ConnectIQ RAG Pipeline – Stato Attuale e Prossimi Passi

📌Contesto

Questo documento serve come **punto di ripartenza ufficiale** del lavoro sulla pipeline RAG di ConnectIQ. Può essere copiato, condiviso e usato come riferimento comune tra i membri del team.

L'obiettivo è evitare di perdere contesto, chiarire **cosa è stato fatto**, **perché è stato fatto**, e **cosa faremo nel prossimo step**.

🕐Obiettivo di Prodotto

Costruire un **Conference Chatbot vendibile**, capace di: - rispondere in modo **diretto e affidabile** a domande su un evento - usare esclusivamente informazioni presenti sul sito ufficiale - rigenerarsi con **un solo comando** a partire da un URL

Esempio di esperienza attesa:

> Inserisco l'URL di un evento → avvio la pipeline → il chatbot è pronto

---

Problema Identificato

Nonostante il sistema fosse **tecnicamente funzionante**, presentava problemi critici lato prodotto:

1. Pipeline spezzata in più comandi manuali
2. Stato incoerente tra documenti, embeddings e chat
3. Risposte a volte corrette, a volte evasive
4. Necessità per l'utente di conoscere dettagli tecnici (embeddings, CLI, step intermedi)

👉Questo rendeva il sistema **non vendibile**, anche se l'AI funzionava.

🔗Stato Attuale (Confermato)

La pipeline attuale funziona correttamente a livello tecnico:

- 👍Crawler attivo e stabile
- 👍Arricchimento semantico sulle pagine chiave (partners, about, team)
- 👍Chunking corretto (pagine corte = 1 chunk)

- 👉🏽Generazione embeddings completata con successo
- 👉🏽Chatbot in grado di rispondere correttamente a domande chiave (es. Project Manager)

Esempio verificato:

> **Who is the Project Manager of Nordic Fintech Week?**
> → *Wassa Camara is the Project Manager of Nordic Fintech Week.*

🖍️Problema Rimasto (Critico)

La pipeline **non è atomica**.

Attualmente richiede: 1. `run_pipeline.py` 2. `embeddings.py`

E se uno step viene dimenticato: - la chat usa dati vecchi - le risposte diventano incoerenti - l'utente non sa perché

👉Questo è **inaccettabile per un prodotto**.

🧰Decisione Architetturale Presa

Decisione chiave

> **Un solo entrypoint deve orchestrare l'intera pipeline.**

Comando finale desiderato

```
python run_pipeline.py https://nfweek.com --max-pages 60
```

Questo comando deve internamente: 1. Eseguire il crawler 2. Eseguire loader + chunking 3. Generare embeddings 4. Portare il sistema in stato READY

🔧Modifica Principale da Implementare

✂️Refactor di `embeddings.py`

- Da script CLI standalone
- A **modulo riutilizzabile** con classe tipo:

```
class EmbeddingPipeline:
    def run(self):
        ...
```

Il CLI resta solo come wrapper, non come unica entry.

✒️Refactor di `run_pipeline.py`

- Diventa **orchestratore unico**
- Importa e chiama direttamente `EmbeddingPipeline.run()`
- Nessun comando manuale extra

Output finale:

```
PIPELINE READY
```

Perché Questa Scelta

- Riduce errori umani
- Migliora UX e affidabilità
- Elimina stati intermedi incoerenti
- Rende il sistema **demo-ready e vendibile**

Questa non è una scelta “da sviluppatori”, ma **da prodotto**.

🗜️Prossimo Step Operativo

Domani si riparte da qui:

1. Refactor `embeddings.py` (classe + CLI wrapper)
2. Refactor `run_pipeline.py` (orchestratore completo)
3. Test end-to-end con **un solo comando**

Solo dopo: - miglioramenti retrieval - multi-event - UI avanzata

---

🔗Stato Finale Atteso

- 👉🏽1 comando
- 👉🏽1 pipeline
- 👉🏽1 stato coerente

- 👉chatbot affidabile
- 👉pronto per demo e vendita

Questo documento è il punto di ripartenza ufficiale.