttavia, come visto in precedenza, se si vuole ottenere risultati di alta qualità, è necessario investire in modelli più avanzati e sofisticati, che spesso sono disponibili solo come servizi cloud vista la loro complessità computazionale.

Infine, sarebbe interessante esplorare l'utilizzo di tecniche avanzate di Prompt Engineering per ottimizzare l'interazione con i modelli LLM. Ad esempio, si potrebbero sviluppare prompt dinamici che si adattano automaticamente al contesto e alle esigenze specifiche dell'utente, migliorando la precisione e l'affidabilità delle risposte, magari utilizzando un LLM proprio per ridefinire i prompt prima ancora che vengano passati al modello di generazione

## **4.4 Un'alternativa potenzialmente promettente: il RAG**

Il Retrieval-Augmented Generation (RAG) è un approccio che combina le capacità generative dei Large Language Models (LLM) con un sistema di recupero di informazioni contestuali rilevanti da una base di dati esterna. Questo metodo permette al modello di integrare conoscenze specifiche del dominio o del contesto, migliorando la coerenza e l'accuratezza delle risposte generate.

Nel caso della generazione di codice DataWeave, il RAG può essere applicato per mitigare alcune delle sfide legate alla conversione da XSLT. Ad esempio, durante il processo di migrazione, il modello potrebbe non disporre di sufficienti informazioni sulle convenzioni di DataWeave o sui pattern di trasformazione specifici del progetto. Integrando un sistema di retrieval, il RAG può recuperare frammenti di codice DataWeave esistenti o esempi di trasformazioni simili dal repository del progetto, fornendo al modello un contesto più accurato per generare codice coerente. Questo approccio ridurrebbe il rischio di allucinazioni, come la generazione di sintassi errata o l'uso improprio di funzioni, garantendo che il codice prodotto sia più affidabile e richieda meno interventi manuali.

Questa tecnica, d'altra parte, richiede un'infrastruttura complessa per gestire il recupero delle informazioni, come un database centralizzato o un sistema di indicizzazione dei file. Inoltre, il RAG può introdurre nuove sfide in termini di integrazione con i modelli LLM, come la necessità di sincronizzare i risultati generati con i dati recuperati e di mantenere la coerenza tra le due fonti di informazione. Tuttavia, se implementato correttamente, il RAG può essere senza dubbio un'opzione interessante per aumentare ancora di più l'efficacia del processo di migrazione.

## 4.5 Conclusioni

In conclusione, il lavoro svolto durante questa tesi ha dimostrato che l'integrazione di intelligenza artificiale nel processo di migrazione da **XSLT** a **DataWeave** è una strada percorribile, ma richiede un approccio mirato e ben pianificato. L'utilizzo di modelli LLM per task specifici ha mostrato risultati promettenti, ma è chiaro che il loro potenziale può essere pienamente sfruttato solo con investimenti significativi in termini di dati, risorse e tecnologie. La creazione di un dataset completo e accurato, insieme al fine-tuning di un modello avanzato, rappresenta la strada più promettente per migliorare ulteriormente l'automazione del processo.

Tuttavia, è importante tenere presente che l'IA non è una soluzione universale e deve essere utilizzata in modo complementare alle competenze umane. Anche i modelli più sofisticati commettono errori e richiedono una revisione manuale, soprattutto in contesti complessi come quello della migrazione di codice. Pertanto, l'obiettivo finale non dovrebbe essere quello di sostituire completamente il lavoro degli sviluppatori, ma piuttosto di supportarli, riducendo il carico di lavoro e migliorando l'efficienza complessiva del processo.

C'è anche da sottolineare che con IA non si deve fare riferimento solamente a strumenti come gli LLM, ma anche a tecniche di *Machine Learning* più tradizionali come la *Classificazione*, *Clustering* e *Regressione*. Queste tecniche purtroppo non sono state utilizzate in questo progetto per mancanza di casi d'uso applicabili, ma questo non significa che in futuro non possano esserci casi in cui la presenza di un classificatore o di un algoritmo di clustering non possa essere utile per automatizzare qualche task.

Il lavoro di tesi è stato incentrato principalmente su queste tecnologie, nell'ambito di un tirocinio che ha richiesto un approfondito studio di diverse tecnologie e software. Oltre allo sviluppo delle strategie presentate, il tirocinio ha richiesto anche un estensivo lavoro orientato al miglioramento del software stesso e alla risoluzione dei suoi problemi. Infatti, il corretto funzio-

namento algoritmico del software di conversione è un requisito cruciale per il funzionamento dell'intero sistema. Questo è stato garantito tramite un costante dialogo e scambio di feedback con gli altri sviluppatori e opportuni test volti a verificare il corretto funzionamento dell'intero applicativo, seguiti da lunghe sessioni di refactoring e bug-fixing.

In sintesi, il progetto Spectrum rappresenta un importante passo avanti verso l'automazione intelligente della migrazione da **XSLT** a **DataWeave**. Con ulteriori investimenti e sviluppi futuri, siamo convinti che questa soluzione possa diventare uno strumento indispensabile per le aziende che devono affrontare la sfida della modernizzazione dei sistemi legacy.

# Bibliografia

- [1] Jeremy Howard and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification*. ACL, 2018.
- [2] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.
- [3] Shruthan Radhakrishna, Yazdan Jamshidi, and Hadi Minooei. Generative ai for dataweave: Generate code from unit tests. *Salesforce 360 Blog*, September 2023.
- [4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.
- [5] Wikipedia. Hallucination (artificial intelligence). [https://en.wikipedia.org/wiki/Hallucination\\_\(artificial\\_intelligence\)](https://en.wikipedia.org/wiki/Hallucination_(artificial_intelligence)).

# Ringraziamenti

E ora la conclusione, o meglio le battute finali.

Un ringraziamento speciale va ai miei relatori, Prof. Giorgio Terracina e il Dott. Sebastiano Antonio Piccolo. Vi ringrazio per avermi supportato, per la vostra disponibilità e pazienza nel guidarmi in questo progetto, per le vostre preziose indicazioni e per avermi incoraggiato a esplorare nuove idee. Grazie per avermi dato l'opportunità di lavorare su un argomento così interessante e stimolante. La vostra passione è stata una fonte di ispirazione per me.

Estendo i miei ringraziamenti all'azienda Lutech S.p.A. per avermi offerto l'opportunità di svolgere il tirocinio curriculare presso la loro struttura. Lavorare sul progetto Spectrum in un contesto aziendale reale è stata un'esperienza estremamente formativa e stimolante. Ringrazio in particolare il mio tutor Mattia e tutti i colleghi per l'accoglienza, la collaborazione e per aver condiviso con me le loro conoscenze, e per essere stati dei colleghi fantastici. In particolare, ringrazio Alessandro che mi ha affiancato e spalleggiato durante l'intero sviluppo del progetto; senza il tuo supporto e la tua disponibilità non sarei riuscito a portare a termine nulla di tutto questo.



A voi, Mamma e Papà. Grazie per l'amore incondizionato, per i sacrifici *silenziosi* che avete fatto per permettermi di studiare e inseguire i miei sogni, per avermi sempre incoraggiato a dare il massimo. A non aver mai fatto pressione su nulla, ad aver sempre creduto in me, ad esserci stati in ogni momento e avermi sempre supportato con il vostro affetto e comprensione. Grazie per non essere mai stati invadenti, per avermi sempre lasciato libero di scegliere e di seguire le mie passioni e per avermi sempre dato la libertà di esprimermi. Grazie per essere le persone che siete, non vi cambierei per nulla al mondo; se sono la persona che sono oggi è principalmente per merito vostro, per avermi fatto capire l'importanza delle cose, dell'impegno, delle cose giuste e di quelle sbagliate e di cosa significa volere bene. Vi amo.



A Francesco, per essere sempre stato la persona che più di tutti ha creduto in qualsiasi cosa io facessi, per avermi spinto, indirizzato e sostenuto in tutto questo percorso che oggi è la mia vita. Per aver avuto la forza di affrontare voci e parole contrarie e per aver combattuto affinché tutto questo potesse realizzarsi. Per esserci stato in qualsiasi momento, per non aver mai voltato le spalle e per avermi sempre supportato. Non è mai facile essere il fratello maggiore, e non è banale riuscire ad esserlo nel modo in cui lo sei stato tu. La nostra famiglia non eccelle nel dimostrare affetto con le parole, e io non sono da meno, ma ci tengo a dirti quanto ti voglio bene e quanto tu sia importante per me, anche se non te lo dico a parole. Grazie anche a Rossella, per esserci stata in tutte le occasioni e per non essere mai stata inopportuna, per avermi sempre supportato e per avermi sempre fatto sentire a casa. Non abbiamo mai coltivato un rapporto diretto, ma questo non mi ha fermato dal volerti un bene immenso e di considerarti molto più della compagna di mio fratello. Grazie anche di sopportare Francesco, che probabilmente è il compito più difficile di tutti.

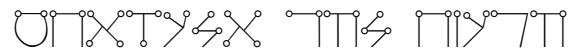
Ai miei Nonni, un ringraziamento carico d'affetto. Grazie per il vostro amore immenso, per le vostre parole, per la vostra accoglienza e per il vostro supporto. Per avermi sempre fatto sentire speciale e per avermi insegnato il valore, a volte difficile da comprendere, della famiglia. Per avermi sempre fatto sentire a casa e per avermi sempre accolto a braccia aperte. Grazie per essere stati una parte così importante della mia vita.

Un pensiero pieno d'affetto e gratitudine va anche a chi purtroppo non è più fisicamente qui con me oggi, ma la cui presenza sento viva nel cuore ogni giorno. So che sareste stati orgogliosi e il vostro ricordo continua ad esserci, per sempre.

Agli Zii, che non mi hanno mai fatto sentire a disagio, che mi hanno sempre riempito di affetto e amore, che mi hanno dato tutto quello che potevano (fisicamente e non, e anche di più). La vostra presenza, il vostro modo di essere unico e il vostro modo di affrontare la vita sono stati per me un esempio da seguire (infatti mi alzo alle 9 per un motivo). Grazie per essere sempre stati voi stessi e per essere così, per l'appunto, unici. Vi voglio un bene immenso

A Michele e Wilma, per avermi sempre fatto sentire a casa, per avermi dato più affetto e amore di quanto potessi mai immaginare, per avermi fatto sentire come parte della famiglia, per non avermi mai fatto sentire di troppo o fuori posto. A Michela e Maria Assunta, per avermi trattato come un fratello, per esserci state e per tutto quello che avete fatto per me. Ve ne sono grato, e sono grato di aver condiviso con voi gli ultimi anni. Vi vorrò sempre bene.

A tutti voi, grazie.



Non so davvero da dove iniziare a scrivere qui. Ci sarebbero così tante cose da dire che servirebbe una tesi appositamente per questa sezione.

Alle persone che sono state la mia quotidianità, che sono state presenti in ogni momento, che mi hanno cercato, che mi hanno capito, che mi hanno sostenuto, che mi hanno reso la persona che sono oggi, che mi hanno fatto crescere, che mi hanno fatto ridere, che mi hanno fatto piangere, che mi hanno fatto arrabbiare, che mi hanno fatto sentire *vivo*.

Grazie a chi c'è stato nei momenti belli e brutti, a chi mi ha fatto sentire speciale, a chi mi ha fatto sentire parte di un gruppo, a chi mi ha fatto sentire amato. Grazie a chi ha voluto costruire un legame profondo con me e che ha voluto conoscermi meglio, a chi ha voluto condividere con me le proprie esperienze e i propri pensieri.

A CiccioFazio per essere tu e solamente te stesso, Ciccio. A Michele per essere stato vicino ad uno rompipalle per mezzo anno ed esserti accollato un Daniele un pochino problematico, anche se avrei voluto strozzarti qualche volta, ti voglio bene. A Lorenzo, per essere davvero il LorenzoPiro della situazione sempre; mi fa sempre piacere e ridere sentire quello che hai da dire e da raccontare. Mi raccomando bro. Ad Aristide per esserti avvicinato a me e per avermi ascoltato quando ne avevo bisogno, spero che in futuro questo possa solo che legarci di più.

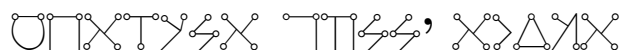
A Ludovica, Valentina e Simona, spero che il nostro rapporto possa solo migliorare, che ci sia modo di conoscerci meglio, di capirci meglio e di continuare ad avere questo legame.

A Laura per essersi dimostrata una persona fantastica, per avermi ascoltato e per essersi aperta con me nel momento più difficile. Sono contento di star costruendo un legame ancora più forte di quello che pensavo potesse esserci. Ho una grande stima per te e ammiro la tua forza di volontà, il tuo coraggio e la tua determinazione nel perseguire i tuoi obiettivi. Grazie per esserci stata e per essere te stessa.

A Shano che è davvero unico e insostituibile. Per essere stato sempre imparziale, amichevole e presente in ogni circostanza. Grazie per essere te stesso, per non avere bisogno di maschere e per essere sempre sincero. Ti voglio bene bro, mi raccomando non fare il sistemista a Milano..

A Giacomo, sei entrato nella mia vita da così poco tempo che mi sembra surreale il tipo di rapporto che abbiamo creato. Ci siamo aperti a vicenda, ci siamo esposti l'uno con l'altro e abbiamo costruito un legame più forte di quanto potessi mai immaginare. Anche se a volte vorrei davvero spaccarti qualsiasi cosa, sei una persona fantastica e sono contento di aver scommesso

in questa amicizia. Sono contento di aver trovato qualcuno come te, che mi ha fatto capire che non bisogna svalutarsi e che non bisogna mai sentirsi inferiori a nessuno, anche se non lo sai probabilmente. Ti voglio assai bene anche se non sono mai stato bravo a dirtelo, però spero che tu lo sappia.



Ad Alex, qualcuno con cui non mi ricordo nemmeno come mi sono conosciuto. Una persona che considero unica e insostituibile. Non ho mai conosciuto qualcuno come te, che mi ha insegnato cosa significa davvero apprezzare e amare qualcosa. Un perfezionista in ogni cosa, capace di dimostrare quello che vuole in un modo che non conosce nessun altro. Sono immensamente grato di esserti aperto, legato e di aver condiviso con me tutto questo tempo insieme. Grazie per avermi fatto scoprire un mondo nuovo e per avermi fatto affezionare a qualcosa di così importante per me. Non è mai stato facile e non lo sarà mai comunicare in modo diretto quanto ti voglio bene, ma non c'è bisogno di farlo perché non è quello il modo in cui lo dimostriamo. Spero davvero di continuare ad avere questo rapporto con te, nonostante i nostri caratteri così diversi (però ho finito Monogatari il giorno prima di scrivere questi ringraziamenti).

A Simone, letteralmente la prima persona con cui ho costruito un legame di amicizia da quando ho messo piede all'università. Una persona che nel corso del tempo è cambiata drasticamente, una persona con la quale ho costruito un rapporto difficilmente costruibile con qualcun altro. Abbiamo imparato a conoscerci in modo diretto ed indiretto, da colleghi a coinquilini ad amici (quasi dinuovo coinquilini, ma meglio che sia rimasto quasi). Hai cambiato spesso il mio modo di vedere le cose, mi hai dato molte prospettive, mi hai fatto crescere e mi hai fatto capire che non è mai troppo tardi per cambiare. Anche a te avrei voluto spaccarti tutto quanto, ma non avremmo questo legame se non ci fossimo aperti l'uno con l'altro. Grazie di essere sempre e comunque te stesso, nonostante tutto il percorso che hai affrontato e che affronti giornalmente<sup>1</sup>. Sei una persona fantastica e sono contento di averti conosciuto. Ti voglio bene, anche se non lo dico mai.

A Domenico, per essere sempre stato un esempio da seguire, per avermi insegnato cosa significa davvero essere un amico, per avermi fatto capire il valore dell'amicizia e per dimostrarmi in qualsiasi momento che non sono mai solo. Per avermi ispirato e fatto capire cosa significa non mollare mai, cosa significa avere la *determinazione* di raggiungere un obiettivo e di non

---

<sup>1</sup>[https://it.wikipedia.org/wiki/Disturbo\\_da\\_deficit\\_di\\_attezzione/iperattivit%C3%A0](https://it.wikipedia.org/wiki/Disturbo_da_deficit_di_attezzione/iperattivit%C3%A0)

fermarsi mai. Sei sempre statao presente, anche più di quanto ti fosse dovuto, e nonostante le distanze e le differenze di vita, sei sempre riuscito a farmi sentire a casa. Un legame forte come questo non è qualcosa che si costruisce con poco impegno e con poche parole, ma è qualcosa che si costruisce con il tempo, con la pazienza e con l'amore. Grazie di essere così e di metterci davvero tutto te stesso in quello che fai nel tuo privato, che mi è stato di esempio e di ispirazione, e di quello che fai per gli altri e per me. Ti voglio bene.

Antonio, Giovanni e Roberto: I fra

Non so da dove iniziare, davvero. Un rapporto con ognuno di voi costruito davvero con tanto, tanto amore e dedizione. Un flow di discussioni, sincerità, zero filtri e zero maschere. Un rapporto che non ha bisogno di parole, che non ha bisogno di essere spiegato, che non ha bisogno di essere raccontato. Un rapporto che è stato costruito con il tempo, con la pazienza e con l'amore. Un rapporto che è stato costruito con le esperienze, con le risate e con le lacrime. Un rapporto che è stato costruito con le parole e i silenzi. Un rapporto costruito nei soggiorni, nelle torrette, e perché no, nelle stufette. Ogni festività aveva il suo momento soggiorno, il momento macchina, il momento casa dell'acqua senza casa dell'acqua. Tutto quanto ha un sapore diverso quando siamo insieme, e ogni volta diventa sempre più unico, con una crescita totale da parte di tutti. Parlo di tutto questo come un'unica entità, un unico gruppo che viaggia sintonizzato sulla stessa frequenza. Sono infinitamente grato di avervi nella mia vita, di poter parlare di qualsiasi cosa con voi, di potermi aprire e di poter essere me stesso senza alcun tipo di filtro. Grazie di essere voi stessi, di avermi fatto crescere, di avermi fatto guardare le cose con occhi diversi e di avermi fatto sentire accettato. Vi voglio un bene immenso.

A Pasquale, per essere sempre stato te stesso. Per avermi supportato per tutti questi anni, per essere sempre stato presente nel momento del bisogno, per riuscire ad essere sempre l'amico che tutti vorrebbero avere. Nonostante la distanza, nonostante la bassissima frequenza di incontri e di conversazioni, sei sempre stato quel punto di riferimento che mi serviva per affrontare qualsiasi cosa. Sei sempre stato una persona fantastica, una persona che ha sempre saputo ascoltare e che ha sempre saputo dare il giusto consiglio. Grazie di essere sempre te stesso, di non aver mai cambiato il tuo modo di essere. Sono contento di averti nella mia vita e di essere stato in grado di costruire un legame così forte con te, così duraturo e così profondo. Ti voglio infinitamente bene, e sono pure venuto a Pescara quest'anno.

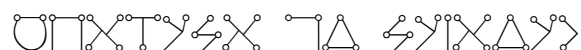


Mentre concludo questo capitolo, mi fermo un istante a guardare indietro. C'è una quiete diversa ora, un senso di percorso compiuto.

Non è più un tentativo di afferrare un nome che sfugge; è più come riconoscere una melodia che hai ascoltato tante volte. Non cerco più disperatamente i dettagli, ma riconosco l'impronta che ha lasciato su di me e su questo percorso. Quella compagnia, quella sintonia, è stata parte integrante di ciò che sono diventato e di ciò che questo traguardo rappresenta. Un lungo periodo di crescita colmo di emozioni e sentimenti.

Oggi vado avanti, com'è giusto che sia. Il tempo scorre e le strade inevitabilmente divergono. Ma la consapevolezza di ciò che è stato, la gratitudine per aver potuto condividere un pezzo di viaggio così importante rimane. Una gratitudine per quel tempo, per il valore inestimabile di ciò che ho ricevuto e imparato. Non è qualcosa che si può cancellare o dimenticare. È parte della trama, intessuta in questi anni e in questo risultato. È un ricordo che porto con me, non come un fardello, ma come una mappa che mi ha aiutato ad arrivare fino a dove sono arrivato oggi.

Grazie. Per esserci stata, per il tempo condiviso, per l'insostituibile supporto e per tutto ciò che è stato. A te.



A me, a Daniele, un messaggio per me stesso e un'auto convinzione dovuta. Sono stati anni di cambiamenti, che fossero modi di affrontare le cose, che fossero modi di vedere il mondo o che fossero approcci diversi in qualsiasi ambito. Sono cambiato come persona, mi sono sbloccato sotto diversi punti di vista e ho raggiunto obiettivi che non credevo raggiungibili. Nonostante questo, un senso di insoddisfazione continua a prevalere sulla mia vita. Questo è un piccolo traguardo che non mi sarei aspettato da raggiungere prima di iniziare questo percorso universitario, e oggi sono qui a concludere la mia carriera. Sono fiero di me e dei miei risultati, di quello che ho fatto e di ciò che ho costruito. Sono fiero di avere una famiglia in grado di capirmi e supportarmi in ogni cosa che faccio, con cognizione di causa. Sono fiero di avere costruito delle amicizie vere, forti e durature, senza maschere e senza menzogne. Sono contento di avere delle persone fantastiche nella mia vita che mi hanno reso la persona che sono oggi, che sono entrate silenziosamente a fare parte delle mie giornate e diventare parte della mia quotidianità, che mi hanno insegnato ad affrontare le sfide più tediose della mia vita. Persone che da sconosciute sono diventate così importanti, che con frasi semplici e

con concetti banali sono riuscite a smuovermi nell'animo e a farmi compiere azioni che non mi sarei mai aspettato di compiere. Mi ritengo fortunato di avervi nella mia vita. Con questo si chiude un capitolo importante della mia vita, che porta con sé diversi sottocapitoli e ne apre infiniti altri. Sono fiero di ciò che sono oggi e di quello che sono arrivato ad avere. A me, a voi, grazie.