Struttura tecnica: "Allungamento" del contesto per Llama (UX: consulente non-tecnico)

Questo documento descrive, passo-passo, l'architettura e il flusso necessari per far sembrare a un utente non tecnico (es. consulente) che **Llama abbia letto e compreso un documento lungo (50+ pagine)**. La pipeline è progettata per essere "invisibile": l'utente carica il file, fa la domanda e riceve risposte professionali, senza preoccuparsi di chunk, token o embeddings.

Principi chiave

- **Magia dietro le quinte**: tutto il preprocessing, chunking e retrieval è automatico e asincrono; l'utente interagisce solo con upload + ask.
- **RAG come esperienza, non come dettaglio**: usare retrieval per fornire a Llama solo i frammenti pertinenti, mantenendo il prompt sotto i limiti di contesto.
- **Multistage retrieval (coarse → fine)**: prima filtro veloce per ridurre l'insieme, poi re-ranking e composizione dei migliori pezzi.
- **Tono consulenziale**: il prompt template trasforma i frammenti tecnici in risposta formale/strategica.
- **Caching & idempotenza** per evitare ricalcoli inutili su documenti già indicizzati.

Componenti della pipeline

1. **Uploader / Ingest API (FastAPI / Express)**

2. Endpoint: `POST /upload` (file multipart)

3. Azioni: validazione formato, virus-scan, conversione a testo (pdfminer / docx2txt / pptx -> text), estrazione metadati (title, author, date), normalizzazione encoding.

4. Response: doc_id (UUID), stato (processing|ready|error)

5. **Preprocessor & Splitter**

6. Converti in testo continuo e normalizza (rimuovi header/footer ripetuti, numeri di pagina, OCR cleanup).

7. Chunking automatico: `chunk_size_tokens = 1024` con `overlap = 128` (valori configurabili per lingua italiana/indonesia).

8. Produce: elenco di chunk `{chunk_id, doc_id, text, start_pos, end_pos}`.

9. **Embedding Service**

10. Modello embedding: `sentence-transformers` o `OpenAI/text-embedding-3-small` / `instructor` a seconda del budget.

11. Calcolo embeddings batch e salvataggio su Vector DB (FAISS, Qdrant o Milvus).
12. Metadati: doc_id, chunk_id, page_range, language, embedding_ts.
13. **Vector DB + Metadata Store**
14. Vector DB per retrieval semantico (top_k search).
15. DB relazionale/NoSQL per metadati e stato dei documenti (Postgres/Mongo).
16. **Retriever (Coarse → Fine)**
17. Step A (coarse): search top 200 con embedding query veloce.
18. Step B (re-ranking): usare un modello di re-rank (cross-encoder leggero) per riordinare top 200 → top N (es. N=10).
19. Step C (composizione): unificare i top N in un contesto continuo con smart trimming (keep most recent/most relevant sentences).
20. **Prompt Builder (Policy + Role)**
21. Template che incapsula: role, istruzioni consulenziali, frammenti recuperati, istruzioni di sintesi/argomentazione, limitazioni (citare pagine/estratti), tono (formale, sintetico, raccomandazione + next steps).
22. Esempio: "Sei un consulente fiscale senior. Sulla base dei seguenti estratti, rispondi a..."
23. **Model Inference (Llama 3.1 8B o altro)**
24. Caricare Llama con adapter LoRA/PEFT se necessario.
25. Eseguire inferenza usando il prompt costruito; se necessario usare streaming per risposte parziali.
26. **Post-Processing & Citation**
27. Normalizza risposta: rimuovi ripetizioni, aggiungi riferimenti a chunk (p.es. "v. pag. 12-13"), genera suggerimenti azionabili e controlli di sicurezza (filtro personal data).
28. **Interfaccia Utente / API Ask**
29. Endpoint: `POST /ask` con `{doc_id, question, user_id, tone}`.
30. Risposta: `{answer, sources:[{chunk_id, score, page_range}], warnings}`.
31. **Monitoraggio / Logging / Cost Control**
    - Metriche: latenza retrieval, costo per query, numero di token usati per prompt, error rate.
    - Alert: se query supera token budget, loggare e inviare warning.

Flusso operativo (sequenza)

1. Utente carica file → `/upload` → risponde immediatamente con `doc_id` e stato `processing`.
2. Worker asincrono prende il file → converti, chunk, embedding, index → stato `ready`.
3. Utente chiede `/ask` col `doc_id` → retriever coarse→fine → build prompt → inferenza Llama → postprocess → return.
4. Cache risultati per query simili (fingerprint della domanda + doc_id) per 24h.

Parametri consigliati (valori iniziali)

- `chunk_size_tokens`: 1024
- `chunk_overlap`: 128
- `embedding_model`: "all-MiniLM-L6-v2" o equivalente economico
- `top_k_coarse`: 200
- `top_n_rerank`: 10
- `prompt_max_tokens`: 3800 (per Llama 4K / 8K adattare)
- `model_context_limit`: 4096 o 8192 -> adattare trimming

Esempio minimo: snippet Python (LangChain-like)

```
# pseudo-code high-level
from langchain import Document, FAISS, OpenAI, LLMChain

# 1) indexing (worker)
docs = pdf_to_docs('report.pdf')
chunks = split_docs(docs, chunk_size=1024, overlap=128)
embeddings = embed_model.encode([c.text for c in chunks])
faiss_index.add(embeddings, metadata=[c.meta for c in chunks])

# 2) ask flow
query_emb = embed_model.encode([user_question])
coarse_ids = faiss_index.search(query_emb, top_k=200)
candidates = fetch_metadata(coarse_ids)
reranked = cross_encoder.rerank(user_question, candidates)  # top 10
context = compose_context(reranked, max_tokens=3800)
prompt = build_prompt(role='consulente', context=context,
question=user_question)
answer = llama_model.generate(prompt)

return format_answer(answer, sources=reranked[:5])
```

UX / UI: cosa vede il collaboratore

- Upload file (drag & drop)
- Barra di stato: "Indicizzazione 0% → 100%" con ETA
- Campo domanda naturale e bottone "Chiedi a Llama"
- Risposta: testo principale + "Fonti & pagine" in fondo + suggerimenti pratici (next steps)

Integrazione con il dataset esistente

- Se hai già conversational dataset (`messages[]`), indicizzalo come documento separato o insieme: ogni conversazione diventa un documento/entry.
- Usa metadati custom (`client_id, case_id, tags`) per retrieval targettizzato.

Sicurezza e privacy

- Maschera PII automatico (regex + ML) prima di indicizzare.
- Access control: solo utenti autorizzati possono chiamare `/ask` su doc_id.
- Encryption at rest per vector DB e storage.

Operazioni e monitoring

- Worker autoscaling per batch di embedding.
- Rate limiting per `/ask` (es. 10 rps per utente) e budget per token per progetto.
- Telemetria: invia metriche a Prometheus/Grafana.

Esempi di prompt template (per ruolo consulente)

```
Sei un consulente senior esperto in fiscalità aziendale. Leggi gli estratti
seguenti (citare pagina) e rispondi alla domanda con un paragrafo sintetico,
poi un elenco di rischi e infine 3 azioni consigliate. Se le informazioni non
sono sufficienti, chiedi chiarimenti.

Estratti:
{context}

Domanda: {question}

Formato risposta:
1) Sintesi (2-4 frasi)
2) Rischi (bullet points)
3) Azioni consigliate (step concreti)
4) Riferimenti (pagina:chunk)
```

Test e validazione

- E2E test: upload di 1 documento 50p → ask su capitolo X → verifica che le fonti risposte coprano la pagina attesa.
- Performance test: latenza medio < 2s per retrieval + 4s inferenza (dipende GPU).

Note finali e next steps rapidi

1. Mettere in piedi un PoC: 1 worker indexing, FAISS locale, Llama in inferenza con LoRA.
2. Implementare endpoint upload + ask e UI minimale (drag & drop + chat).
3. Validare con 10 documenti reali e 5 collaboratori.

Se vuoi, te genero anche il codice pronto `docker-compose.yml` + `dockerfile` per il PoC, e uno script `deploy_poc.sh` per avviare tutto su RunPod. Vuoi che lo prepari ora?