POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica
(Computer Engineering)



Tesi di Laurea Magistrale

Agentic AI applicata ai Processi Industriali

Relatori

Prof. Paolo GARZA

Ing. Maurizio GALASSO

Candidato

Matteo FLORES

2 ottobre 2026

A zia Dea.

Sommario

Oggi i System Engineers sono costretti a redigere e revisionare documenti di requisiti di sistema, in ambito industriale, manualmente. Questo tipo di lavoro ripetitivo può essere non solo dispendioso in termini di tempo investito, ma anche soggetto a errore umano, il quale si riflette inevitabilmente in produzione. Documenti di requisiti scritti da diverse persone che usano diverse terminologie possono risultare a prima vista coerenti, quando in realtà sono presenti forti inconsistenze (o viceversa). Inoltre, è necessario valutare la correttezza terminologica e tecnica dei requisiti e la loro copertura rispetto al progetto di cui fanno parte.

I requisiti devono anche essere coerenti rispetto ai macchinari utilizzati, citati nei documenti FMEA, al fine di isolare i guasti e ridurre al minimo il rischio per i lavoratori. Inoltre, i requisiti devono essere validati e verificati da test, e pertanto i System Engineers hanno anche il compito di controllare manualmente la copertura dei requisiti rispetto ai test che li validano.

In questa Tesi, viene dimostrato come la GenAI possa automatizzare questo tipo di mansioni, permettendo all'ingegnere di concentrarsi su compiti di maggiore complessità e più ad alto livello. Durante il mio tirocinio curriculare presso Accenture, ho preso parte allo sviluppo di tre tool (e tre agenti AI costruiti attorno ad essi):

- Rana (Requirement ANAlyzer), industrializzazione di una precedente PoC, consente l'analisi di Correctness, Consistency e Completeness;
- FMEA, Proof of Concept che dimostra le potenzialità della GenAI per l'analisi di documenti FMEA (Failure Mode and Effects Analysis)
- V&V, Proof of Concept che dimostra le potenzialità della GenAI per l'analisi di documenti Test Cases, Test Objectives e Test Sessions.

I tre tool sono stati sviluppati secondo le convenzioni dell'Ingegneria del Software, ognuno di mette a disposizione un server MCP che permette all'agente AI di comunicare con il backend, svolgendo di fatto la stessa funzione dei router nelle chiamate API REST.

Indice

Elenco delle tabelle x

Elenco delle figure xi

1 Dagli LLM alla Agentic AI 1
- 1.1 Le prime forme di Large Language Models . . . 1
- 1.2 Tokenization . . . 3
- 1.3 Transformers . . . 4
- 1.4 Architetture comuni di LLM . . . 9
- 1.5 Fine-tuning e Prompt Engineering . . . 9
  - 1.5.1 Tecniche Base di Prompting . . . 12
  - 1.5.2 Tecniche di Ragionamento (Reasoning) . . . 14
  - 1.5.3 Pattern Architetturali Avanzati . . . 16
- 1.6 Agenti . . . 16
  - 1.6.1 Workflows . . . 16
  - 1.6.2 Agents - processo decisionale . . . 17
  - 1.6.3 Schemi di interazione tra agenti . . . 19
- 1.7 Model Context Protocol . . . 19
- 1.8 Agent2Agent Protocol . . . 22
- 1.9 Gli agenti AI applicati al mondo reale . . . 23

2 GenAIX Framework 25
- 2.1 AIHub . . . 25
  - 2.1.1 System Prompt . . . 27
  - 2.1.2 Gli Agenti . . . 28
- 2.2 Artifacts . . . 30
- 2.3 Knowledge . . . 30
  - 2.3.1 La semantica delle frasi . . . 30
  - 2.3.2 La struttura del Knowledge . . . 30
  - 2.3.3 Embedding di record di database . . . 31
  - 2.3.4 Clustering . . . 32

3 Rana, FMEA, V&V 35
3.1 SSO . . . 36
3.2 Rana . . . 36
3.2.1 Input: Requisiti, CSN, TSG . . . 39
3.2.2 Data Model e caricamento del file di input . . . 42
3.2.3 Analisi: metriche, score e reasoning . . . 43
3.2.4 Correctness . . . 48
3.2.5 Consistency . . . 51
3.2.6 Completeness . . . 52
3.2.7 Voting . . . 55
3.3 FMEA . . . 57
3.3.1 Input: FMEA document, Grading Definitions, Actions, Mitigations, P-diagram . . . 57
3.3.2 Data Model . . . 58
3.3.3 Analisi: metriche, score e reasoning . . . 59
3.3.4 Correctness . . . 60
3.3.5 Consistency . . . 61
3.4 V&V . . . 63
3.4.1 Input: Requisiti, Test Objectives, Test Cases Test Sessions . 65
3.4.2 Data Model . . . 65
3.4.3 Analisi: metriche, score e reasoning . . . 66
3.4.4 Correctness . . . 66
3.4.5 Consistency . . . 67

4 Agenti 69
4.1 Rana Agent . . . 70
4.1.1 Rana MCP server - funzionalità di base . . . 70
4.1.2 Funzionalità aggiuntive . . . 74
4.2 FMEA Agent . . . 80
4.3 V&V Agent . . . 81

5 Valore di business 83
5.1 Necessità del progetto . . . 83
5.2 Analisi dei Tempi (Time-to-Value) . . . 84
5.2.1 Correctness . . . 84
5.2.2 Consistency . . . 86
5.2.3 Completeness . . . 88
5.2.4 Confronto con il System Enigneer . . . 88
5.3 Analisi dei Costi (Ottimizzazione) . . . 90
5.3.1 Correctness . . . 90
5.3.2 Consistency . . . 91

5.3.3 Completeness 92
5.3.4 Confronto con il System Engineer 92

6 Conclusioni e progetti futuri 95
6.1 Il nuovo ruolo del System Engineer 95
6.2 Progetti futuri 96

A Rana - metriche e prompt 97
A.1 Consistency by project 97
A.2 Consistency between projects 99
A.3 Consistency versus libraries 103

B Agenti - Prompts 107
B.1 Rana 107
B.2 FMEA 112
B.3 V&V 114

Bibliografia 117

Elenco delle tabelle

3.1 Rana - Use Cases Covering (UC-AIREQ) 37
3.2 Esempio di slide *Solution configurations* 41
3.3 Esempio della differenza tra Must e Want, per uno SHR. 43
3.4 Interpretazione del campo *FMEA Range*. 55
3.5 FMEA - Use Cases Covering 57
3.6 V&V - Use Cases Covering 64

5.1 Latenza media per requisito dei principali workflow di Rana. 84
5.2 Confronto tra il tempo medio stimato di revisione manuale da parte di un Junior e Senior System Engineer e il tempo medio di esecuzione di Rana. 89
5.3 Confronto del Time To Value (TTV) tra un Senior System Engineer e Rana. 89
5.4 Costo medio per requisito e numero medio di token di output dei principali workflow di Rana. 90
5.5 Costo stimato per le eseguire le analisi analisi complete su un intero progetto, per un Junior SE, Senior SE e Rana. 92

Elenco delle figure

1.1 Esempio di utilizzo di LLM basato su RNN per tradurre *it is indeed sunny today* dall'inglese all'italiano. 2
1.2 Struttura del transformer 5
1.3 PE, frase 1 6
1.4 PE, frase 2 6
1.5 Meccanismo di *attention* 7
1.6 Calcolo del vettore in output 7
1.7 Struttura del transformer 8
1.8 Injection degli Adapters nel modello: i layer di Feed Forward e Multi-Head Attention sono congelati, mentre gli Adapter e i Layer Norm sono addestrati a posteriori. 11
1.9 Esempio di Self Consistency con Chain-of-Thoughts. 15
1.10 A sinistra, il flusso di un'applicazione basata su LLM. A destra, un'applicazione basata su agenti. 17
1.11 Esempio di Router Chain. 17
1.12 Processo decisionale degli agenti. 18
1.13 Interazione tra agenti: schema Mixed 19
1.14 Figure di interazione del protocollo MCP 20
1.15 Schema di funzionamento del protocollo MCP: interazioni tra le parti 21
1.16 Schema di funzionamento del protocollo A2A 22

2.1 [5]Struttura interna dell'agente ritornato dalla funzione create_agent di LangChain. 29
2.2 Operazione di clustering all'interno del Knowledge 33

3.1 Rappresentazione grafica della chiamata ai tool per mezzo del delegation token 38
3.2 Analisi che Rana può eseguire sui requisiti. 38
3.3 Parentela tra requisiti. 40
3.4 Schema ER di Rana. 42
3.5 Linking degli SHR con i CSN ad opera del Knowledge. 44

3.6 Interfaccia del SEBot che permette il lancio delle analisi. 44
3.7 Parallelizzazione delle analisi con l'utilizzo dei semafori. 46
3.8 Componente dell'interfaccia del SEBot che permette di abilitare il meccanismo di voting. 56
3.9 Schema ER di FMEA. 59
3.10 Schema ER di V&V. 65

4.1 Interfaccia della chat per interagire con il Rana Agent. 71

Capitolo 1

Dagli LLM alla Agentic AI

1.1 Le prime forme di Large Language Models

Come espresso nel corso di Large Languages Models for Software Engineering al Politecnico di Torino[1], i **Large Languages Models** (da qui in poi riportati con la sigla ***LLM***) sono modelli probabilistici di linguaggi naturali. La definizione formale si basa sul meccanismo di predizione della parola successiva, sulla base di tutte quelle precedentemente generate:

$$P(w_1, w_2, ..., w_{T-1}, w_T) = \prod_{t=1}^{T} P(w_t | w_{t-1}, w_{t-2}, ..., w_2, w_1) \tag{1.1}$$

Stando a questa definizione, è pertanto necessario definire un (ampio) vocabolario di parole, in modo da essere in grado di assegnare, a ciascuna di esse, la probabilità con la quale verrà "pescata". Inizialmente, le probabilità assegnate alle parole, venivano calcolate per mezzo di **Feed-Forward Neural Network** (FFN), composta da diversi layer lineari e non lineari, che portavano alla generazione di cosiddetti *logit* (un punteggio assegnato alle parole), che a loro volta venivano dati in pasto alla funzione di *softmax* per trasformarli in probabilità, rispettando i vincoli:

$$0 \leq P(w_1) \leq 1 \qquad \sum_i w_i = 1 \tag{1.2}$$

Tuttavia, questa formulazione del problema si rivelò, col tempo, inadatta a diversi compiti linguistici:

- Un vocabolario con molti elementi richiede alti costi computazionali per via del calcolo delle probabilità.
- Le parole non presenti nel vocabolario (*Out-of-vocabulary*, OOV) hanno probabilità nulla di essere scelte.

Un altro problema frequente, ad esempio, è l'individuazione di lettere maiuscole: un LLM è addestrato a trasformare le lettere minuscole in maiuscole, ove necessario, sarebbe inefficace. Difatti, stando a quanto affermato finora, l'LLM è in grado di generare parole basandosi unicamente su quelle precedenti. Il seguente esempio evidenzia al meglio il problema:

> [...] upon his pale face. "it may be so, or it may not", **m**r. holmes said he, "but if you are so very sharp [...]

La lettera *m* di *mr.* non sarà mai convertita in maiuscola siccome fino al momento in cui il modello ha "letto" quella lettera dal testo originale, non saprà mai che subito dopo ci sarà *-r.* .
Problema ancora più importante, per via di come sono costruite le reti neurali di tipo FFN, l'input e l'output hanno lunghezza fissa (si pensi al numero di neuroni in ingresso e in uscita). Inoltre, come ogni rete neurale, soffre di problemi di calcolo numerico causati dall'aggiornamento dei pesi, quali il *vanishing gradient* e *l'exploding gradient*.
Dunque si è passati all'introduzione di reti neurali ricorrenti (**Recurrent Neural Networks**), che prevedono la presenza di uno stato latente *h*, gestito dalla Long Short Term Memory (LSTM), e che prendono in input la parola attuale e lo stato dell'iterazione precedente, producendo la parola predetta e lo stato aggiornato. Infatti, con questa tipologia di reti neurali, l'input può avere lunghezza variabile. Tuttavia, i problemi legati alla backpropagation dell'aggiornamento dei pesi del modello persistono. In aggiunta, il processo non può ancora essere parallelizzato.

Come mostrato in Figura 1.1, il modello non produce un risultato corretto.

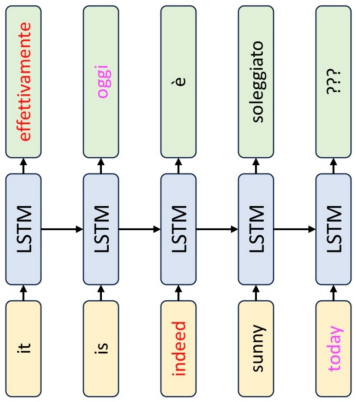


Figura 1.1: Esempio di utilizzo di LLM basato su RNN per tradurre *it is indeed sunny today* dall'inglese all'italiano.

Nonostante l'introduzione delle RNN, alcuni compiti rimangono privi di soluzione.

Si pensi alla traduzione di frasi da una lingua all'altra: per via della struttura delle RNN, a una parola in input corrisponde una e una sola parola in output: come è noto, questa associazione viene meno quando si traduce da una lingua all'altra, in quanto a parità di significato, le parole di frasi in due lingue diverse potrebbero non avere una corrispondenza 1-a-1. Di seguito un esempio di traduzione dall'inglese all'italiano:

> it is indeed sunny today → effettivamente oggi è soleggiato

1.2 Tokenization

Finora si è parlato impropriamente di *parole* in input e in output al modello linguistico: in realtà il testo non viene dato in pasto direttamente alle reti neurali, in quanto sono in grado di analizzare solamente numeri; il passaggio che permette di convertire il testo in numeri decimali prende il nome di **tokenization**[1]. La suddivisione delle frasi in token può avvenire in diversi modi:

- **Character-level tokenization**: ogni carattere costituisce un token. Esempio:

  ```
  M r s . ␣ R u c a s t l e ␣ w a s ␣ d o w n s t a i r
  s , ␣ s o ␣ I ␣ h a d ␣ a n ␣ a d m i r a b l e ␣ o p
  p o r t u n i t y .
  ```

  Il vantaggio di utilizzare questa tecnica di tokenization è evidente: il problema della generazione di parole nuove, non esistenti nel vocabolario (Out-of-vocabulary, OOV), non si pone più, in quanto la nuova parola sarà composta da caratteri che sono già presenti nel vocabolario stesso. Tuttavia risulta estremamente inefficiente, considerando anche che, molte parole che si ripetono spesso in frasi diverse, sono sempre trattate come singoli caratteri, perdendo così la possibilità di gestirle come unico token. Inoltre, è molto più complesso catturare il significato semantico delle singole parole e delle intere frasi.

- **Word-level tokenization**: ogni parola, spaziatura e segno di punteggiatura costituiscono un token. Esempio:

  ```
  Mrs . ␣ Rucastle ␣ was ␣ downstairs , ␣ so ␣ I ␣ had ␣
  an ␣ admirable ␣ opportunity .
  ```

In questo caso, il modello potrà gestire sequenze più piccole, riducendo drasticamente la complessità computazionale e rendendo anche più semplice catturare il significato semantico delle parole. Tuttavia, il problema delle parole fuori dal vocabolario persiste. Inoltre, parole con suffissi e prefissi avranno un token diverso rispetto a quelle di forma base. Infine, le parole più rare saranno inevitabilmente viste di meno in fase di training del modello, rendendolo più impreciso e soggetto a errore.

- **Subword-level tokenization**: il testo è suddiviso in sotto-parole. Esempio:

  `Mrs` `.` `␣` `R` `uca` `st` `le` `␣` `was` `␣` `downstairs` `,` `␣` `so` `␣` `I` `␣` `had` `␣` `an` `␣` `ad` `mir` `able` `␣` `opportunity` `.`

  Con quest'ultima tecnica, che risulta essere una via di mezzo tra le due precedentemente presentate, consente di gestire correttamente le parole Out-of-vocabulary ottimizzando la generalizzazione ma senza peggiorare drasticamente l'efficienza. In sostanza, questa tecnica di tokenization cerca di raggruppare perlopiù sillabe che si presentano con maggiore frequenza. Esistono varie tecniche per stabilire come accorpare le lettere per formare sillabe frequenti: un metodo comunemente utilizzato è il Byte-Pair Encoding, semplice da applicare ma ha bisogno di un *corpus*, ossia una collezione di testi al fine di valutare la frequenza delle sillabe.

1.3 Transformers

L'introduzione del concetto di *attention*[1], alla base dei **transformer**, rappresenta un punto di svolta nel mondo degli LLM: infatti, non solo abbandona del tutto la logica per cui a una parola in input ne corrisponde una e una sola in output, ma permette anche di parallelizzare l'intero processo di generazione.

La struttura di base dei transformer, rappresentata in Figura 1.2, è principalmente composta da due blocchi:

- **encoder**, che prende in input i token e restituisce il contesto semantico
- **decoder**, che dal contesto generato dall'encoder produce, sequenzialmente, i token di output

Positional Embedding

Passando i token in input direttamente all'encoder, il risultato è che le seguenti sequenze in input (considerando, per semplificazione, una parola per token) risultano essere uguali dal punto di vista del modello:

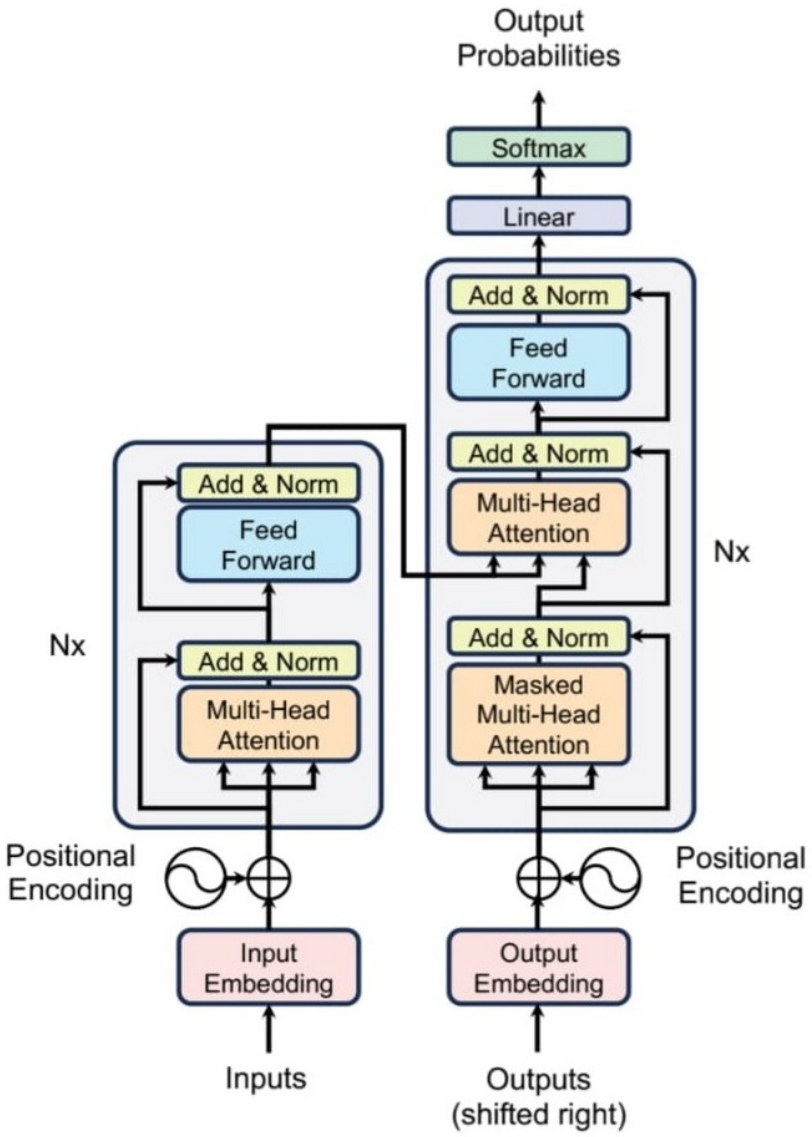
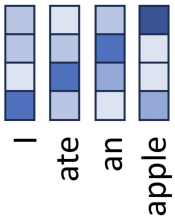
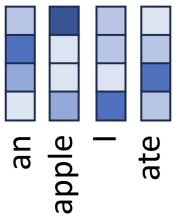


Figura 1.2: Struttura del transformer

1. I ate an apple



2. an apple I ate



Siccome l'ordine delle parole cambia il significato della frase, i token in input non passano subito nell'encoder, ma vengono prima dati in pasto a un blocco chiamato *positional embedding*, il quale si occupa di aggiungere una componente *posizionale* (appunto, i *positional vectors*) ai token. Come si può notare dalle Figura 1.3 e Figura 1.4), il risultato a valle dell'applicazione del PE è diverso.

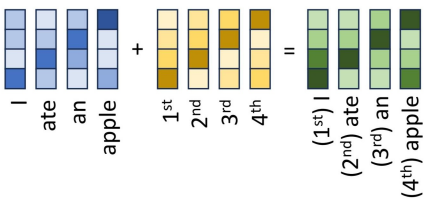


Figura 1.3: PE, frase 1

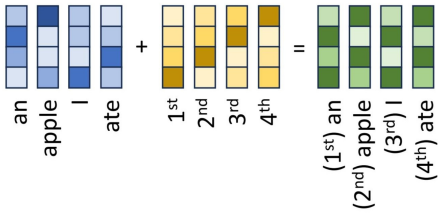


Figura 1.4: PE, frase 2

Esistono diverse tecniche di PE, la più comune è la *Sinusoidal Positional encoding*. Il PE preserva la similarità tra vettori di token di posizioni vicine,

Attention

Il concetto di **attention** (o *attenzione*) è il punto di svolta che segna il passaggio dagli LLM basati su FFN a quelli basati su transformer, utilizzati oggigiorno. Questo meccanismo, permette di *decidere* che peso dare a ogni vettore (token) in input

(appunto, *attenzione*). Una versione semplificata del meccanismo di attenzione è riportata in Figura 1.5.

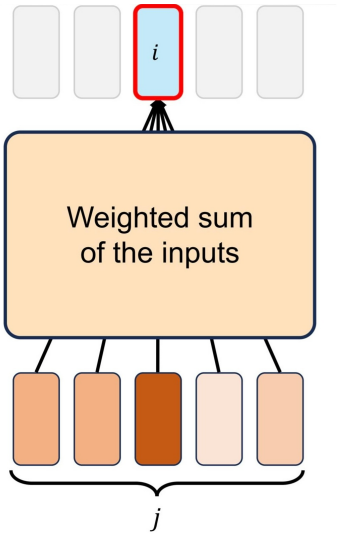


Figura 1.5: Meccanismo di *attention*

$$out_i = \sum_j Attn_weight(i,j) f(in_j)$$

Figura 1.6: Calcolo del vettore in output

Pertanto, come si nota dalla formula rappresentata in Figura 1.6, l'output è, in sostanza, determinato dai pesi.

L'attention layer è, in realtà composto da tre gruppi di input:

- Queries (Q), i valori da cercare
- Keys (K), i valori che devono essere cercati rispetto alle queires
- Values (V), i valori associati a ogni chiave

Questi 3 input vengono estratti dall'input originale del modello, attraverso delle apposite matrici (W^Q, W^K e W^V), i cui valori sono stabiliti in fase di addestramento del modello stesso (Figura 1.7).

Alla fine, il calcolo che esegue il meccanismo di attention sulla base di queries, keys e values è il seguente:

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.3}$$

dove d_k rappresentala dimensione dei vettori Q e K.

Come si nota in Figura 1.2, l'attenzione è utilizzata in tre situazioni:

- **Encoder self-attention** - ogni token *fa attenzione* a tutti gli altri token della sequenza in input.

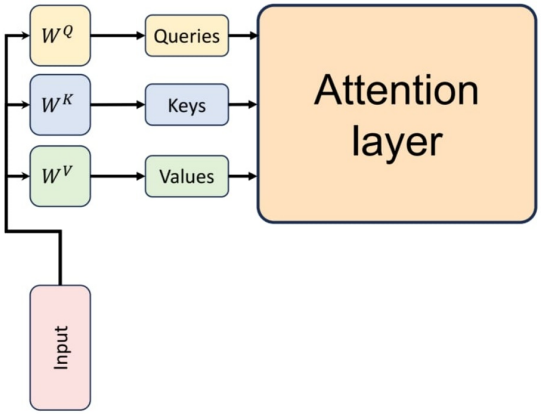


Figura 1.7: Struttura del transformer

- **Decoder (masked) self-attention** - siccome i token in output vengono estratti sequenzialmente, la loro generazione deve dipendere unicamente dal risultato dell'encoder e dai token precedentemente prodotti in output dal decoder. Di conseguenza, è necessario applicare una maschera per evitare di *fare attenzione* anche sui token non ancora generati:

$$MaskedAttention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}} + Mask\right)V \tag{1.4}$$

- **Encoder-decoder cross-attention** - usata dal decoder, prende keys e values dall'output dell'encoder e le queries dalla sequenza del decoder.

Ogni blocco di *attention* è chiamato *attention head* (Figura 1.7). Come illustrato anche in Figura 1.2, può essere necessario utilizzare più *attention head* in parallelo, dettato dalla necessità di *fare attenzione* a contesti diversi. Per questo motivo, un'ulteriore matrice W^O si occupa di unire i risultati delle varie teste.

Resitual Connections e Layer Normalization

Per aggirare i problemi legati alla backpropagation (Sezione 1.1), i transformer applicano due tecniche:

- Per abbattere il *vanishing gradient*, si ricorre al concetto di *residual connections* (precedentemente introdotto dalle *Residual Networks*), permettendo di aggiungere l'input dei blocchi di attention al loro output.
- Per superare l'*exploding gradient*, viene applicata una normalizzazione del singolo token rispetto al layer (*Layer Normalization*), riducendo così valori troppo elevati e difficili da gestire.

Il blocco di ***Add&Norm*** è inserito, all'interno dei transformer, nel contesto dei blocchi di *attention* e FFN (Figura 1.2).

1.4 Architetture comuni di LLM

Con l'introduzione dei transformer, si è passati all'utilizzo di architetture basate su encoder-decoder[1]; la più comune è il Text-to-Text Tranfer Transformer (**T5**), un singolo modello per diversi compiti linguistici: infatti, non è più necessario eseguire un ulteriore addestramento del modello specifico per un determinato compito (*finetuning*), ma è sufficiente condizionare l'input a tal scopo. Esempio:

> Input: translate english to german what is you profession?
> Output: <pad>Was ist dein Beruf?</s>

Tuttavia, T5 non è un modello adatto alle conversazioni, è addestrato solo per alcuni compiti.
Per altri compiti linguistici, invece, il decoder risulta superfluo, introducendo i cosiddetti modelli *encoder-only*: tali modelli solo utilizzati per compiti come classificazione di testi, analisi sentimentale, ecc... Tra i modelli *encoder-only* si distingue BERT (Bidirectional Encoder Representations from Transformers).
Per altri compiti ancora, invece, risulta necessario solo il decoder (modelli *decoder-only*): tali modelli ricevono in input una sequenza di token e la estendono; la generazione dell'output, in questo caso, è *autoregressiva*. Tra i modelli *decoder-only*, si distinguono i modelli **GPT** (Generative Pre-Trained Transformer), che si prestano ad essere ottimi modelli per chat.
Nello sviluppo degli agenti presentati nei capitoli successivi, sono stati utilizzati proprio i modelli della famiglia GPT (da GPT-5 in poi).

1.5 Fine-tuning e Prompt Engineering

I modelli di LLM sono addestrati su compiti generici, di conseguenza è necessario applicare tecniche di allo scopo di "specializzare" il modello per il compito che sta per affrontare[1].
Una di queste è il classico **fine-tuning** utilizzato nel mondo del Machine Learning classico: si tratta di un addestramento a posteriori, in cui tutti i pesi del modello subiscono un aggiornamento a valle dell'addestramento generico. Chiaramente si tratta della metologia migliore in termini di performance (comparabili all'addestramento da zero per quel compito specifico). Tuttavia, questo tipo di addestramento ha costi computazionali elevati e risulta dispendioso se applicato a ogni compito specifico.
Il **feature-based transfer**, invece, è una tecnica specifica di fine-tuning in cui non

tutto il modello viene ri-addestrato, ma solo gli strati più esterni, vicini all'output: pertanto, la parte di estrazione del contesto (chiamata *backbone*) viene *congelata*, in modo che la parte che si occupa di generare i token (chiamata *head*), presente negli strati più esterni, viene riaddestrata indipendentemente dal resto del modello. Le performance di questa tecnica sono molto simili al fine-tuning del modello nella sua interezza, ma compiti complessi richiedono comunque modifiche più in profondità del modello; inoltre, il costo computazionale resta comunque elevato.
Alcune tecniche di fine-tuning riducono le limitazioni computazionali appena riportate, prendendo il nome di ***Parameter-efficient Fine-Tuning (PEFT)***:

- **BitFit** (*Bias-term Fine-tuning*): è un fine-tuning specifico solo per il termine di bias b degli strati lineari:
$$y = Wx + b \tag{1.5}$$

- **Adapters**: sono nuovi strati del modello che vengono addestrati a posteriori insieme agli strati di Layer&Norm, lasciando il resto del modello congelato (Figura 1.8). Gli Adapters sono semplicemente costituiti da un down-projecting layer, un layer di non-linerity e un up-projecting layer.

- **LoRA** (*Low Ranking Adaptation*): l'addestramento qui riguarda la parte incrementale di aggiornamento dei parametri:
$$W' = W + \Delta W \tag{1.6}$$
dove ΔW è in genere una matrice *near low-rank* fattorizzabile con due matrici A, B più piccole, riducendo dunque il numero di parametri da aggiornare. Questa tecnica permette di aumentare i benefici all'aumentare della dimensione del modello; ad esempio, con GPT-3 avremmo fino 10 mila parametri in meno.

- **Prompt tuning**: invece di addestrare il modello o parte di esso (*Model tuning*), fornire informazioni extra nel testo di input talvolta è sufficiente per condizionare il modello stesso e generare un output di qualità.

Ad oggi, il prompt tuning è maggiormente utilizzato in quanto è il metodo computazionalmente più efficiente di applicare del *tuning* al modello linguistico senza un riaddestramento esplicito e quindi senza aggiornamento dei parametri dello stesso.

Il prompt tuning[2] si basa sul concetto di **prompt**, ossia sequenze di istruzioni e contesto fornite in input al modello linguistico, allo scopo di indirizzarlo verso un compito specifico. I principi fondamentali per la scrittura di un buon prompt sono:

- **Specificness**: più specifico è il prompt, più accurato sarà l'output

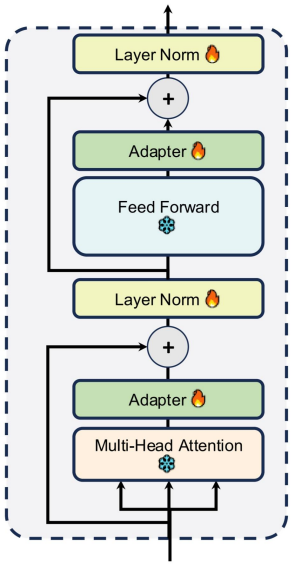


Figura 1.8: Injection degli Adapters nel modello: i layer di Feed Forward e Multi-Head Attention sono congelati, mentre gli Adapter e i Layer Norm sono addestrati a posteriori.

- **Step-wise approach**: suddividere il testo del prompt in parti più piccole migliora le prestazioni del modello
- **Iterate and improve**: il prompt-tuning consiste nel provare diversi input e migliorarli, iterativamente.

Il **prompt engineering** consiste nell'ottimizzare la struttura dei prompt al fine di ottenere risultati migliori. Con il tempo, si è andata a definire una struttura di prompt che ad oggi risulta essere lo standard de-facto:

- Instructions: definiscono il comando all'LLM, ossia il compito per cui riceve l'input.
- Context: informazioni aggiuntive utili a definire risposte più accurate.
- Input data: l'input reale dell'utente, che dovrà essere elaborato per ottenere l'output.
- Output indicator.

Di seguito alcune delle tecniche più note e ancora oggi adottate nel prompt engineering (Sottosezione 1.5.1, Sottosezione 1.5.2, Sottosezione 1.5.3), che possono anche essere adottate in maniera combinata.

1.5.1 Tecniche Base di Prompting

Per dare specificità al modello linguistico ed evitare che generi una risposta troppo diversa dal risultato atteso, in genere si adotta la tecnica del *priming*, che si applica introducendo input addizionale per guidare il modello nella generazione. Esempio: Senza *priming*:

> Write a story about a detective solving a mystery.

Con *priming*

> Imagine a world where technology is advanced, and detectives use futuristic gadgets to solve crimes. The city is filled with neon lights and towering skyscrapers, creating a cyberpunk atmosphere. The detective in this story is known for his sharp mind and unorthodox methods. He's tasked with solving a mystery involving a missing scientist and a powerful corporation hiding dark secrets.

Un'altra tecnica di prompting è il Fill-in-the-blank Prompting, che consente all'utente di concentrarsi su un aspetto in particolare del testo, lasciando all'LLM il compito di completare gli spazi *blank*, dando dei suggerimenti. Esempio:

> Create a story about a detective in *[choose a setting: e.g., a futuristic city, a small coastal town, a bustling international airport]*. The detective, *[choose a name for the detective]*, is known for *[describe a characteristic or skill: e.g., keen observation skills, a knack for technology]*. The story begins with *[set the scene: e.g., an unusual disappearance, a cryptic message left behind]*. As *[detective's name]* delves into the case, they use *[mention a tool or method: e.g.:, advanced gadgets, old-fashioned deduction]* to uncover clues. Through *[describe a key location or detail in the investigation]*, *[detective's name]* discovers *[reveal a twist or critical piece of information]*. However, they face *[introduce a challenge or obstacle: e.g., a powerful adversary, a race against time]*. How does *[detective's name]* navigate this situation? Tell the story of how they solve the mystery.

Per dare una struttura chiara all'LLM di come trattare il testo, il prompt può essere composto dalle seguenti sezioni esplicite:

- **Role**: il ruolo dell'LLM
- **Result**: output desiderato
- **Goal**: scopo dell'output

- **Context**: informazioni aggiuntive che rispondono a: chi? che cosa? dove? perché?
- **Constraints**: limitazioni e linee guida

Esempio:

> Role: You are a tasked with writing a compelling narrative about a detective who embarks on a mission to uncover the truth behind a perplexing mystery.
> Goal: Your objective is to engage the reader with a gripping tale that revolves around the detective's journey from initial discovery to solving the intricate puzzle.
> Context: The story is set in a bustling city where a prominent public figure has been found murdered in their home, leaving behind cryptic clues that point to a web deceit.
> Result: As the story progresses, reveal how the detective meticulously pieces together the evidence, confronts suspects, and unravels the hidden motives behind the crime.
> Constraint: Despite facing numerous red herrings and conflicting testimonies, ensure the narrative maintains suspense and coherence, leading to a satisfying resolution that ties all lose ends.

A volte fornire degli esempi di input e output desiderati può guidare il modello linguistico verso risultati migliori. In base al numero di esempi, si parla di:

- **Zero-Shot**: nessun esempio fornito nel prompt. Il modello utilizza le proprie conoscenze per completare il compito.
- **One-Shot**: un singolo esempio descritto nel prompt.
- **Few-Shot**: più esempi o linee guida riportati nel prompt.

Esempio di few-shot prompting:

> The odds number in this group add up to an even number: 4, 8, 9 ,15, 12, 2 1.
> Answer: the answer is false
>
> The odds number in this group add up to an even number: 17, 10, 19, 4, 8, 12, 24.
> Answer: the answer is true
>
> The odds number in this group add up to an even number: 16, 11,

14, 4, 8, 13, 24.
Answer: the answer is true

The odds number in this group add up to an even number: 9, 10, 12, 13, 4, 2, 1.
Answer: the answer is false

The odds number in this group add up to an even number: 15, 32, 5, 13, 72, 7, 1.
Answer: ?

1.5.2 Tecniche di Ragionamento (Reasoning)

Usando tecniche di one-shot o few-shot prompting in cui viene esplicitato il ragionamento che porta al risultato (Chain of Thought Prompting), viene forzato in output il *reasoning*, ossia il processo logico che ha portato alla produzione di un certo risultato, in modo da ridurre gli errori. Esempio:

Prompting standard

Input:
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How may tennis balls does he have now?
A: The answer is 11.
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many abbles do they have?

Output:
A: The answer is 27. → wrong answer

Chain-of-Thought Prompting

Input:
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How may tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Output:
A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. → correct answer

Alcuni compiti sono più complessi di altri, pertanto potrebbe non esserci un unico CoT corretto. In questi casi, si ricorre al Self consistency con CoT (CoT-SC, Figura 1.9), in cui vengono eseguiti più CoT in parallelo e la risposta con maggior frequenza è considerata quella corretta.

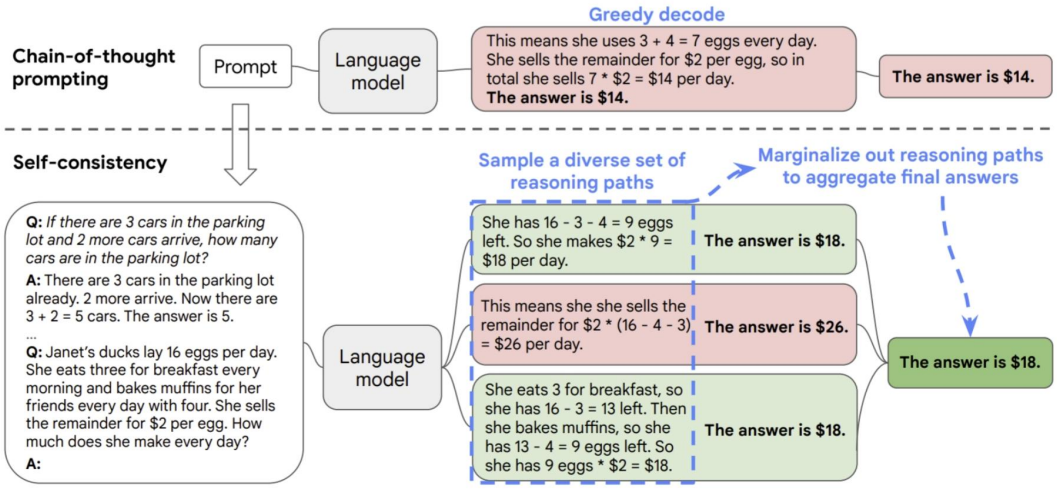


Figura 1.9: Esempio di Self Consistency con Chain-of-Thoughts.

Un'estensione ulteriore del CoT è il Tree of Thoughts (ToT), la cui esecuzione avviene mediante i seguenti passaggi:

1. Thought decomposition: mentre con il CoT classico non vi è una decomposizione esplicita in piccoli step, con il ToT il problema è decomposto in piccole porzioni di pensiero.
2. Thought generation: vengono generati k candidati per il prossimo step di pensiero
3. State evaluator: un *evaluator* stabilisce, per ognuno degli stati generati, lo stato di completamento verso l'obiettivo finale.
4. Search algorithm: algoritmi di tree search (come il Breadth-First-Search o il Depth-First Search) vengono utilizzati per selezionare il percorso ottimale

Per analizzare il codice, invece, si utilizza la Structrued Chain of Thought, tecnica particolare di CoT utilizzata per la generazione di codice (dunque, testo strutturato, come iterazioni, selezioni, eccetera).

1.5.3 Pattern Architetturali Avanzati

Può capitare che il testo prodotto dal modello diverga molto dall'argomento della richiesta, generi risultati che hanno poco senso o fornisca informazioni false: in questo caso, si parla di **allucinazioni**. Al fine di ridurle, si ricorre alla Retrieval-Augmented Generation (**RAG**), un *design pattern* architetturale che, oltre a mitigare le allucinazioni, agisce anche su obsolescenza, affidabilità o confusione del modello. Il pattern consiste nel ripescare informazioni da una *knowledge source*, che in genere è rappresentata da artefatti o database, anche in locale, spesso legata semanticamente al prompt, in modo da fornire al modello informazioni aggiuntive di contesto e sempre aggiornate.

Reasoning and Acting (**ReAct**) è, invece, un *design pattern* comportamentale, in cui, attraverso tecniche di prompt engineering, viene istruito il modello a:

- Pensare (Reasoning): espone "a voce alta" il ragionamento che fa prima di eseguire ogni tipo di operazione concreta
- Agire (Acting): esegue quanto pianificato nella fase precedente

Come la RAG, ReAct è utilizzato per ridurre le allucinazioni.

1.6 Agenti

Le applicazioni tradizionali basate su LLM hanno un flusso predefinito; ad esempio, parlando di RAG, le informazioni vengono estratte in maniera passiva e poi fornite al modello in automatico. Naturalmente, flussi predefiniti limitano enormemente il potenziale degli LLM.
Flussi dinamici, invece, sono decisi dagli **agenti** (Figura 1.10), i quali differiscono dai modelli precedenti in quanto decidono autonomamente le azioni da intraprendere per arrivare all'output.

La scelta dinamica dei flussi, permette di utilizzare componenti di memoria o strumenti (chiamati **tool**), quando l'agente lo ritiene necessario: questo dinamismo, permette anche di gestire compiti più complessi.

1.6.1 Workflows

I **workflows** sono sequenze di nodi (anche dette *chains*) predefinite in cui, pertanto, non vi è alcun processo decisionale. Le chains possono essere:

- Sequential Chain: il caso più semplice di catena di chiamate all'LLM, in cui ci sono N nodi che hanno come input l'output del nodo precedente, e producono l'input del nodo successivo.

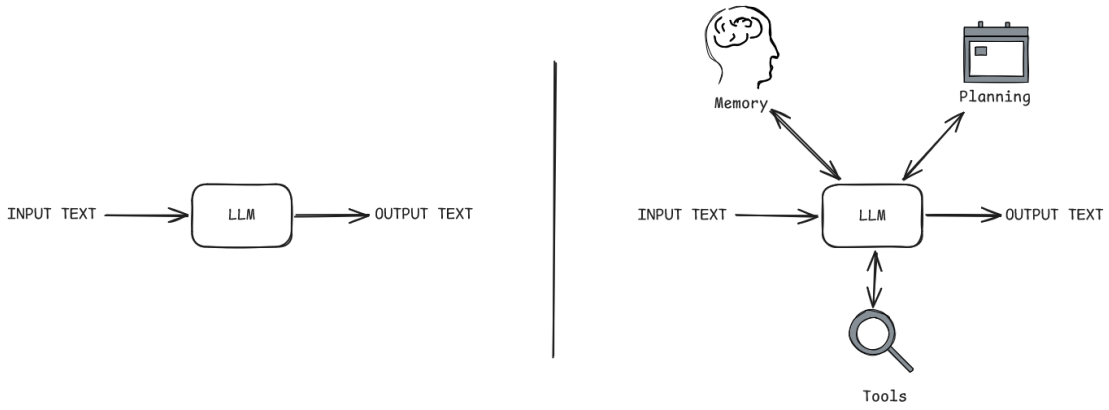


Figura 1.10: A sinistra, il flusso di un'applicazione basata su LLM. A destra, un'applicazione basata su agenti.

- Tree Chain: sequential chains in cui vengono gestiti input multipli al fine di generare pontenzialmente diversi output. La struttura, infatti, è quella tipica di un albero.

1.6.2 Agents - processo decisionale

Il processo decisionale entra in gioco introducendo la Router Chain (Figura 1.11), in cui un nodo rappresenta il *decision maker* che, in base al tipo di compito che deve gestire, decide in quale delle Destination Chains il flusso deve proseguire; pertanto, nel momento in cui entra in gioco un nodo con potere decisionale sul flusso di esecuzione, si parla di agente.

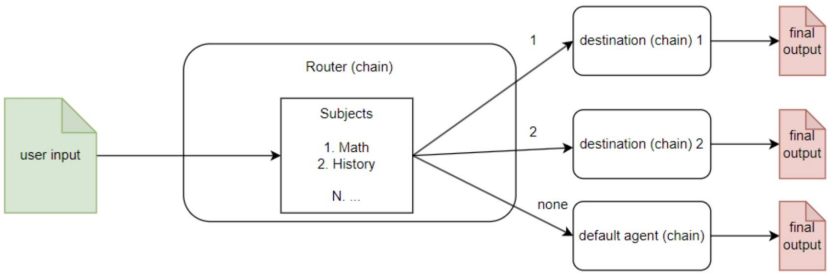


Figura 1.11: Esempio di Router Chain.

Facendo un esempio di differenza tra LLM semplice e agente, alla domanda:

What's the weather like?

nel caso di una semplice chiamata all'LLM la risposta sarà composta da informazioni sulle attuali condizioni meteorologiche. L'agente può, in aggiunta all'informazione principale sul meteo, consigliare l'abbigliamento ideale, se è il caso di portare l'ombrello con sè, suggerire attività all'aperto o al chiuso, al mare o in zone più riparate da forti venti.

Un'altra novità introdotta dagli agenti è il superamento dei classici Directed Acyclic Graph, permettendo di introdurre delle catene cicliche (passando quindi alle State Machines) che rislutano essere molto utili in processi di revisione e approvazione del risultato generato.

Aumentando il livello di astrazione, possiamo dire che l'agente rappresenta il *decision maker* che stabilisce gli obiettivi, pianifica, e infine agisce (Figura 1.12).

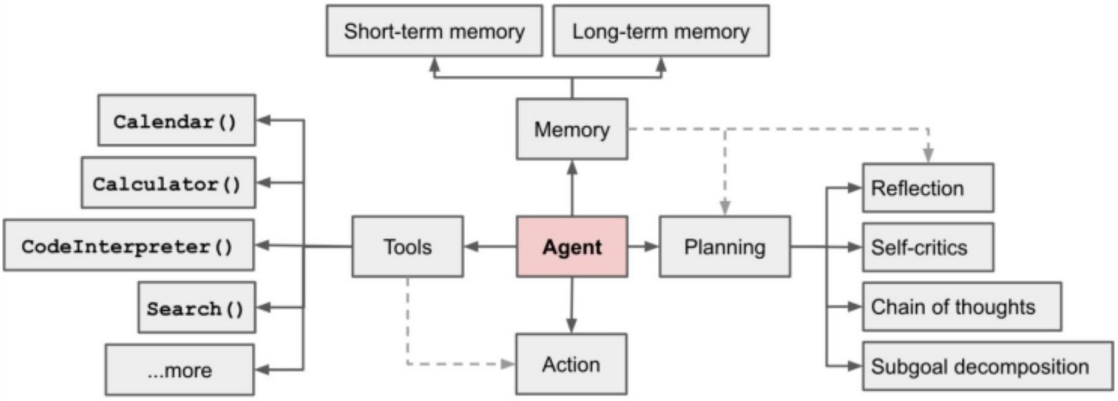


Figura 1.12: Processo decisionale degli agenti.

In base al compito da gestire, più o meno complesso, l'agente può servirsi di:

- **Tools**: estensioni che ampliano le capacità dell'agente. Esempi possono essere Calendar per impostare eventi sul calendario, Calculator per eseguire operazioni di tipo numerico, CodeInterpreter per eseguire analisi di porzioni di codice, Search per ottenere informazioni in tempo reale dalla rete globale.

- **Memory**: è divisa in Short-term memory, che mantiene il contesto della conversazione, e Long-term memory, che contiene informazioni persistenti, potenzialmente utili successivamente (come, per esempio, le preferenze dell'utente).

- **Planning**: modulo che permette la pianificazioni delle azioni. In genere è composto da: Reflection (che valuta le azioni precedenti e identifica margini di miglioramento a valle di un processo complesso), Self-critics (che valuta il grado di bontà dell'azione intrapresa nell'immediato), Chain of thoughts (che

gestisce il ragionamento sequenziale, step-by-step), Subgoal decomposition (che gestisce la scomposizione dell'obiettivo finale in piccoli sotto-problemi.

1.6.3 Schemi di interazione tra agenti

Gli agenti possono interagire tra di loro seguendo i seguenti schemi:

- **Cooperative**: gli agenti si prefiggono un obiettivo comune e cooperano al fine ultimo del raggiungimento di tale obiettivo; a tale scopo, comunicano tra di loro e condividono informazioni per eseguire i loro rispettivi compiti. Tra le informazioni condivise, vi sono i feedback di esecuzione, al fine di ottimizzare l'intero processo.
- **Adversarial**: gli agenti lavorano indipendentemente per massimizzare il loro interessi. Gli agenti, in questo caso, formulano dei piani di strategia competitiva per ottenere risultati migliori degli altri rivali.
- **Mixed** (Figura 1.13): questo schema unisce quello Cooperative e Adversarial, attraverso la forma Parallel o Hierarchial: la prima prevede che gli agenti collaborino indipendentemente senza interferire gli uni con gli altri, pur condividendo informazioni; la seconda, invece, prevede una struttura di agenti ad albero, in cui il nodo *parent* stabilisce obiettivi e sotto-compiti, i quali vengono eseguiti dai nodi *child*.

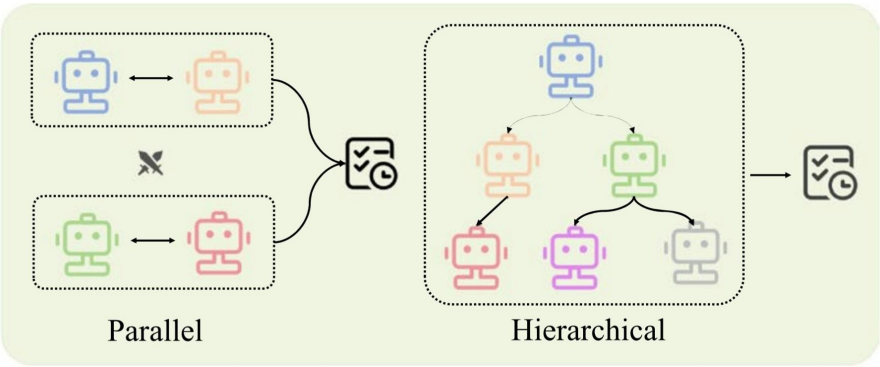


Figura 1.13: Interazione tra agenti: schema Mixed

1.7 Model Context Protocol

Dalla nascita degli agenti, l'architettura alla base della comunicazione degli agenti con i tools e tra diversi agenti era affidata direttamente al team che si occupa

dell'intero sviluppo degli agenti.
Nel 2024, l'azienda americana di AI Anthropic, ha pubblicato il **Model Context Protocol** (**MCP**)[3], protocollo open source che stabilisce le interazioni tra gli agenti e i tools a livello architetturale. Ad oggi, è diventato uno standard de-facto largamente adottato in questo campo.
Come spiegato proprio da Anthropic, MCP è considerato la porta USB delle applicazioni basate su AI: infatti, rappresenta proprio il ponte di connessione tra l'applicazione e i sistemi esterni (Figura 1.14).

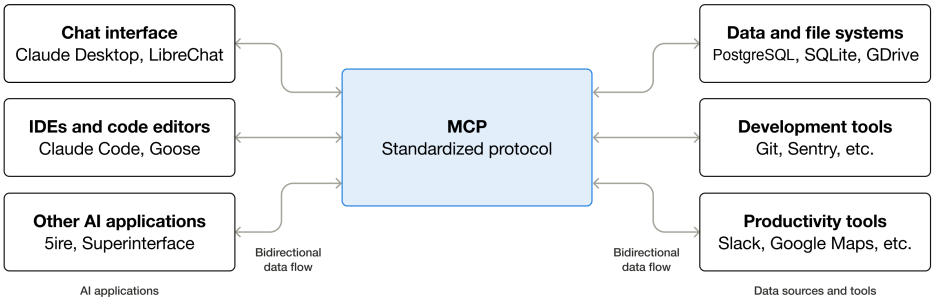


Figura 1.14: Figure di interazione del protocollo MCP

Il protocollo si basa su architettura client-server, in cui gli attori che interagiscono sono (Figura 1.15):

- **MCP Host**, che si occupa di gestire uno o più MCP Client.
- **MCP Client**, che si connette a un MCP Server.
- **MCP Server**, che accetta le connessioni di uno o più MCP Client, a cui fornisce contesto.

Ad esempio, l'MCP Host è rappresentato da Visual Studio Code, noto IDE, che stabilisce una connessione con il Sentry MCP Server: VSCode, dunque, avrà un'istanza di MCP Client. Quando prova a connettersi a un altro MCP Server, per esempio un server locale che gestisce il filesystem, verrà avviata una nuova istanza di MCP Client.
Il protocollo è diviso in due layer:

- **Data layer**: stabilisce il protocollo da utilizzare per la struttura dei messaggi è il JSON-RPC 2.0 (JavaScript Object Notation - Remote Procedure Call), che permette al client di far eseguire procedure al server, alternativa più leggera rispetto alle API REST.

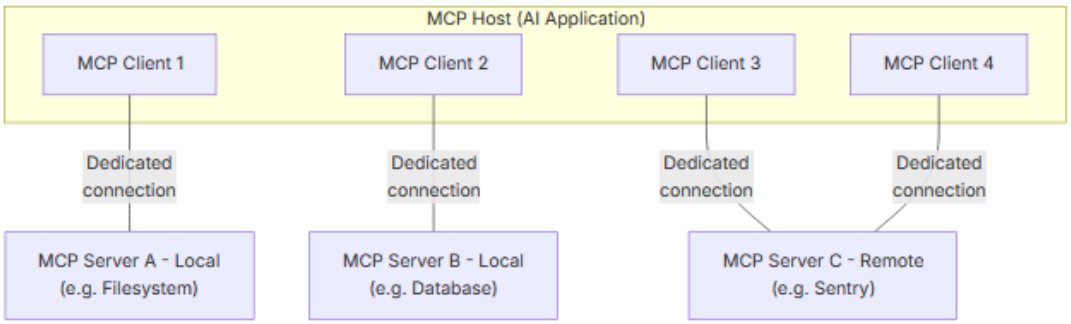


Figura 1.15: Schema di funzionamento del protocollo MCP: interazioni tra le parti

- **Transport layer**: stabilisce le modalità di trasmissione dei messaggi tra le parti, che può essere:
  - stdio transport: canali standard di input e output, come file di testo o binari.
  - streamable HTTP transport: consente di trasmettere le informazioni tramite streaming, spesso avviati per mezzo di chiamate HTTP POST o eventi (quindi utilizzando message brokers come Kafka).

Le funzionalità esposte dagli MCP Server sono:

- **Tools**: funzioni che l'LLM può chiamare direttamente; possono essere chiamate API, query al database o attivazioni di altre logiche.
- **Resources**: dati in sola lettura, come contenuti di file, documentazioni API o schemi di database.
- **Prompts**: template predefiniti di prompt da utilizzare all'occorrenza, scritti per Tools e Resources.

Gli MCP Client, invece, oltre a utilizzare le features esposte dai Server, espongono ai Server a loro volta:

- **Elicitation**: permette ai server di richiedere specifiche informazioni all'utente.
- **Roots**: comunicano al server su quali cartelle concentrarsi, consentendo di definire uno *scope*.
- **Sampling**: permette al Server di utilizzare l'LLM tramite il Client per alcuni compiti, abilitando di fatto l'*agentic workflow*.

1.8 Agent2Agent Protocol

Nonostante l'introduzione del protocollo MCP sia innovativo nel mondo dell'AI, la comunicazione tra più agenti non risultava ancora ben definita: a tal scopo, Google ha introdotto nel 2025 l'**Agent2Agent Protocol** (**A2A**)[4].
Questo protocollo, analogamente a MCP, è diventato uno standard-de-facto complementare al primo, che introduce le modalità di comunicazione tra due o più agenti ben distinti. Come mostrato in Figura 1.16, permette di definire la comunicazione tra due agenti potenzialmente sviluppati da due persone diverse, con architettura e strutture diverse.

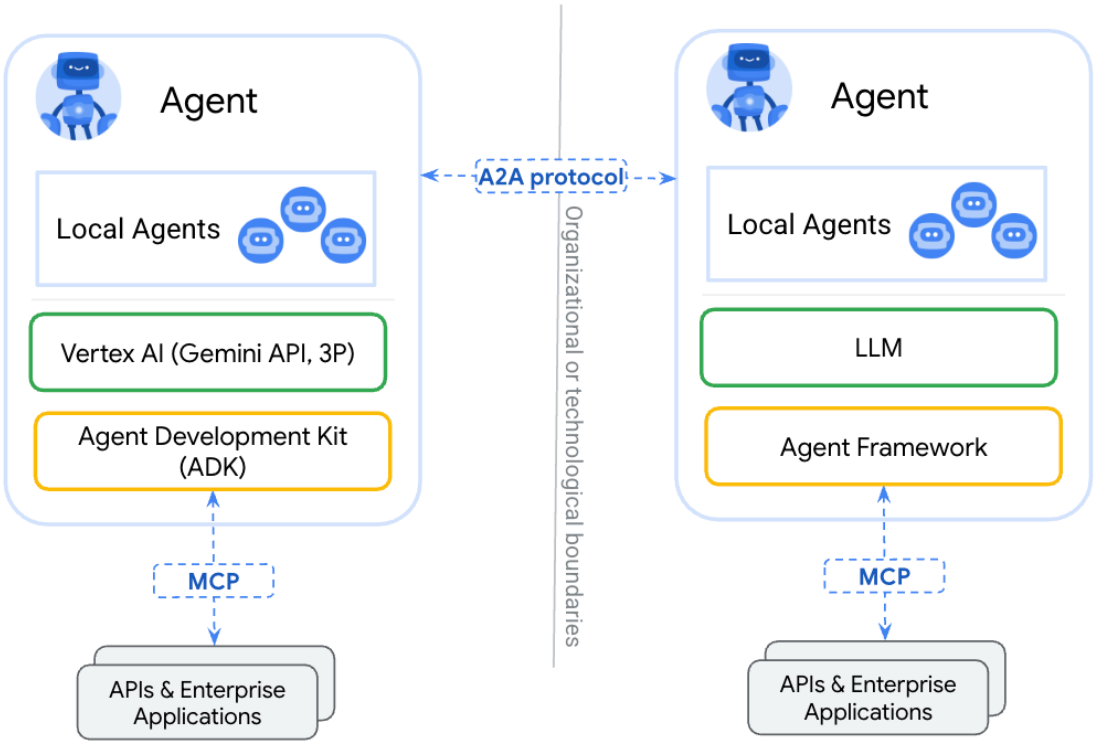


Figura 1.16: Schema di funzionamento del protocollo A2A

Prendendo in esame l'esempio presente nella documentazione ufficiale di Google, un utente che desidera pianificare un viaggio internazionale ha bisogno di orchestrare diversi agenti: un *flight booking agent*, un *hotel reservation agent*, un *agent for local tour recommendations* e un *currency conversion agent*. Senza A2A si presenterebbero diversi problemi:

- Senza un protocollo di comunicazione, verrebbe naturale al programmatore incapsulare i due agenti come tool per permettere lo scambio di informazioni, come avviene con l'utilizzo di MCP. Questa soluzione risulta efficace, tuttavia riduce drasticamente il potenziale degli agenti, la cui natura è quella di comunicare direttamente.
- Ogni interazione richiede tipologie di messaggi ad hoc per ogni coppia di agenti, perdendo così la capacità di generalizzare il problema.
- Aumentando il numero di agenti, sorge anche il problema di scalabilità: infatti, sarebbe difficile coordinare ogni coppia di agenti, operazione potenzialmente soggetta a frequenti errori di implementazione, riducendone anche l'interoperabilità.
- Le comunicazioni ad hoc spesso mancano di una comunicazione sicura, il che risulta estremamente critico per il traffico di dati sensibili.

Il protocollo A2A risolve i problemi sopra elencati, introducendo uno standard basato su altri preesistenti, come HTTP, JSON-RPC. Il ciclo di vita della richiesta A2A è composto da: *agent discovery*, *authentication*, *sendMessage* API e *sendMessageStream* API. I due attori principali sono l'A2A Client, agente AI che comunica con il Server per mezzo di A2A, e l'A2A Server, il quale espone endpoint HTTP e processa le richieste dei Client. Le interazioni possono avvenire per mezzo di **polling** (il Client chiede periodicamente al Server lo stato di completamento della richiesta), **streaming con Server-Sent Events (SSE)** o **push notifications**, a seconda della tipologia di task e della durata del processo.

1.9 Gli agenti AI applicati al mondo reale

Pur trattandosi di una tecnologia relativamente recente, l'AI risulta già ampiamente adottata non solo dal mondo tech, ma anche dal mondo industriale più in generale. Le aziende italiane e mondiali stanno vivendo una trasformazione dei prodotti e processi produttivi mediante il supporto sempre più presente e attivo dell'intelligenza artificiale.
L'introduzione dei protocolli MCP (Sezione 1.7) e A2A (Sezione 1.8) ha permesso l'introduzione degli agenti AI nei processi industriali, riuscendo a portare a termine con successo compiti che richiedono un livello di complessità, per cui i semplici LLM fallivano. Nel caso analizzato in questa tesi, gli agenti hanno permesso di migliorare i processi di produzione, in particolare a supporto di documenti per i requisiti di produzione, analisi e prevenzione di guasti e validazione a valle del processo produttivo, aumentando la velocità e limitando al minimo l'errore umano, come approfondito nei capitoli successivi.

Capitolo 2

GenAIX Framework

Nel periodo in cui ho svolto il tirocinio presso la società di consulenza Accenture, ho avuto l'opportunità di entrare a contatto con realtà internazionali per quanto riguarda l'integrazione della Generative AI (quindi degli agenti) nei processi industriali. Prima di entrare nel dettaglio del progetto di cui mi sono occupato (Capitolo 3), è necessario fare un approfondimento su uno strumento indispensabile nel corso del mio lavoro.
Il team di progettazione e sviluppo ha messo a punto, prima ancora che iniziasse la mia attività in azienda, un framework "general purpose", che offrisse una base tecnica di partenza da cui iniziare con lo sviluppo del progetto commissionato dai diversi clienti: **GenAIX** (dove la lettera *X* è presa in prestito dalla divisione *Industry X* di Accenture) è composto da diversi sottomoduli, ognuno di essi con una specifica funzione. I diversi sottomoduli non sono considerati dei microservizi in senso classico come noto dall'ingegneria del software, ma piuttosto delle sottoporzioni di codice di un unico progetto, specializzati in un particolare ambito della Generative AI (in questo caso, si dice che GenAIX è un **cookie cutter**). I sottomoduli comunicano tra di loro per mezzo di chiamate HTTP/REST (interne alla rete), permettendo di fatto il loro disaccoppiamento. Di seguito vengono esposte le funzioni dei principali sottomoduli presenti nel framework, utilizzati nel progetto commissionato dal cliente.

2.1 AIHub

Il sottomodulo **AIHub**, come suggerisce il nome, costituisce il cuore delle applicazioni AI che utilizzano GenAIX, ed è utilizzato principalmente per l'orchestrazione degli agenti. Come gli altri sottomoduli, l'AIHub è un componente **thread-safe**, il che consente all'applicativo finale di prestarsi molto bene all'utilizzo *multiutente*. Ad esempio, se l'utente A utilizza l'agente `genaix_agent` e l'utente B fa partire

subito dopo una nuova richiesta, anche per lo stesso agente, il modulo è in grado di gestire le due richieste in maniera separata e coerente. Il risultato finale, consiste in due processi con risultati coerenti con le singole richieste che non interferiscono a vicenda.
Siccome l'applicazione è nata per essere *event-driven*, in questo sottomodulo il framework offre un middleware che consente di inviare messaggi nei *topic* attraverso il message broker **Kafka**.
Il processo che passa attraverso l'AIHub si compone dei seguenti passaggi:

- Un altro sottomodulo o il frontend creano un thread, fornendo all'AIHub il `template_id` e il `configuration_id`.
- Lo stesso sottomodulo esegue un'operazione di *trigger* attraverso una chiamata `POST`, al fine di avviare il thread appena creato. Esempio di input passato al trigger:

```
{
    "type": "human_interaction",
    "thread_public_id": "",
    "context_public_id": "",
    "meta": { "prompt": "Hello, agent!" }
}
```

- L'AIHub lancia il processo (agente o workflow) associato a quel thread.
- L'AIHub esegue tutte le operazioni richieste, per cui il thread esiste.
- L'AIHub invia dei messaggi Kafka per aggiornare i *consumer* (che può essere la UI, il control panel o un altro *consumer*) sullo stato dell'esecuzione (in particolare, su attività che coinvolgono il `TodoActivityMiddleware`, il `SummaryActivityMiddleware` e l'`ActivityStreamMiddleware`).
- L'esecuzione termina con successo, rispondendo al chiamante, il quale può interrogare lo stato attraverso le chiamate API alle rotte `processes` e `states`.

Le interazioni con l'AIHub con l'esterno (ad esempio, con il frontend) avvengono per mezzo di chiamate API REST (utilizzando il modulo Python `FastAPI`). Le comunicazioni con gli altri moduli avvengono per mezzo del `GenAIXClient`, i cui metodi sono invocati direttamente dagli altri moduli, i quali permettono di effettuare chiamate API al modulo interessato e di utilizzare thread, trigger, workflow e agenti.
Come anticipato, questo sottomodulo è anche usato per definire **workflow** e **tool**,

ma il suo scopo ultimo è quello di definire **agenti**, i loro comportamenti, le loro funzionalità e le interazioni con altri sistemi. Ogni agente del framework (quello "nativo", `genaix`, e quelli implementati *ad-hoc* per il progetto, esposti successivamente), è il principale orchestratore, che decide se e come interagire con tool e altri LLM.

2.1.1 System Prompt

L'AIHub definisce una struttura comune di system prompt, il **Base Prompt**, utilizzato dagli agenti definiti nel sotto-modulo. Il Base Prompt prompt è suddiviso in **3 layer** logici.
Il prompt inizia sempre con l'**Identity Prompt [Layer 1]**, il quale definisce i principi generali dell'agente, lo stile comunicativo e l'utilizzo (generico) dei tool:

Identity Prompt

"You are GenAIX Agent — a general-purpose execution agent for the GenAIX platform. Your primary job is to execute whatever the user asks, using all available tools at your disposal." [...]

Quello che impone l'Identity Prompt, di fatto, è il comportamento che deve avere l'agente rispetto alle richieste dell'utente, che si può riassumere in:

- Esecuzione diretta delle richieste dell'utente.
- Utilizzo prioritario di tool rispetto all'LLM, dove possibile.
- Presentazione dei risultati in modo diretto e conciso.

All'Identity Prompt segue sempre l'**Output Standards Prompt [Layer 2]**, il quale a sua volta è composto da:

- **Formatting Prompt**, che definisce il formato dell'output (Markdown), le regole di struttura e presentazione delle risposte in output (ad esempio, per tabelle, elenchi puntati, sezioni, comandi, porzioni di codice).
- **Citation Prompt**, che definisce le regole per le citazioni di documenti e immagini.
- **Security Prompt**, che definisce le regole di sicurezza da adottare. Ad esempio, l'agente non dovrà mai fornire, in risposta all'utente, credenziali, token API keys, o file d'ambiente (`.env`).

Successivamente è presente il **Tool Execution Prompt [Layer 3]**, che espone le direttive generali per la chiamata e l'esecuzione dei tool. Questo prompt è a sua volta composto da:

- **General Tool Execution Prompt**, che istruisce il modello a utilizzare i tool solo quando necessario, definisce l'utilizzo di tool in task multi-step, il riconoscimento di esecuzioni di tool non andate a buon fine e l'interpretazione dei risultati.
- **Blocking Tools Prompt**, che istruisce il modello a chiamare il tool e aspettare il risultato, trattandosi di una chiamata bloccante.
- **Detached Tools Prompt**, che istruisce il modello a comunicare il progresso dell'esecuzione, trattandosi di chiamate non bloccanti ai tool.
- **Handoff Prompt**, che definisce il comportamento dell'agente rispetto a deleghe dell'esecuzione ad altri agenti o tool.
- **Stateful Tools Prompt**, che istruisce il modello a utilizzare le informazioni *stateful* tipiche dei tool in questione.
- **Workflow Tools Prompt**, che istruisce il modello a chiamare workflow che al loro interno eseguono workflow.

Il Base Prompt appena definito costituisce l'elemento comune a tutti gli agenti definiti nel framework. Ciò che, invece, distingue gli agenti gli uni dagli altri è il **Role Prompt [Layer 4]**, ossia la porzione di prompt che definisce l'obiettivo, il dominio e la "persona" dell'agente in questione. Nel caso dell'agente presente nella versione di default del framework, `genaix_agent`, in quanto agente generico, è una semplice indicazione per l'agente a seguire le richieste dell'utente, prediligendo l'esecuzione dei tool alla chiamata diretta all'LLM.

2.1.2 Gli Agenti

La struttura dell'agente è definita utilizzando i nodi e gli archi di LangChain[5], che fornisce un'architettura basata su macchine a stati finiti. In particolare, la funzione `create_agent` di LangChain costruisce una struttura di base rappresentata in Figura 2.1: la richiesta dell'utente (sottoforma di `HumanMessage` di LangChain) viene passata all'LLM, che, nel caso del progetto, fa parte della famiglia GPT, il quale decide se il compito è di carattere generico e quindi può fornire lui stesso una risposta, oppure se è il caso di utilizzare uno dei tool a disposizione; nel caso in cui vengano utilizzati i tool, il modello decide poi se è opportuno chiamare altri tool o procedere con la risposta all'utente. I tool forniti all'agente sono l'insieme dei *local tools* e di quelli ottenuti dai server MCP con cui il client sta comunicando (è il caso di Rana, FMEA, e V&V, come riportato nel Capitolo 3).

All'agente vengono anche passati i middleware progettati per l'agente stesso. Tra questi, risulta molto importante il `GenAIXToolWrapMiddleware`, un semplice wrapper che permette l'utilizzo dei tool nelle modalità:

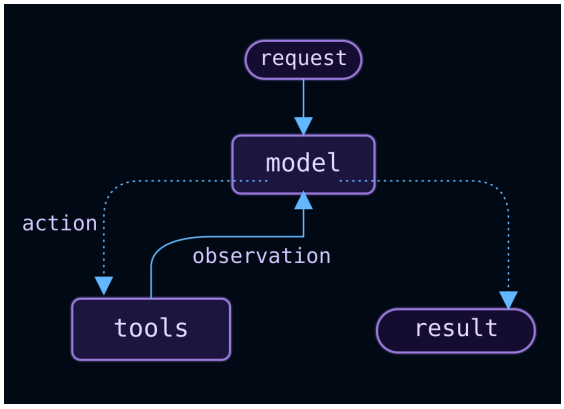


Figura 2.1: [5]Struttura interna dell'agente ritornato dalla funzione create_agent di LangChain.

- Blocking: l'agente resta in attesa finché il tool non ha completato il suo processo e inviato una risposta al chiamante.
- Detached: a differenza della modalità precedente, permette un utilizzo *asincrono* del tool, pertanto risulta *non-bloccante*.
- Handoff: il controllo viene ceduto dall'agente chiamante a un altro agente.

Altri middleware degni di nota sono:

- **`ToolCompletionMiddleware`**, che segnala il completamento di un tool chiamato in modalità *detatched*.
- **`ActivityStreamMiddleware`**, che intercetta le esecuzioni chiave di agenti e workflow e pubblica eventi Kafka su eventi di attività (come chiamate ai tool, reasoning, ecc...). Viene utilizzato per monitorare l'attività di agenti e workflow da frontend.
- **`TodoActivityMiddleware`**, che eredita la classe built-in `TodoListMiddleware` di LangChain che gestisce tool injection, state management, system prompt injection e parallel-call guardrail. Questo middleware aggiunge in più alla classe ereditata la pubblicazione di eventi su topic Kafka.
- **`SummaryActivityMiddleware`**, che intercetta quando il middleware built-in di LangChain `SummarizationMiddleware` ha aggiornato la chiave `conversation_summary` e pubblica un evento su Kafka.

2.2 Artifacts

Il sottomodulo **Artifacts** permette il salvataggio e la gestione dei cosiddetti *artifacts* (o *artefatti*), ossia file e documenti di diverso formato. Permette di disaccoppiare la logica principale del progetto in sviluppo rispetto alla gestione dei file. I principali tipi di file supportati sono quelli del pacchetto Office (`.doc`, `.docx`, `.xls`, `.xlsx`, `.ppt`, `.pptx`), PDF (`.pdf`), immagini (`.png`, `.jpeg`, `.gif`) e cartelle compresse (`.zip`).
Nel progetto in questione, in particolare, questo sottomodulo viene utilizzato per il salvataggio di un file di tipo `pptx`, il cui scopo è approfondito nella Sezione 3.2.

2.3 Knowledge

2.3.1 La semantica delle frasi

È noto che, invertendo l'ordine delle parole di una frase (come discusso nella Sezione 1.3) o semplicemente cambiandone la punteggiatura, il significato può essere completamente diverso. Ad esempio, la seguente frase riporta lo stato d'animo dei viaggiatori:

> Most of the time**,** travelers worry.

Rimuovendo la virgola, non si parla più di viaggiatori, bensì di viaggiatori nel tempo:

> Most of the time travelers worry.

Per dare una rappresentazione matematica alla semantica delle frasi, permettendo pertanto anche il confronto tra le stesse, si ricorre agli **embedding**, ossia sequenze di numeri decimali (anche detti *vector*) che catturano il significato della frase. La loro generazione avviene ad opera di una categoria speciale di modelli di LLM ed è, al contrario della normale generazione di LLM, deterministica (pertanto, a parità di modello utilizzato, due frasi avranno la stessa rappresentazione vettoriale).

2.3.2 La struttura del Knowledge

Il sottomodulo **Knowledge** gestisce una parte di AI di cui l'AIHub non si occupa: infatti, il suo scopo principale è la gestione di embedding di vari tipi di risorse.

Le funzionalità di base di questo sottomodulo consentono, tra le varie operazioni, l'**ingestion** di documenti presenti in **Artifcats** (Sezione 2.2):

1. Il documento selezionato viene estratto da Artifacts per mezzo delle operazioni esposte dal sottomodulo.

2. L'artefatto viene diviso in *chunk*.
3. Per ogni chunk prodotto, viene generato l'embedding associato.
4. Gli embedding prodotti vengono salvati nel database.

L'ultima operazione tra quelle appena riportate racchiude dentro di sè operazioni *ad-hoc* per gli embedding, che sono quasi completamente nascoste al programmatore: infatti, il salvataggio non avviene all'interno di un classico database relazionale come PostgreSQL o non relazionale come MongoDB, ma si tratta di una famiglia di database specializzati nella gestione degli embedding che prende il nome di vector-store; nel nostro caso, utilizziamo **Qdrant**, un vector-store specializzato nella ricerca e filtraggio degli embedding ad alto *throughput*, ossia con un elevato traffico di dati (in previsione di un frequente utilizzo da parte di diversi utenti). Qdrant raccoglie gli embedding per partizioni logiche chiamate **collections** (o collezioni, in italiano). Nel caso del Knowledge, utilizziamo le collezioni per un gruppo di documenti che fanno parte dello stesso gruppo (ad esempio, i documenti che riguardano i guasti dei macchinari). Durante l'operazione di ingestion, infatti, oltre all'identificativo dell'artefatto da gestire, viene passato anche l'indice della collezione (*vector-index*) a cui associare il (o gli) embedding da produrre.

Un'altra operazione che espone il Knowledge, analogamente all'ingestion, è la **retrieval**, ossia il recupero dell'embedding associato a un certo identificativo e facente parte di una certa collezione.

2.3.3 Embedding di record di database

Un'importante limitazione di questa versione base del Knowledge è che non gestisce l'ingestion e la retrieval di **record di database relazionali**: nel progetto in questione, infatti, è necessario generare e utilizzare embedding associati a campi tabellari anziché documenti veri e propri, come approfondito in Sezione 3.2, Sezione 3.3, Sezione 3.4. Pertanto, mi sono occupato di estendere il Knowledge implementando questa nuova funzionalità.

L'ingestion di record di database relazionale inizia con il recupero del (o dei) record in questione: la connessione al db avviene direttamente dal Knowledge verso, potenzialmente, un altro sottomodulo; pertanto, sono necessarie informazioni come la stringa di connessione al database, la tabella di riferimento, le chiavi primarie dei record di cui fare l'ingestion, e i campi della tabella da cui generare l'embedding (in quanto potenzialmente alcuni campi potrebbero essere superflui o addirittura fonte di disturbi che non permetterebbero di catturare la vera semantica di interesse). La scelta di utilizzare una connessione con interrogazioni dirette al database è motivata dal fatto che il framework è un cookie cutter, come precedentemente enunciato (Capitolo 2), a differenza delle classiche interazioni tramite chiamate API tra microservizi.

Nel progetto sviluppato per il cliente è emerso che questi record possono subire modifiche nel tempo. Pertanto, per evitare di eseguire nuove ingestion ogni volta che fosse necessario un embedding, ma al tempo stesso evitando di conservare embedding che contengono informazioni obsolete, è stato necessario aggiungere una logica di aggiornamento. Analogamente a come viene fatto in alcuni casi per l'invalidazione della cache, il chiamante indica al Knowledge su quali campi eseguire l'*hash*, ossia una stringa alfanumerica che rappresenta in modo univoco i campi di cui si vuole controllare l'aggiornamento.

Dunque, ogni volta che un sottomodulo deve effettuare l'ingestion di un record di una tabella di un suo database, il Knowledge controlla se il record in questione ha già subito un'ingestion in passato e, in caso positivo, calcola l'*hash* dei campi indicati del nuovo record e lo confronta con quello che ha salvato nel momento in cui è stato generato l'embedding: nel caso in cui non coincidano, si tratta di un aggiornamento e pertanto è necessaria la generazione di un nuovo embedding; altrimenti, il record non è stato aggiornato e viene saltato e segnato come "skipped".

2.3.4 Clustering

Come spiegato in seguito, la generazione di embedding di record si è rivelata indispensabile al fine di effettuare operazioni di **clustering** sui record stessi. Data la necessità di raggruppare semanticamente un numero non definito a priori di record, ho implementato una funzionalità in Knowledge che soddisfa questo bisogno.

Il sottomodulo che ha bisogno di *clusterizzare* un insieme di record di una tabella di un suo database, a seguito della fase di ingestion (Sottosezione 2.3.3), può chiamare il metodo `cluster_index_by_record_ids` dal `knowledge_client`, passandogli informazioni quali: l'indice della collezione, gli identificativi dei record da clusterizzare, l'algoritmo di clustering da utilizzare (in base agli algoritmi che mette a disposizione il Knowledge è possibile indicare il numero di cluster a priori o definirlo a valle dell'esecuzione del processo), i parametri dell'algoritmo di clustering. Quello che fa questo metodo del `ClusteringService` è eseguire l'algoritmo di clustering indicato dal chiamante (ad esempio, HDBSCAN, Sezione 3.2), utilizzando la **cosine similarity** (comunemente utilizzata per gli embedding) come metrica di distanza tra i record (rappresentati, appunto, dai loro embedding); il processo restituisce i cluster appena formati, rappresentati da una lista di dizionari con il numero progressivo del cluster in chiave e una lista di identificativi dei record come valori, e il numero di outliers, i quali costituiscono un cluster a parte, indicato come cluster di id `-1`. Una versione semplificata e ad alto livello del processo di clustering è rappresentata in Figura 2.2.

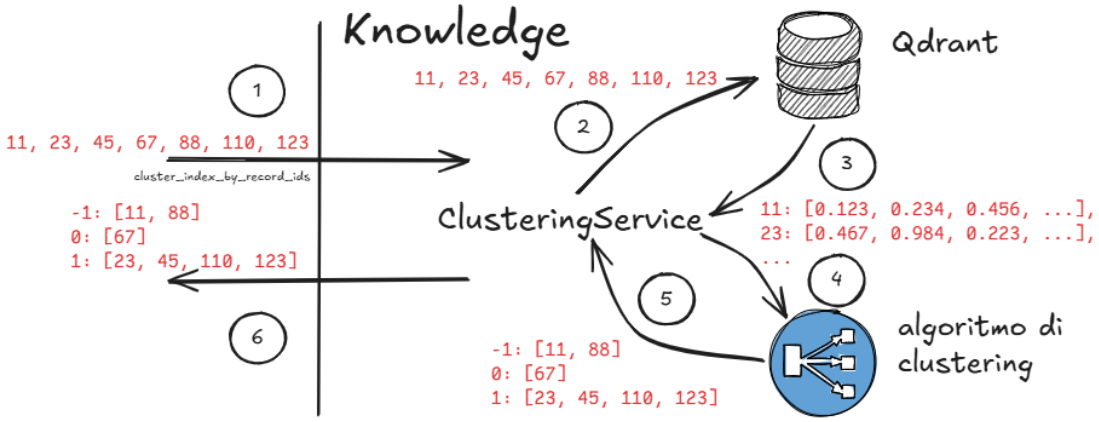


Figura 2.2: Operazione di clustering all'interno del Knowledge

Capitolo 3

Rana, FMEA, V&V

Il progetto commissionato dal cliente consiste nello sviluppo di tre tool in ambito industriale:

- **Rana**, che è l'industrializzazione di una precedente PoC (Proof of Concept), obiettivo principale del cliente, permette l'analisi di requisiti di sistema.
- **FMEA**, PoC che consiste nell'analisi di documenti FMEA che riportano le cause e gli effetti di guasti ai macchinari.
- **V&V**, PoC che consente di analizzare i test che permettono di validare requisiti di sistema.

Tutti e tre i tool fanno parte di tre sottomoduli che estendono GenAIX (Capitolo 2), e hanno i loro backend; in particolare, il backend di Rana comunica con un ulteriore sottomodulo che espone il frontend della piattaforma **SEBot (System Engineering Bot)**, permettendo ai System Engineer, in futuri rilasci del software, di muoversi agilmente da un tool all'altro.

Inoltre, ad ogni tool, è associato un **agente** il quale permette la comunicazione via chat con l'utente e di fatto esegue buona parte delle operazioni che si possono effettuare dalla GUI e altre operazioni che le limitazioni dell'interfaccia grafica non permettono di effettuare, come approfondito nel Capitolo 4.

Durante lo sviluppo del progetto, avviato a novembre 2025 e conclusosi ad agosto 2026, è stata seguita una metodologia **agile**, con incontri periodici con lo staff tecnico (di cui ho fatto parte) e lo staff funzionale (che ha fornito direttive a livello industriale), e ulteriori riunioni settimanali direttamente con il cliente, al fine di fornire aggiornamenti progressivi e ottenere feedback periodici per massimizzare la soddisfazione del cliente stesso e la piena riuscita del progetto. I feedback dell'utente sono stati facilitati dalla produzione di report in formato excel (da parte di Rana, FMEA e V&V), i quali riportano il contenuto del workspace e i dettagli sulle analisi eseguite sulle singole entità.

3.1 SSO

Siccome l'utilizzo di Rana (e, in future industrializzazioni, anche di FMEA e V&V) è strettamente legato all'utenza (Sottosezione 3.2.2), è stato necessario implementare un meccanismo di **Single Sign-On**, in modo tale che eseguire l'autenticazione al sistema aziendale del cliente direttamente dal browser, fosse sufficiente per accedere anche al SEBot e al backend di Rana con la stessa utenza.

Il modulo che si occupa di gestire l'orchestrazione degli utenti è l'**IdentityService**, che al suo interno espone chiamate HTTP per registrare utenti e recuperare le loro informazioni. Per ogni utente vengono salvati: l'identificativo intero progressivo (`user_id`), il `public_id` (UUID), il provider (in fase di sviluppo è il realm di GenAIX di Keycloak, in fase di deploy è Azure), il nome, l'email.

Tutti gli endpoint di Rana sono protetti da OIDC (con protocollo OAuth 2.1): pertanto, se l'utente non risulta autenticato, la risposta sarà sempre `401 Unauthorized`.

La protezione non si limita solo agli endpoint di chiamate API dei tre sottomoduli, ma si estende anche ai tool MCP definiti negli stessi. Poiché gli agenti devono poter accedere al workspace dell'utente e devono eseguire analisi per conto dell'utente, è necessario che agiscano per conto dell'utente che ha effettuato il login. Pertanto, l'AIHub ottiene lo `user_id` (e altre informazioni come `name` e `email`) dell'utente che si è autenticato dal SEBot e costruisce un **delegation token** ad hoc, contenente questa informazione, per eseguire le chiamate ai tool interessati. L'MCP Server di Rana ha il compito di validare il token (tramite la chiave simmetrica che condivide con l'AIHub) ed estrarre lo `user_id`, necessario per eseguire l'operazione richiesta, la quale è strettamente legata all'utente. L'introduzione del delegation token è motivata dal fatto che il JWT che si trova in SEBot non viene passato a AIHub e poi a Rana, con il rischio di essere letto da soggetti malevoli e quindi compromesso. Il processo è rappresentato in Figura 3.1.

Non sono state necessarie ulteriori misure di sicurezza, in quanto il sistema è stato rilasciato nell'infrastruttura del cliente, il quale usufruirà del servizio esclusivamente all'interno della stessa.

3.2 Rana

Come anticipato, **Rana** (da Requirement ANAlyzer), risulta essere l'industrializzazione di una prima versione del tool, in cui è stata valutata la fattibilità della sua realizzazione.

Il cliente ha inizialmente fornito gli Use Cases di Rana (Tabella 3.1), i cui obiettivi sono riportati in sintesi nella Figura 3.2.

Tabella 3.1: Rana - Use Cases Covering (UC-AIREQ)

	UC	Use Cases Description	Analysis Mode	Correctness	Consistency	Completeness
UC-AIREQ	UC1	**Requirements Cleaning and Standardization.** Ensure all requirements are clear, consistent, complete, and compliant with international standards and best practices in requirements engineering.	By Project	✓	✓	✓
	UC2	**Requirement Pre-validation.** Verify that SHRs comply with client's Requirement Validation procedure (criteria Proper, Clear, Verifiable/Testable).	By Project	✓	✓	✓
	UC3	**SHR–SR Cascade.** Ensure System Requirements (SRs) are correctly derived from Stakeholder Requirements (SHRs), avoiding duplication and scope creep.	By Project	✓		
	UC4	**SR–SSR Cascade.** Ensure Sub-System Requirements (SSRs) are correctly derived from System Requirements (SRs).	By Project	✓		
	UC5	**IR Consistency.** Ensure Interface Requirements (IRs) clearly define an agreement between linked systems.	By Project	✓	✓	✓
	UC6	**Requirement Libraries Internal Consistency.** Verify that requirements within client's libraries do not contradict each other.	By Project / Between Project		✓	
	UC7	**Project Requirements to Libraries Consistency.** Ensure project requirements do not conflict with requirements from the official libraries.	Versus Libraries		✓	
	UC8	**Automatic Generation of Project Requirements Proposal.** Automatically generate a first draft of project requirements based on standard specification template.	Agent			
	UC9	**Missing SR Cascade.** Identify cases where System Requirement specification is missing some SRs, based on Stakeholder Specification, and create them.	By Project			✓
	UC10	**Project-to-Project Requirements Consistency.** Ensure project requirements are not in conflict with those of other projects within the same solution.	Between Project		✓	

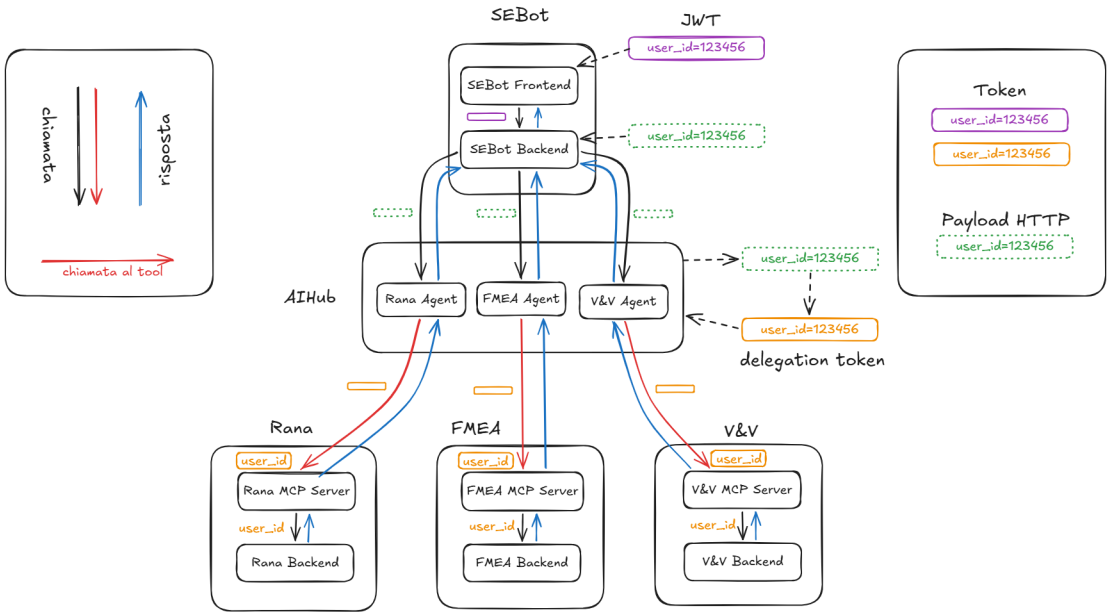


Figura 3.1: Rappresentazione grafica della chiamata ai tool per mezzo del delegation token

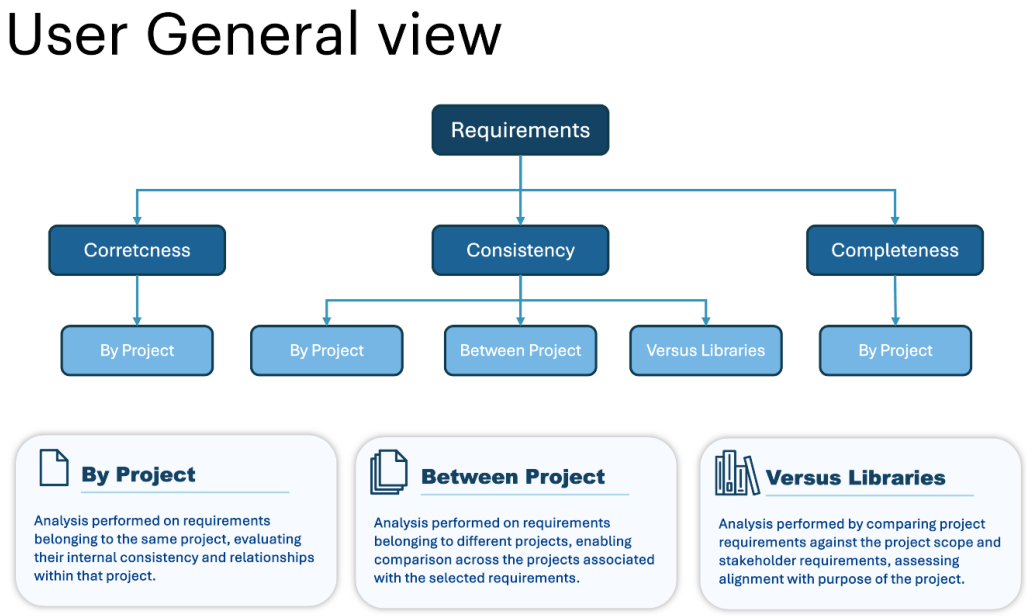


Figura 3.2: Analisi che Rana può eseguire sui requisiti.

Di seguito vengono riportati le strategie e i metodi applicati per affrontare l'implementazione di ogni Use Case.

3.2.1 Input: Requisiti, CSN, TSG

In prima istanza, il cliente ha fornito il formato dei file di input che il tool si aspetta.

Il cuore del tool è l'analisi di requisiti, pertanto il tool deve fornire la possibilità di caricare un file excel (i cui campi sono riportati nella Sottosezione 3.2.2). I requisiti, per come li hanno definiti i System Engineers, possono essere di quattro tipi:

- **Stakeholder Requirements (SHR)**: si tratta di requisiti che esprimono cosa gli stakeholder (che possono essere operatori della manutenzione, proprietari, ...) vorrebbero che si realizzasse riguardo a un certo progetto e di cosa hanno bisogno. Sono requisiti scritti in linguaggio semplice e non tecnico-ingegneristico. Esempio:

  > The washing machine shall be capable of washing a maximum load of 10 kg of dry laundry with an efficiency $\geq$ 90 %.

- **System Requirements (SR)**: sono la traduzione ingegneristica degli SHR, ossia definiscono cosa il sistema deve fare. Ad esempio:

  > A standard "Cotton 60 °C" washing cycle at nominal 10 kg load shall not exceed 50 litres of total water consumption, measured per IEC 60456 test conditions.

- **Sub-system Requirements (SSR)**: sono la scomposizione dei SR in requisiti riguardanti sottosistemi individuali (meccanico, elettrico, di controllo, software, ...). Ogni sottosistema ha il suo budget totale. Ad esempio:

  > The water-intake valve assembly shall regulate inflow to $\leq$ 9 L per fill phase across 0.5–8 bar supply pressure

- **Interface Requirements (IR)**: si tratta di requisiti che permettono ai sottosistemi di lavorare insieme, definendo parametri di comunicazione tra gli stessi. Esempio:

  > At the interface between the drum shaft and the motor drive pulley, radial run-out shall be $\leq$ 0.10 mm at full 10 kg load, so that vibration transferred to the chassis stays within the acoustic limit ($\leq$ 70 dB(A) @ 1 m during spin).

Ognuno dei requisiti può avere un **parent** a cui fa riferimento: mentre il requisito parent è generico, il requisito child approfondisce alcuni aspetti definiti dal suo parent. I requisiti non possono avere come parent qualsiasi tipo di requisito: la "parentela" è definita in Figura 3.3.

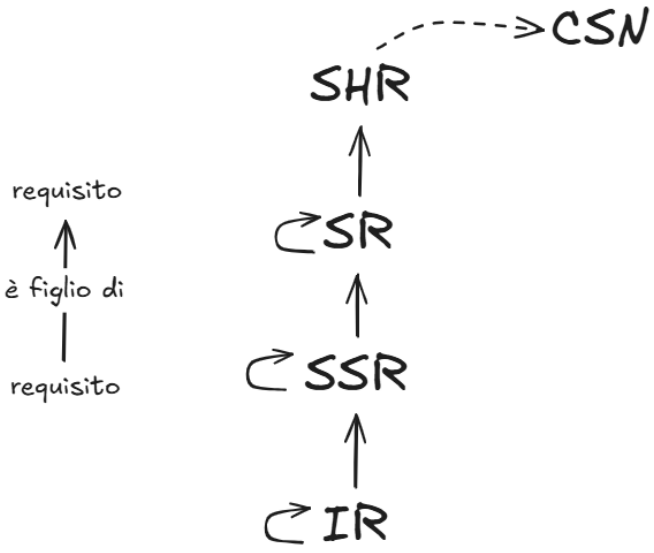


Figura 3.3: Parentela tra requisiti.

Tra i documenti forniti dal cliente c'è anche quello dei **Customer Solution Needs** (**CSN**, Figura 3.3), ossia raccolte di bisogni e necessità grezzi da parte dei suoi clienti sui prodotti di cui si occupa. Questi bisogni non sono altro che un livello ulteriore di astrazione degli SHR, i quali ora rappresentano, pertanto, un'analisi strutturata dei CSN. Esempio di CSN:

> Deliver a step-change in running cost and remote-control convenience to differentiate in the mid-range segment.

Infine, per ogni progetto, viene fornita una presentazione PowerPoint riguardante la **Technical Solution Governance (TSG)**, ossia una presentazione del processo strutturato della creazione di un nuovo prodotto. Vengono illustrate informazioni come la valutazione di maturità, rischi e gap (Sezione 3.3), il progresso di V&V (Sezione 3.4), e le decisioni richieste. Questa presentazione risulta molto utile nell'analisi dei requisiti, perché è presente una slide (spesso chiamata "Solution Configurations"), che ci fornisce una panoramica di quali tematiche del progetto devono essere implementate o eseguite (*in-scope*) e di quali invece il progetto non ne richiede l'attuazione (*out-of-scope*). Il contenuto si presenta come mostrato in Tabella 3.2.

Tabella 3.2: Esempio di slide *Solution configurations*

Category	IN SCOPE	OUT OF SCOPE
Performance	• **Nominal Load Capacity:** wash up to 10 kg dry laundry at efficiency ≥90% • **Max Cycle Duration:** "Cotton 60°C" ≤120 min; cotton program ≤100 min	• Loads >10 kg; efficiency <90% • Cycles exceeding stated durations
Safety	• **Overvoltage Protection:** circuit prevents component damage above 250 V • **Door Lock:** locked during cycle; unlocks only below 40°C	• Sustained operation beyond protection design • Manual door release mid-cycle
Usability	• **Intuitive UI:** start a standard cycle in ≤3 interactions • **Cycle End Indicator:** visual *and* acoustic signal at cycle end	• Flows requiring >3 interactions • Silent / indicator-less end states
Environmental	• **Energy Efficiency:** at least class A++ per applicable EU regulation • **Water Consumption:** minimised usage; standard cycle ≤50 L	• Efficiency classes below A++ • Cycles exceeding 50 L
Reliability	• **Drum Durability:** no structural failure for ≥5000 wash cycles	• Operation beyond 5000 cycles (not warranted)
Maintainability	• **Maintenance Access:** filter, pump, gasket accessible without housing removal	• Repairs needing external housing disassembly
Functional	• **Automatic Load Detection:** auto-detect load; adjust water and detergent	• Manual water / detergent dosing
Diagnostic	• **Fault Indication:** unique error code per fault type, shown to user	• Uncoded / undisplayed faults
Connectivity	• **Remote Control:** start, monitor, stop cycles via app (Wi-Fi IEEE 802.11n)	• Non-802.11n or wired-only remote control
Acoustic	• **Operational Noise:** spin at full load ≤70 dB(A) at 1 m	• Noise above 70 dB(A) at 1 m

Questa tabella sarà molto utile al tool per stabilire se un certo gruppo di requisiti copre interamente l'obiettivo del progetto (analisi di completezza, Sottosezione 3.2.6).

3.2.2 Data Model e caricamento del file di input

Prima di procedere con il dettaglio dell'implementazione delle varie logiche, è necessario fornire una panoramica della struttura del database di Rana. Lo schema Entità-Relazione su cui si basano le principali tabelle è riportato nella Figura 3.4.

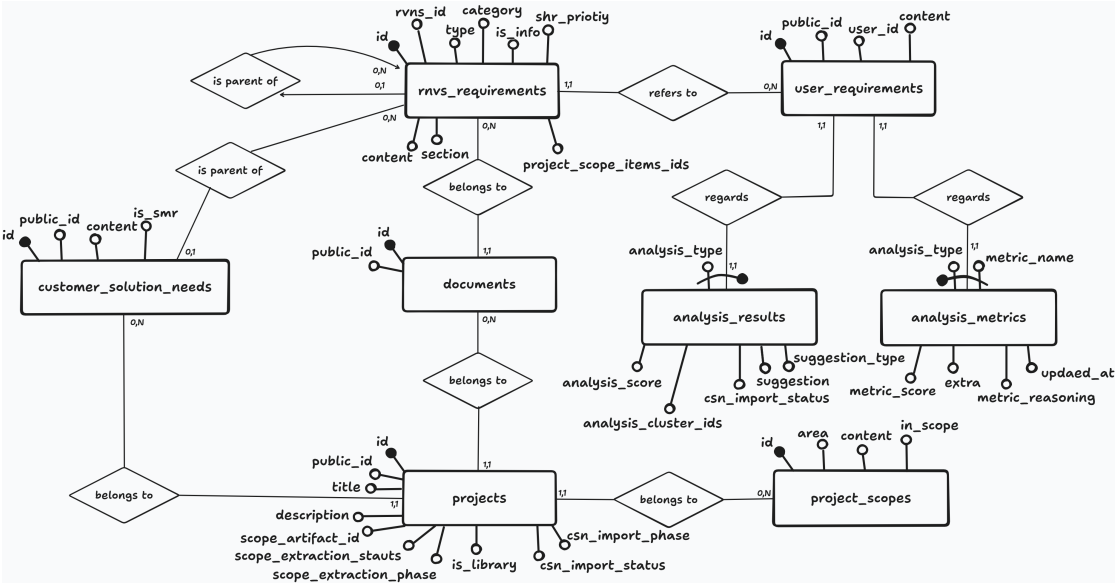


Figura 3.4: Schema ER di Rana.

Nonostante il diagramma ER sia sufficiente a definire le tabelle e le loro relazioni, è opportuno soffermarsi su alcuni aspetti.

Per quanto riguarda la tabella `rnvs_requirements`, il campo `shr_priority` risulta fondamentale nell'esito dell'analisi di (Sottosezione 3.2.5). Due SHR, infatti, possono avere testo molto simile, ma differiscono per questo campo, il cui valore può essere solo *Must* o *Want*: il primo indica che il requisito deve raggiungere un determinato target, il secondo indica un target che si desidera raggiungere, e pertanto più stringente del primo. Un esempio è riportato in Tabella 3.3.

Il caricamento dei requisiti nel tool avviene per mezzo dell'apposito pulsante nell'interfaccia utente, il quale richiama un excel parser costruito ad-hoc nel backend di Rana. Ogni file fa parte di un document, il quale a sua volta rientra in un progetto (indicato dall'apposita colonna `project`).

SHR Priority	SHR text
Must	The washing machine shall not exceed *50 liters* of water consumption during a standard washing cycle.
Want	The washing machine should not exceed *40 liters* of water consumption during a standard washing cycle.

Tabella 3.3: Esempio della differenza tra Must e Want, per uno SHR.

Una volta importati i requisiti, un utente può voler modificare il contenuto di alcuni requisiti (oppure può voler accettare le modifiche suggerite a valle di un'analisi, Sottosezione 3.2.4, Sottosezione 3.2.6); tuttavia, ogni utente ha la necessita di visualizzare esclusivamente le proprie modifiche e non quelle di altri utenti. Questa necessità giustifica l'esistenza della tabella `user_requirements` (Figura 3.4), che fa riferimento a un requisito presente in `rnvs_requirements` ma ne altera il contenuto, limitandolo al workspace dell'utente in questione.

Analogamente ai requisiti, anche il file excel dei CSN può essere caricato, questa volta però all'interno di un progetto (e non nel workspace generico, come per i requisiti), in quanto i CSN afferiscono unicamente a un progetto. A differenza della logica di ricerca della "parentela" dei requisiti, non vi è una colonna che espliciti l'identificativo del Customer Solution Need parent degli SHR: è necessario, pertanto, associare ogni SHR al CSN più vicino semanticamente. Al fine di effettuare il linking, la logica di Rana si occupa di effettuare l'ingestion (Sottosezione 2.3.3) di tutti gli SHR già presenti nel workspace e di tutti i CSN appena caricati nel database, in modo da effettuare un clustering semantico (Sottosezione 2.3.4) utilizzando l'algoritmo offerto dalla classe `NearestCentroid` in Knowledge; non si tratta di un vero e proprio algoritmo di clustering, bensì di una semplice associazione dei punti (SHR) nello spazio vettoriale al centroide più vicino (CSN), come illustrato in Figura 3.5.

Anche l'importazione del TSG è strettamente legata alla presenza di un progetto. La logica di estrazione dei dati dalla tabella nella slide "Solution Configurations", più complessa perché richiede l'intervento dell'AIHub, è molto legata all'analisi di completezza e pertanto approfondita nella Sottosezione 3.2.6.

Successivamente, nell'ultima fase del progetto, è stata definita una logica per importare i requisiti direttamente da RV&S (Requirements Validation & Source), il software utilizzato dal cliente per eseguire altre operazioni sui requisiti.

3.2.3 Analisi: metriche, score e reasoning

Una volta importati tutti i file necessari, l'utente può eseguire diverse analisi messe a disposizione da Rana (Figura 3.6).

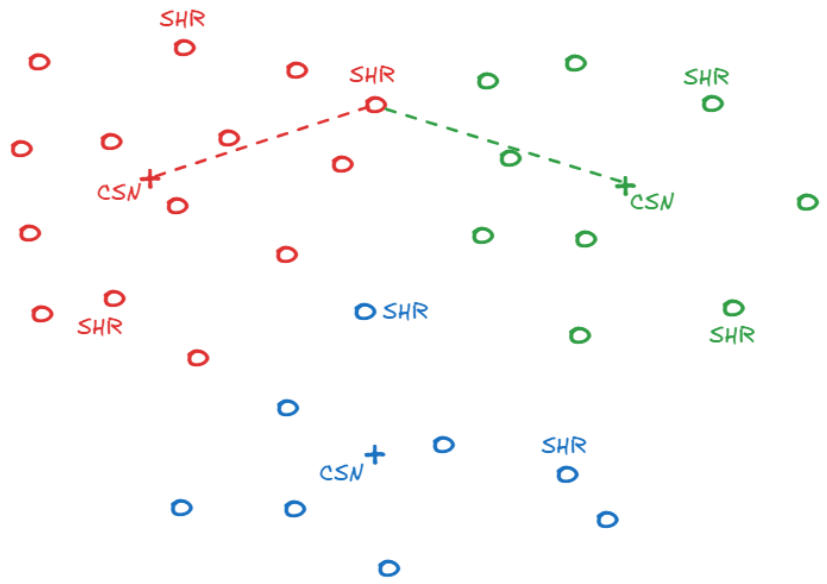


Figura 3.5: Linking degli SHR con i CSN ad opera del Knowledge.

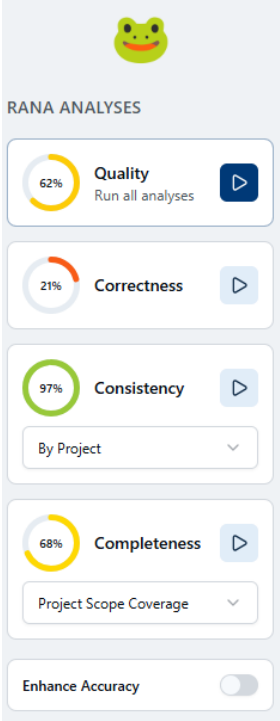


Figura 3.6: Interfaccia del SEBot che permette il lancio delle analisi.

Per ogni analisi, in base alla tipologia di requisito, viene assegnato uno score per una metrica predefinita. Ad ogni richiesta di analisi, Rana prepara i dati necessari e li invia all'AIHub per mezzo dell'`aihub_connect` (Sezione 2.1) invocando il workflow necessario. Ognuno dei tre workflow delle analisi è composto da un singolo nodo che fa una chiamata all'LLM, il quale risponde con gli **score** e i **reasoning** relativi alle singole metriche per ognuno dei requisiti in input. In base alle indicazioni fornite dal cliente, gli score che l'LLM può fornire (in base agli input forniti, alle informazioni di contesto, alle indicazioni del prompt, e al proprio "giudizio") sono:

- 0.0 (0%): il requisito non soddisfa affatto la metrica valutata;
- 0.3 (30%): il requisito esprime in maniera grossolana il contenuto valutato dalla metrica, ma resta non accettabile;
- 0.7 (70%): il requisito per quella metrica è accettabile, ma può essere migliorato;
- 1.0 (100%): il requisito soddisfa appieno la metrica in questione e non necessita di ulteriori miglioramenti.

Lo score complessivo di un requisito, per un certo tipo di analisi, è calcolato tramite **somma pesata** degli score di tutte le metriche relative all'analisi in questione; il sistema di pesi permette di non penalizzare (o valorizzare) troppo lo score finale a causa di metriche di importanza minore e, viceversa, è utile per far risaltare le metriche considerate di maggior importanza. I pesi sono stati definiti dal cliente nell'apposito file `analyses_weights.json`. Lo score ottenuto dalla media pesata viene arrotondato al valore più vicino tra quelli definiti (0.0, 0.3, 0.7, 1.0). Infine, al lato della schermata, Rana mostra il resoconto delle analisi, ossia la media degli score di tutti i requisiti analizzati, divisi per analisi (Figura 3.6).

Le metriche che l'LLM deve valutare sono stabilite dai prompt delle analisi, progettati in modo accurato in collaborazione con il lato funzionale del team composto da ingegneri meccanici, e presenti sempre nell'AIHub (analogamente a quanto effettuato per FMEA, Sottosezione 3.3.3, e V&V, Sottosezione 3.4.3).

Siccome il GenAIX supporta le esecuzioni in parallelo (Capitolo 2), le analisi vengono pertanto eseguite in parallelo per ottimizzare i tempi di esecuzione e migliorare la user experience; pur non trattandosi di una vera e propria esecuzione multi-thread, la logica sfrutta i tempi morti di esecuzione, in cui il processo è costretto ad attendere una risposta I/O-bound (ad esempio una chiamata a un LLM remoto) che possono richiedere da 5 secondi a 3 minuti (in base al workflow) per lanciare analisi su altri requisiti in coda. La concorrenza asincrona è stata implementata per mezzo di due **semafori** (Figura 3.7):

- `user_sem`, che pone un limite massimo al numero di analisi in parallelo che possono essere eseguire da uno stesso utente. Ogni utente ha il suo `user_sem`.

- `global_sem`, che pone un limite massimo al numero di analisi in parallelo che possono essere eseguite da tutti gli utenti nello stesso momento. In quanto globale, questo semaforo è condiviso tra tutti gli utenti

L'utilizzo dei semafori è indispensabile per evitare di sovraccaricare la risorsa cloud dell'LLM Provider e raggiungere il rate limit.

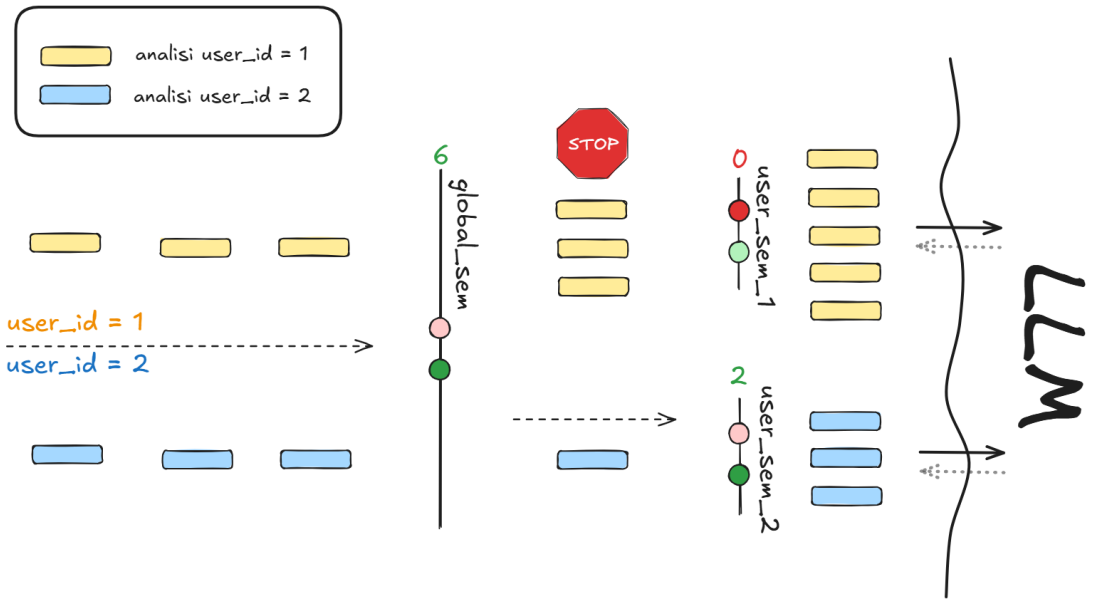


Figura 3.7: Parallelizzazione delle analisi con l'utilizzo dei semafori.

Per eseguire le coroutines in parallelo, è stato necessario ricorrere al modulo Python *asyncio*, che appunto sfrutta l'intattività del processo durante l'attesa della risposta di un workflow (`await`) per chiamarne un altro. Ciò che viene parallelizzata, pertanto, è la chiamata al workflow: le funzioni `_process_cluster`, `_run_correctness_wf` e `_run_proper_check_wf` sono funzioni wrapper che mascherano l'utilizzo dei semafori `global_sem` e `user_sem` e la chiamata all'LLM.

Per semplicità, al fine di orientare l'esposizione alla struttura della logica senza approfondire dettagli ripetitivi e fuori contesto, le analisi riportate in seguito (Sottosezione 3.2.4, Sottosezione 3.2.5, Sottosezione 3.2.6) verranno esaminate come se i workflow fossero eseguiti in maniera sequenziale.

A seguito di numerosi tentativi, i modelli che infine sono stati scelti per i workflow sono:

- **gpt-5.4-mini** con reasoning effort impostato a **low** per i workflow di `check_consistency`, `correctness_proper_check`, `link_scope` (Project Scope linking)
- **gpt-5-mini** con reasoning effort impostato a **low** per il workflow di `check_correctness`

A fronte di numerosi cambiamenti dei diversi prompt in fase di sviluppo, il team ha predisposto un sistema di **testing** per assicurarsi che nuove modifiche non andassero ad impattare risultati precedentemente corretti. I singoli test cases riguardano l'esecuzione singola di un workflow, il cui risultato è validato in base alle opzioni definite nel file di configurazione `.yaml` per il singolo test:

Esempio di configurazione di un test case

```
- metric_name: Traceability
  score: [0, 0.3]
  reasoning:
    assert_type: none
    expected: ""
- metric_name: Clear - Unambiguous
  score: [0.7, 1.0]
  reasoning:
    assert_type: none
    expected: ""
- metric_name: Verifiable/Validatable
  score: [0.7, 1.0]
  reasoning:
    assert_type: none
    expected: ""
```

Per ogni metrica (`metric_name`) vengono definiti gli score ammessi (`score`) per far passare il test, mentre la verifica del reasoning può avvenire in 3 modalità:

- `none`, nessuna verifica del reasoning.
- `substring`, la stringa presente nel campo `expected` deve comparire nel testo del reasoning prodotto.
- `llm`, ossia la verifica è delegata a un LLM, le cui istruzioni sul contenuto atteso del reasoning sono contenute in `expected` (è usato solo per contenuti attesi più complessi).

Qualora nel database siano presenti uno o più suggerimenti per uno stesso requisito, nel caso in cui siano state lanciate le anlisi di correttezza e consistenza,

il workflow `merge_improved` si occupa di unire i due suggerimenti. Il system prompt è disponibile in Appendice A.

3.2.4 Correctness

L'analisi di correttezza è la più semplice delle analisi, il cui obiettivo è analizzare la correttezza dei singoli requisiti in base alle metriche definite. Per questa analisi sono necessari: il requisito da analizzare, il suo parent (se presente), la lista degli acronimi e il vocabolario. Questi ultimi due input sono (quasi) sempre presenti in ogni chiamata all'LLM (non solo per la correttezza) su richiesta di Rana, in quanto permettono di comprendere meglio il contenuto di ogni requisito, che può riguardare acronimi o termini specifici utilizzati esclusivamente dall'azienda cliente (pertanto, fornisce maggiore contesto sul suo linguaggio interno).

La funzione `check_correctness` è quella nella business logic che si occupa di preparare i dati necessari da inviare ai workflow e che attende i risultati da salvare nel database. Le metriche valutate (e, in alcuni casi, il flusso di esecuzione) variano in base al tipo di requisito selezionato:

- SHR (UC1, UC2): prima di procedere con la vera e propria analisi di correttezza è necessario verificare che il requisito Stakeholder selezionato non sia stato erroneamente categorizzato come tale, quando in realtà si tratta di un requisito System. La funzione `_check_correctness_proper_check_wf` chiama il `correctness_proper_check` workflow nell'AIHub, che si occupa di controllare se il requisito contiene terminologie tali da classificarlo come SR. Il prompt infatti istruisce:

  Correctness Proper Check Prompt

  "You are an expert systems engineer trained in the INCOSE Systems Engineering Handbook and IEEE 29148:2018 requirements engineering standard.
  Your task is to determine whether the provided requirement is a proper Stakeholder Requirement (SHR).
  Criteria
  - If the Stakeholder Requirement contains a comparison with a value with unit of measurement (e.g. "greater than 5 mL", "less than 10 inches", "equal to 3 degrees") or a percentage (e.g. "greater than 80%", "less than 50%") or a currency (e.g. "greater than $100", "less than €50"), then it is a SHR. Otherwise, if it contains a comparison with another product or standard, it should be a System Requirement (SR).
  - If the Stakeholder Requirement compares the performance or compliancy with a standard, then it is not a SHR, but rather a SR.

- If the Stakeholder Requirement has a project process or document deliverable as the grammatical subject of the "shall" clause, then it is NOT a proper SHR.
- If the "shall" subject is the product/system/component/kit, then references to project artifacts, CIs, issues, tickets, processes, or document deliverables in the object or complement of the sentence are not rejection reasons.
- If the Stakeholder Requirement compares a business/performance metric to an index <number>, it is a proper SHR.
- Custom Criteria: If the requirement defines the minimum number of different suppliers per base material, it is a proper SHR only when that number is different from 3. If the number is 3, it is not a proper SHR. In that case mention that it is already defined in the System Requirement Library." [...]

Lo score assegnato per la metrica Proper è 0.0 se il requisito è erroneamente classificato come SHR, 1.0 altrimenti.

Nel caso in cui si tratti di un proper SHR, allora le metriche che vengono valutate sono:

SHR Correctness Prompt

"[...]
Quality Rules
1. **Necessary** - Clearly needed to satisfy a specific stakeholder need and supported by a clear justification of why it is required.
2. **Proper** - Stated at the correct stakeholder level, expressing what is needed rather than how to implement it.
3. **Clear - Unambiguous** - Uses precise, objective, and measurable terms allowing only one interpretation.
4. **Achievable** - Technically and practically feasible within current or near-term technology, resources, schedule, and budget.
5. **Verifiable/Validatable** - Objectively measurable, testable or referencing other products/specification.
6. **Traceability** - Clearly derived from and consistent with the parent Customer Solution Need, if one exists. If no parent, the requirement should still be well-justified and necessary on its own.
[...]"

- SR (UC1, UC3, UC4): per quanto riguarda i SR, invece, sono state richieste le valutazioni delle seguenti metriche:

SR Correctness Prompt

"[...]
Quality Rules
1. **Necessary** - Represents something reasonably needed and valuable for the sub-system purpose.
2. **Appropriate** - Assess whether the requirement is appropriate for a system-level specification and consistent with its declared category.
3. **Unambiguous** - Uses precise, specific, quantitative terms where possible.
4. **Complete** - Provides sufficient information to understand and implement. Stands alone as a full statement.
5. **Singular** - Expresses one distinct behavior, performance, or constraint.
6. **Feasible** - Technically and operationally achievable by the sub-system.
7. **Verifiable/Validatable** - Objectively measurable or testable at sub-system level.
8. **Correct** - Logically consistent with the sub-system role and parent requirement.
9. **Conforming** - Follows approved syntax "[System element] shall [action verb] [object] [qualifier]."
10. **Implementation-Free** - Avoids prescribing internal design, components, technologies, algorithms, or architectures unless explicitly required.
11. **Traceability** - Clearly derives from, refines, or allocates part of the parent requirement, if one exists.
[...]"

Degna di nota è la metrica *Singular*: qualora lo score assegnato non fosse 1.0, infatti, l'LLM si occuperebbe anche di scomporre il requisito analizzato in un numero sufficiente di requisiti atomici, suggerendo all'utente questi nuovi requisiti anziché quello di partenza. L'utente può accettare il suggerimento e integrare definitivamente questi nuovi requisiti per sottoporli ad altre analisi.

- SSR (UC1, UC3, UC4): per i SSR le metriche analizzate solo le stesse dei SR (con piccole opportune variazioni nel prompt).

- IR (UC5): per gli IR l'unica metrica valutata è `IR Clarity`:

IR Correctness Prompt

"[...]
Metrics
- **IR Clarity**:
  - **Description**: Evaluates whether an interface requirement is referring to the same system element as the title in the architecture (info) context. It either is or isn't referring to the same system element as the architecture (info) context.(Eg. Product, Equipment, Machine...)
[...]"

Al fine di confrontare l'IR analizzato con il requisito più vicino "Architecture (Info)" è stata implementata una logica per estrarre dal database i requisiti con `is_info=TRUE` (Figura 3.4), e con `section` "più vicina" a quella del requisito analizzato (Longest Prefix Match).

3.2.5 Consistency

L'analisi di consistenza è stata più complessa a livello implementativo rispetto alla correttezza. Siccome valutare la consistenza tra più requisiti richiede il confronto tra gli stessi, dapprima il team tecnico si è posto il problema di passare all'LLM un numero ragionevole di requisiti da confrontare prima che il modello allucinasse; infatti, è impensabile passare diverse decine di requisiti con diversi campi e con l'aspettativa che l'LLM sia in grado di individuare le inconsistenze senza trascurarne nessuna e, soprattutto, senza inventarne di nuove inesistenti.

Pertanto, a valle di profonde riflessioni con il team funzionale e il cliente, si è concordato che le inconsistenze possono presentarsi solo tra requisiti che affrontano tematiche simili. In conclusione, il team tecnico ha deciso di servirsi del Knowledge, implementando l'ingestion dei requisiti (che, di fatto, sono record di PostgreSQL, Sottosezione 2.3.3), per poi procedere a **clusterizzare i requisiti semanticamente vicini** (Sottosezione 2.3.4). Avendo come obiettivo del clustering quello di separare i requisiti in gruppi riguardanti stessi problemi, configurazioni o sistemi, e non avendo bisogno di stabilire a priori un numero di cluster predefinito (per cui un algoritmo come K-Means non sarebbe stato adatto al contesto), è stato scelto **HDBSCAN**[6] come algoritmo di clustering. A differenza del suo predecessore, DBSCAN, non è necessario definire il parametro ϵ per stabilire il livello di densità stimato dei cluster, ma valuta diversi livelli di densità e costruisce i cluster scegliendo la densità in base alla loro stabilità, riducendo così il tuning manuale.

L'utente può selezionare uno dei seguenti tipi di analisi di consistency:

- **By-project** (UC1, UC6): la consistenza viene valutata tra requisiti appartenenti a uno stesso progetto.

- **Between-projects** (UC10): la consistenza è valutata tra requisiti di due progetti. Ogni requisito, infatti, deve essere consistente rispetto ai requisiti definiti nell'altro progetto in esame.
- **Versus-libraries** (UC7): il cliente mette a disposizione una serie di requisiti già analizzati e validati dai System Engineer, e pertanto costituiscono fonte di verità assoluta (da cui prendere spunto anche per la generazione di nuovi requisiti, Sottosezione 3.2.6). Le librerie vengono caricate nel tool come normali requisiti, il cui progetto è marcato come `is_library` (Figura 3.4). I requisiti selezionati devono rispettare criteri di consistenza rispetto alle librerie, tenendo conto che in caso di inconsistenza, queste ultime sono necessariamente corrette.

Le analisi di consistenza interna (by-project) e tra progetti (between-projects) devono poter essere eseguite anche tra le stesse librerie (UC6).

Per ognuna delle tre tipologie di , la logica di esecuzione dal punto di vista di Rana è la seguente:

1. Raccolta dei requisiti selezionati.
2. Divisione dei requisiti per tipo (primo raggruppamento).
3. Divisione dei requisiti in gruppi (cluster) di requisiti simili (chiamata al Knowledge per ingestion e clustering, secondo raggruppamento).
4. Chiamata del workflow in base al tipo di requisiti e al tipo di analisi selezionata per il gruppo in esame (chiamata all'AIHub).
5. Salvataggio dell'analisi fornita dall'LLM.

Le metriche valutate variano in base al tipo di consistenza e, come per la correttezza, in base al tipo di requisito; esse sono riportate in Appendice A.

3.2.6 Completeness

L'analisi di completezza è eseguita su gruppi di requisiti appartenenti allo stesso progetto e solo su SR (UC9); le modalità di completezza sono: **Project Scope Coverage** e **SSR FMEA Cascading**.

Project Scope Coverage

Una parte non deterministica dell'analisi di completezza viene eseguita a monte del lancio dell'analisi, con il caricamento della presentazione TSG (Sottosezione 3.2.1, Sottosezione 3.2.2); infatti, quando l'utente carica il PowerPoint, la logica effettua le seguenti operazioni:

1. Carica il PPTX in Artifcats (Sezione 2.2).

2. Cerca la slide con titolo “Solution configurations”

3. Se non la trova, ritorna errore al frontend. Altrimenti, converte la slide in un’immagine png.

4. Tutti i SR del progetto in questione vengono suddivisi in batch da 3 requisiti ciascuno.

5. Per ogni batch, esegue una chiamata HTTP POST all’AIHub all’endpoint `/vision/describe` (tramite il `knowledge_client`) con il seguente prompt:

   TSG Table Extraction Prompt

   Extract the ’Solution configurations’ table from this image. The table has one row per area/category (e.g. PRODUCT, PACKAGE, EQUIPMENT, etc.). Each row has exactly three fields: the area name, the in-scope items, and the out-of-scope items. Return ONLY a JSON array of objects with EXACTLY these three keys for every element:
   Area – the area/category name (e.g. PRODUCT, PACKAGE, EQUIPMENT);
   In scope – a single string with all in-scope items for that area, separated by semicolons if multiple;
   Out of scope – a single string with all out-of-scope items for that area, separated by semicolons if multiple, or an empty string if none.

   Do NOT rename or add keys. Do NOT use literal placeholders such as ... in any field. If a cell contains ... or another placeholder, try to infer the intended content from nearby cells or the header; if you cannot reliably infer it, set that field to an empty string. Trim whitespace from all values. No markdown, no explanation, just the JSON array.

6. Il testo ritornato dal modello di computer vision dell’AIHub viene convertito in JSON.

7. le informazioni estratte dalla tabella della slide vengono salvate nella tabella `project_scopes` del database (Figura 3.4).

Un esempio del contenuto della slide è raffigurato in Tabella 3.2.

A valle del processo di **Project Scope linking**, ogni requisito di tipo System nella tabella `rvns_requirements` avrà nel campo `linked_scope_items_ids` una lista contenente gli identificativi degli items della tabella `project_scopes`.

Selezionando i SR, l'unica metrica valutata è la Sufficient Coverage, che misura il livello di copertura del project scope rispetto ai SR; non ha bisogno dell'LLM, infatti è calcolata in maniera deterministica.

Al lancio dell'analisi di completezza, la funzione `check_completeness` raggruppa i SR selezionati e, se il TSG è stato precedentemente caricato, calcola la frazione di project scope items in-scope coperti da almeno un requisito, ossia quanto è stato coperto il project scope. Questa frazione viene poi arrotondata allo score (per l'unica metrica di Sufficient Coverage) più vicino tra 0.0, 0.3 e 0.7; se tutti gli item in scope del progetto sono stati collegati ad almeno un SR, allora lo score è di 1.0. Se, invece, almeno un requisito è collegato ad almeno un item out-of-scope, allora lo score è diminuito ($1.0 \rightarrow 0.7$, $0.7 \rightarrow 0.3$, $0.3 \rightarrow 0.0$).

SSR FMEA Cascading

La seconda delle due modalità di lancio della completezza permette di valutare la completezza dei requisiti selezionati rispetto a **documenti FMEA** (Sezione 3.3): infatti, per ognuno dei SR selezionati, Rana chiede all'FMEA di recuperare quegli FMEA items riguardanti i requisiti in analisi (tramite la colonna `jeopardizes` della tabella `fmea_items`, che contiene gli identificativi dei requisiti collegati, Sottosezione 3.3.2).

Il campo che interessa per l'analisi di completezza è **FMEA Range** (Tabella 3.4), ossia l'**Action Priority** del guasto. Secondo le linee guida fornite dal cliente, ogni requisito di tipo System legato a un FMEA item con Range High deve avere dei requisiti Subsystem figli associati. Senza di essi, a fronte di un rischio alto di guasto, sarebbero difficili operazioni come:

- allocare le mitigazioni;
- definire test di verifica;
- fare il failure trace verso le cause tecniche;
- dimostrare la copertura del rischio.

Pertanto, se per FMEA items con rischio minore l'assenza di SSR figli può essere trascurabile, con un rischio elevato diventa un condizione necessaria.

Dunque, lo score assegnato per la metrica FMEA SSR Cascading è:

- 0.0 se il requisito è associato ad almeno un FMEA item con FMEA Range High e non ha SSR figli.
- 0.7 se il requisito non è legato a nessun FMEA item presente nel tool FMEA.

FMEA Range	Significato
High	Rischio elevato, richiede azioni prioritizzate.
Medium	Rischio moderato, da monitorare o mitigare.
Low	Rischio accettabile o comunque meno critico.
Out of Range	Item informativo oppure valori S/O/D non validi per il calcolo.

Tabella 3.4: Interpretazione del campo *FMEA Range*.

- 1.0 se il requisito è associato ad almeno un FMEA item con FMEA Range High e ha almeno un SSR figlio, oppure se è associato solo a FMEA items con FMEA Range minori di High.

Per i SR a cui è stato assegnato lo score 0.0, Rana prova subito a mitigare al problema proponendo all'utente **nuovi requisiti SSR** da associare al requisito che non ha superato l'analisi.

L'operazione effettuata da Rana, per ogni requisito in questione, consiste in una ricerca di requisiti System in un progetto **golden standard** (progetto di riferimento considerato corretto, fornito dal cliente) della stessa famiglia e suggerire all'utente i suoi SSR figli come nuovi figli del requisito che non ha passato l'analisi di FMEA SSR Cascading. La ricerca del requisito SR più simile è effettuata per mezzo del clustering, usando l'algoritmo `NearestCentroid` (come per il linking dei CSN). L'utente, analogamente a quanto accade con la metrica *Singular* dell'analisi di correttezza (Sottosezione 3.2.4), può accettare il suggerimento e adattare il contenuto del SSR accettato nel proprio workspace in base alle necessità. Rilanciando l'analisi di completezza con questa modalità sugli stessi requisiti, a valle dell'aggiunta dei nuovi SSR figli, lo score assegnato sarà pertanto pari a 1.0.

3.2.7 Voting

Nel corso dell'implementazione di Rana, a valle dei numerosi confronti con il cliente, è emersa la scarsa ripetibilità dei risultati sia nella generazione delle analisi, che in altre fasi aleatorie come il linking del Project Scope.

Nonostante il perfezionamento dei prompt, le procedure che richiedevano l'intervento dell'LLM avevano una ripetibilità compresa tra il 60% e l'80% delle iterazioni, inaccettabile per questo tipo di operazioni. Per via degli aggiornamenti degli ultimi modelli della famiglia GPT, non è stato neanche possibile intervenire sulla temperatura per aumentare il grado di precisione dei modelli.

Pertanto, il team di sviluppo ha concordato che la strategia migliore per mitigare il problema è stata l'implementazione di un meccanismo di **voting**, ossia

un sistema per il quale le operazioni che richiedono l'utilizzo di workflow devono essere **ripetute** N **volte**, e il risultato che si verifica con maggior ripetibilità è quello effettivamente considerato finale (Self Consistency, Sottosezione 1.5.2).

L'utente può, dall'interfaccia del SEBot (Figura 3.8), abilitare questo meccanismo e selezionare tra un'accuratezza *High* e *Extra High*, che si traduce nell'impostazione del valore di N a 3 o 5. L'abilitazione di *Enhance Accuracy* si limita soltanto alle analisi; pertanto, un System Engineer, sulla base di diversi fattori come il numero di requisiti da analizzare, la loro complessità e il budget, può decidere se abilitarlo o limitarsi ai risultati "single shot".

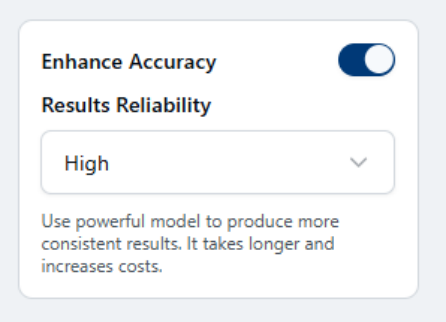


Figura 3.8: Componente dell'interfaccia del SEBot che permette di abilitare il meccanismo di voting.

Per quanto riguarda il linking del Project Scope, invece, il valore di N è fissato a 3, indipendentemente dall'attivazione di *Enhance Accuracy*: trattandosi di un'operazione che ricade sull'analisi di completezza (Sottosezione 3.2.6) e che ha bisogno di particolare precisione, il team ha concordato che fosse corretto non lasciare all'utente la libertà di decidere sul grado di precisione di quest'operazione.

Con l'introduzione del meccanismo di voting, siccome le analisi erano ancora eseguite in maniera sequenziale, è emerso un altro problema di efficienza: infatti, se senza il suo utilizzo il **tempo di esecuzione** delle analisi è lento ma comunque accettabile, attivando l'esecuzione multipla di una stessa analisi più volte, ha notevolmente allungato i tempi di esecuzione delle analisi (da 3 a 5 volte). Prendendo in esame il caso più semplice, l'analisi di correttezza è passata da un tempo di esecuzione da 30 secondi a 3 minuti: pensando di dover analizzare diverse centinaia di requisiti in successione, il tempo di esecuzione non è più un elemento trascurabile. Dal momento in cui è stato implementato questo meccanismo, si è passati alla **concorrenza asincrona** delle analisi, come esposto in precedenza (Sottosezione 3.2.3).

3.3 FMEA

FMEA è una delle due PoC del progetto, il cui scopo è quello di analizzare documenti FMEA (Failure Mode and Effects Analysis) prodotti dai System Engineers per anticipare e analizzare le cause di guasti ai macchinari industriali.

Come per Rana (Tabella 3.1), anche per FMEA il cliente ha definito degli Use Cases, riportati in Tabella 3.5. Trattandosi di una PoC, al fine di valutare l'efficacia di un'implementazione completa del tool, gli Use Cases delineati sono più generici rispetto a quelli di Rana e, pertanto, ognuno di essi non rientra in un caso implementativo a sé ma vengono considerati nella funzionalità d'insieme del tool (le analisi in particolare, in realtà, rientrano tutte nello UC3).

Tabella 3.5: FMEA - Use Cases Covering

UC	Use Cases Description
UC1	**Merge Sub-System FMEAs for Library Product Life Cycle.** Act as an SSO and consolidate Sub-System FMEAs across projects into a reusable library covering the Product Life Cycle.
UC2	**Propose a Sub-System FMEA for a Project.** Act as an SSO and generate a Sub-System FMEA proposal for a project based on a given set of requirements, including identification and branching of relevant failure modes.
UC3	**FMEA Sanity Check Against Similar Projects.** Perform a consistency and completeness review of an existing FMEA by cross-checking it against FMEAs from similar projects.
UC4	**FMEA Session Assistant.** Provide real-time support during FMEA sessions by assisting participants with analysis, documentation, and identification of potential risks and failure modes.

3.3.1 Input: FMEA document, Grading Definitions, Actions, Mitigations, P-diagram

Come per Rana (Sottosezione 3.2.1), anche per FMEA il cliente ha fornito dei file di input che sono oggetto di validazione da parte del tool o semplicemente elementi di arricchimento del contesto.

Il documento cardine del tool, senza cui il tool stesso non esisterebbe, è il **documento FMEA**, che è prorio quel documento che riporta le analisi dei guasti, scritto da uno o più System Engineer, e risulta essere l'oggetto delle analisi del tool: per ogni **failure mode** sono riportati causa, effetto e i valori principali usati per misurare il rischio (Severity, Occurrence e Detection). Ogni failure mode, in realtà, può avere più effetti e più cause di guasti, pertanto, per ogni riga del documento, parliamo di **FMEA item**.

I vari livelli di Severity, Occurrence e Detection vengono assegnati secondo specifici criteri interni all'azienda cliente: pertanto, il documento di **Grading Definitions** è un ulteriore file di input le cui informazioni sono estratte e salvate nel database di FMEA. I dati, rappresentati in formato tabellare come per il documento FMEA, riportano, per ognuna delle tre metriche e per ogni punteggio da 1 a 10, la descrizione che spiega il significato dell'assegnazione del punteggio corrispondente. Come approfondito nella Sottosezione 3.3.4, è una parte decisiva nell'analisi di correttezza.

Due documenti strettamente legati all'FMEA document sono: **Mitigations**, che indicano i metodi di mitigazione in caso di guasti, e **Actions**, ossia le attività operative necessarie per realizzare le mitigazioni. Entrambe le informazioni sono caricate come file excel in formato tabellare e pertanto risultano facilmente importabili nel database.

L'ultimo documento fornito dal cliente è il **P-diagram** (Parameter Diagram), un excel che descrive il comportamento di una funzione e i fattori che potrebbero portare a malfunzionamenti. I documenti P-diagram caricati dall'utente sono composti, di fatto, da una serie di caselle di testo, ognuna contenente la categoria e un elenco puntato. Le categorie possono essere, ad esempio: Function, Input Signals, Ideal Response, Noise Factors, Control Factors, Potential Error States. Avendo definito questo formato, il parsing del documento non richiede l'intervento del modello di computer vision per estrarre le informazioni, come per il TSG in Rana (Sottosezione 3.2.6), ma è stato sufficiente applicare delle RegEx nell'XML del documento per identificare correttamente le caselle di testo. Per questa PoC, il P-diagram all'interno del tool non viene utilizzato attivamente al fine di influenzare gli esiti delle analisi, ma viene semplicemente utilizzato dall'agente per il confronto tra P-diagram di diversi progetti; potrà essere, invece, oggetto di maggiore contesto delle analisi nella futura industrializzazione del tool.

3.3.2 Data Model

Come per Rana (Sottosezione 3.2.2), il punto di partenza è stato il data model, punto cruciale da qui deriva l'intero funzionamento del sottomodulo a livello di business logic. La versione del diagramma Entità-Relazione concordato dal team di sviluppo è rappresentato in Figura 3.9.

Il punto centrale è rappresentato dagli FMEA items, ossia una modalità di guasto definita dai System Engineers, con tutte le informazioni associate. Ad ogni FMEA item possiamo associare diverse mitigazioni e azioni da intraprendere per risolvere o compensare al meglio il guasto (informazioni ottenute dai rispettivi file, come definito nella Sottosezione 3.3.1).

Assumendo, anche per questo tool, che il file caricato costituisca un documento, ogni documento è associato a un progetto (indicato nell'apposito file del documento

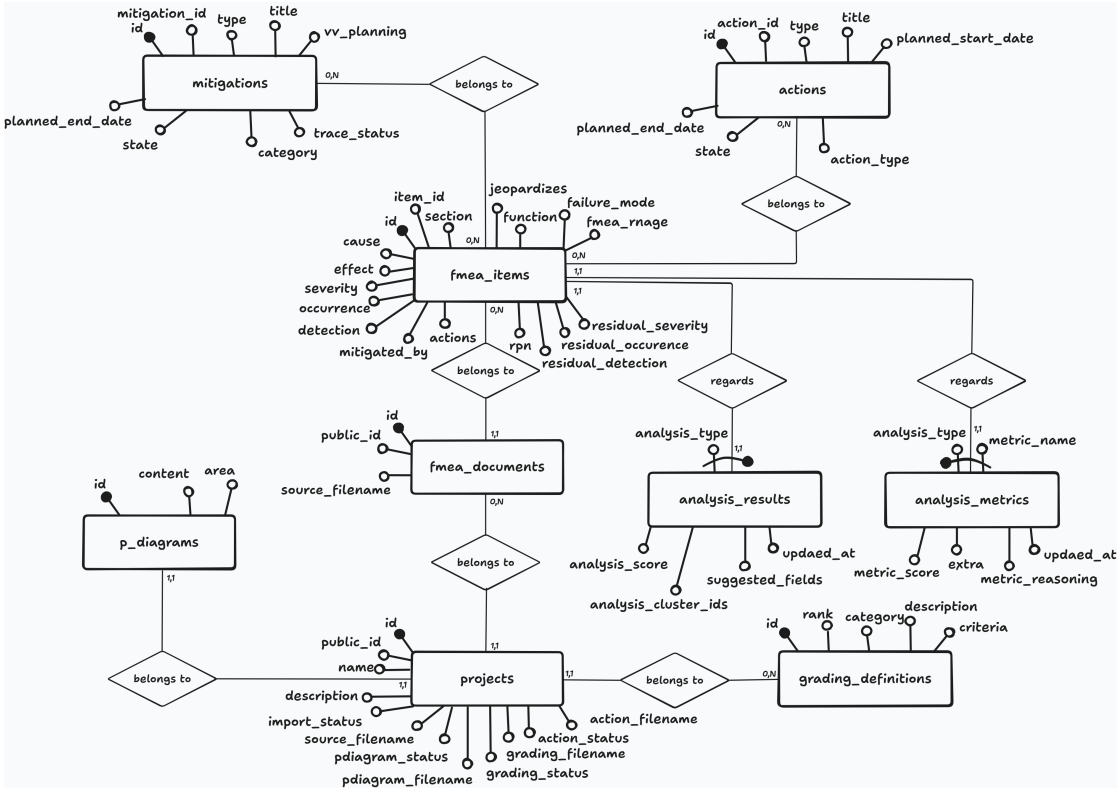


Figura 3.9: Schema ER di FMEA.

FMEA), cui possiamo associare P-diagram e documento di grading definitions.

Infine, anche questo tool dispone delle tabelle `analysis_metrics` e `analysis_results` per salvare i risultati delle analisi eseguite sugli FMEA items.

3.3.3 Analisi: metriche, score e reasoning

Le analisi che l'utente può eseguire nell'FMEA tool si limitano a **correttezza** (Sottosezione 3.3.4) e (Sottosezione 3.3.5) e, come accade in Rana (Sottosezione 3.2.3), il tool assegna uno score per ogni metrica valutata, che è un valore tra 0.0, 0.3, 0.7 e 1.0 (espresso in percentuale nel frontend). Per questa PoC non è prevista un'analisi di completezza.

Anche qui, lo **score** per ogni metrica e il relativo **reasoning** sono ricavati per mezzo dell'LLM, a cui si accede tramite l'`aihub_connect`. Tuttavia, siccome l'obiettivo dello sviluppo di questo tool era verificarne la fattibilità, l'efficienza in termini di tempi di esecuzione non è stata oggetto d'interesse da parte del cliente, e pertanto si è deciso per eseguire le analisi in maniera sequenziale.

A differenza di Rana, però, lo score assegnato a ogni FMEA item, per ogni tipo di analisi, non è calcolato tramite somma pesata, ma è il punteggio minimo assegnato tra le metriche dell'analisi in questione, attribuendo, pertanto, una forte penalizzazione in egual modo a tutte le metriche.

3.3.4 Correctness

Analogamente a quella sui requisiti (Sottosezione 3.2.4), l'analisi di correttezza sugli FMEA items viene eseguita su ogni item preso singolarmente. Per ogni item, il tool fornisce all'LLM l'item da analizzare e le grading definitions per il progetto in questione (necessarie per poter valutare la correttezza dei valori assegnati a Severity, Occurrence e Detection). Le metriche valutate sono le seguenti:

FMEA Correctness Prompt

"[...]
Metrics
- **Unambiguous**:
  - **Description**: Free of subjective terms and multiple plausible readings. It is clear the desired quality, characteristic, or function prescribed by the FMEA item.
  [...]
- **Severity Adherence**:
  - **Description**: The FMEA item residual severity level is consistent with the content of the item and the definitions of severity levels. Do not penalize items with Residual Range Medium or Low.
  [...]
- **Occurrence Adherence**:
  - **Description**: The FMEA item residual occurrence level is consistent with the content of the item and the definitions of occurrence levels. Do not penalize items with Residual Range Medium or Low.
  [...]
- **Detection Adherence**:
  - **Description**: The FMEA item residual detection level is consistent with the content of the item and the definitions of detection levels. Do not penalize items with Residual Range Medium or Low. Design Control check is not supported now, but it will be available in the future, so DON'T penalize the score for this reason.
[...]"

3.3.5 Consistency

L'analisi di consistenza, invece, coinvolge più FMEA items insieme, al fine di trovare inconsistenze e contraddizioni reciproche. Come accade per l'analisi di in Rana (Sottosezione 3.2.5), è necessario raggruppare gli item in base a criteri stabiliti a priori; pertanto, è necessario eseguire l'ingestion nel Knowledge (Sottosezione 2.3.3) al fine di generare embedding che poi saranno raggruppati per mezzo di operazioni di clustering (Sottosezione 2.3.4). In base alla metrica da analizzare, gli embedding vengono generati sui campi:

- `failure_mode`
- `failure_mode` e `cause`
- `failure_mode` e `effect`

L'analisi di consistenza può essere lanciata in due modalità:

- **By-project**: le inconsistenze vengono valutate tra FMEA items facenti parte dello stesso progetto. Le metriche valutate solo:
  - Per items che riguardano stesso guasto con cause simili:

    FMEA Consistency By Project (similar Failure Mode and Cause) Prompt

    "[...]
    ## Metrics
    - **Consistent Occurrence Values**:
      - **Description**: FMEA items within the same project that refer to the same or semantically equivalent failure cause shall use consistent Occurrence ratings. Equivalent causes are expected to reflect the same estimated likelihood of occurrence and therefore should not be assigned significantly different Occurrence values without clear justification.
    [...]"

  - Per items che riguardano stesso guasto con effetti simili:

    FMEA Consistency By Project (similar Failure Mode and Effect) Prompt

    "[...]
    ## Metrics

- **Consistent Severity Values**:
  - **Description**: FMEA items within the same project that refer to the same or semantically equivalent failure effect shall use consistent Severity ratings. Equivalent effects are expected to represent the same level of impact on the customer, system, safety, quality, or operational performance and therefore should not be assigned significantly different Severity values without clear justification. [...]"

- **Between-projects**: le inconsistenze vengono valutate tra FMEA items appartenenti a due progetti diversi. Le metriche valutate sono:

  – Per items che riguardano lo stesso guasto:

  FMEA Consistency Between Projects (similar Failure Mode) Prompt

  "[...]
  ## Metrics
  - **Consistent Detection Values**:
    - **Description**: FMEA items within the same project that refer to the same or semantically equivalent failure mode shall use consistent Detection ratings. Equivalent failure modes are expected to have a comparable probability of being detected by existing controls and therefore should not be assigned significantly different Detection values without clear justification. [...]
  - **Consistent Effects**:
    - **Description**: FMEA items within the same project that refer to the same or semantically equivalent failure mode shall identify consistent failure effects. Equivalent failure modes are expected to lead to the same, or substantially equivalent, system, customer, safety, quality, or operational consequences, and should not be associated with conflicting or unrelated effects without clear justification.
  [...]"

  – Per items che riguardano lo stesso guasto e con cause simili:

FMEA Consistency Between Projects (similar Failure Mode and Cause) Prompt

"[...]
Metrics
- **Consistent Occurrence Values**:
  - **Description**: FMEA items within the same project that refer to the same, or semantically equivalent, combination of failure mode and failure cause shall use consistent Occurrence ratings. Equivalent failure mode-cause combinations are expected to represent the same likelihood of occurrence and therefore should not be assigned significantly different Occurrence values without clear technical justification.
[...]"

– Per items che riguardano lo stesso guasto e con effetti simili:

FMEA Consistency Between Projects (similar Failure Mode and Effect) Prompt

"[...]
Metrics
- **Consistent Severity Values**:
  - **Description**: FMEA items within the same project that refer to the same, or semantically equivalent, combination of failure mode and failure effect shall use consistent Severity ratings. Equivalent failure mode-effect combinations are expected to represent the same level of impact on the customer, system, safety, quality, or operational performance and therefore should not be assigned significantly different Severity values without clear technical justification.
[...]"

3.4 V&V

I requisiti definiti e analizzati da Rana (Sezione 3.2), devono essere testati e validati dai System Engineers, mediante vari tipi di test. Prendendo come esempio il seguente requisito:

The washing machine shall detect load size automatically.

i test utilizzati sono:

- **Test Objectives**: sono test che rispondono alla domanda: *cosa voglio dimostrare?*. Ogni Test Objective raggruppa uno o più Test Cases. In riferimento al requisito di cui sopra, un esempio di Test Objective può essere:

  Verify load detection accuracy.

- **Test Cases**: sono test che rispondono alla domanda: *come dimostro l'obiettivo?*. Esempio di Test Case:

  Load machine with:
  - 2 kg
  - 5 kg
  - 8 kg

  Compare estimated load vs real load.

- **Test Sessions**: sono test che rispondono alla domanda: *quando, dove e con quali test eseguo la verifica?* Ogni Test Session riguarda sempre un solo Test Objective. Un esempio di Test Session è:

  KDP8 Verification Campaign

A fronte di centinaia di requisiti e migliaia di test, il tool V&V, oggetto dell'ultima delle due PoC, ha come obiettivo l'analisi dei test caricati nel sistema. Anche per questa PoC, il cliente ha delineato gli Use Cases riportati in Tabella 3.6. Anche per questa PoC, gli Use Cases sono volutamente generici e vanno considerati nella totalità del tool sviluppato.

Tabella 3.6: V&V - Use Cases Covering

UC	Use Cases Description
UC1	**Requirement V&V Strategy.** Propose applicable Test Cases for a Requirement (traceability matrix) and suggested TeRL.
UC2	**Product/Solution V&V Strategy.** Propose a sequence of Test Objectives and Test Cases based on project scope.
UC3	**Applicable Test Cases for an Architectural Element.** List applicable Test Cases for an architectural element based on project scope.
UC4	**Coverage Test vs. Requirements and Risks.** Highlight V&V gaps at different levels and for different elements of the SoI.
UC5	**Create Test Cases (and/or Test Objectives) Based on Requirements and Risks.** Create Test Cases to verify requirements and investigate risks without a best practice.

3.4.1 Input: Requisiti, Test Objectives, Test Cases Test Sessions

Una parte delle informazioni che l'utente deve fornire al tool, corrisponde esattamente con un altro input centrale in Rana: i requisiti, pur avendo un ruolo meno centrale, sono indispensabili per valutare la copertura rispetto ai test, come definito nelle analisi (Sottosezione 3.4.4).

Oltre al documento dei requisiti, un excel identico a quello fornito a Rana, l'utente fornisce anche i file excel contenenti Test Objective, Test Cases e Test Sessions, i cui campi sono riportati nelle rispettive tabelle e le relazioni ricavate.

3.4.2 Data Model

Il data model per V&V si limita a salvare le informazioni contenute nei file di input e a rappresentare quelle che li legano tra loro, come mostrato nel diagramma Entità-Relazione in Figura 3.10. A differenza di Rana, però, non è presente l'entità `user_requirements` (Sottosezione 3.2.2), in quanto per V&V non è prevista la modifica del contenuto dei requisiti all'interno del proprio workspace. Pertanto, ogni entità è, di fatto, condivisa a tutti gli utenti.

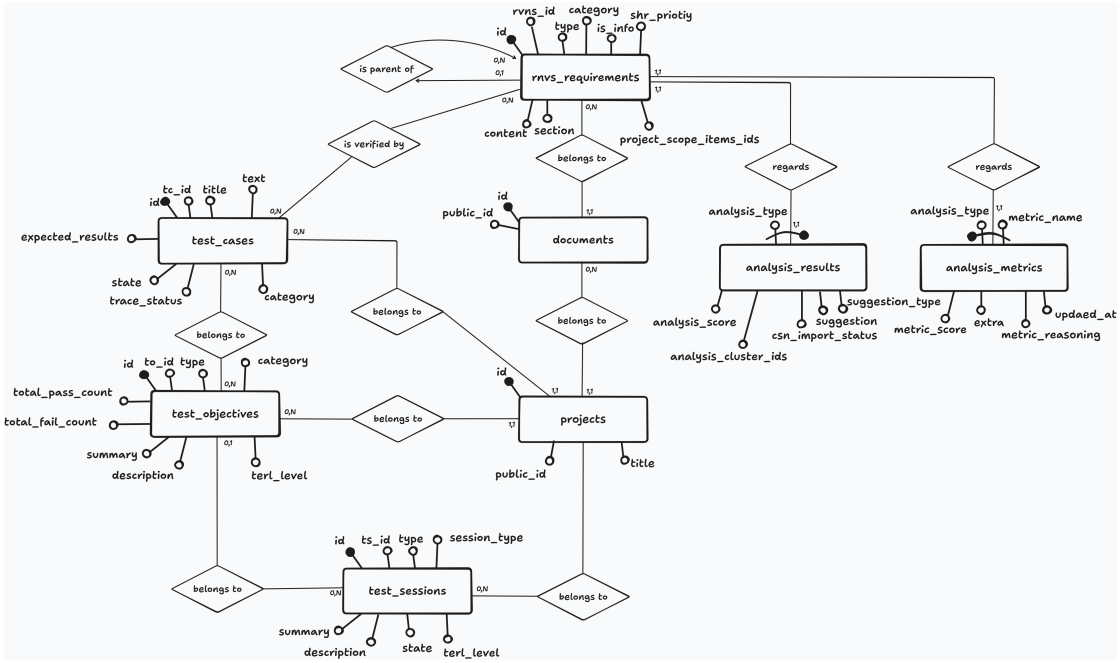


Figura 3.10: Schema ER di V&V.

3.4.3 Analisi: metriche, score e reasoning

Anche per V&V, l'utente può eseguire analisi sui requisiti o sui test disponibili nel sistema: in base al tipo di analisi, per ogni requisito o test analizzato, viene restituito uno score per ogni metrica analizzata. Come per gli altri tool, gli score assegnabili sono 0.0, 0.3, 0.7 e 1.0, in base al grado di bontà del requisito o del test rispetto alla metrica in analisi.

Come per FMEA (Sottosezione 3.3.3), lo score finale del requisito o del test analizzato è ricavato come il minimo score tra le metriche valutate per quel tipo di analisi. Lo score per analisi in tutto il tool, invece, è la media degli score finali di requisiti e test per tipo di analisi.

Le analisi che l'utente può lanciare sono **correttezza** e . Ai fini della PoC, non è prevista un'analisi di completezza.

3.4.4 Correctness

L'analisi di correttezza è valutata su requisiti o test valutati singolarmente; la maggior parte delle metriche sono valutate in maniera deterministica, senza l'ausilio di LLM. La correttezza può essere lanciata su requisiti, test objectives e test cases.

Correctness sui requisiti

L'unica metrica valutata, e solo per requisiti di tipo Subsystem e Interface (SSR e IR, Figura 3.3), è *Missing TeRL8 Coverage*, che si limita a verificare la copertura del requisito rispetto ai Test Cases; in particolare, se si tratta di requisiti fisici (ossia quelli che contengono *Test* nel campo *TeRL8*), la metrica ha score 1.0 se, tra i Test Cases (anche questi fisici) legati al requisito in analisi tramite il campo `verified_by` della tabella `rvns_requirements`, almeno uno di questi rientra in un Test Objective con $TeRL = 8$; altrimenti, lo score è 0.0.

Qualora invece non si tratti di test fisici, la mancata copertura è meno penalizzante: si va a ricercare, infatti, Test Objective TeRL8 legati ai Test Cases, relativi al requisito in analisi, della stessa categoria. In caso di successo lo score è di 1.0, altrimenti 0.7.

Il campo $TeRL$, Technology Readiness Level, indica il livello di maturità tecnica di una certa soluzione: quello su cui si intende focalizzarsi per quest'analisi, secondo le direttive del cliente, è un livello molto elevato.

Correctness sui Test Objectives

Le metriche valutate sono:

- *Traceability VS TO*: valutata deterministicamente solo per i test con $TeRL = 8$, restituisce uno score di 1.0 è legato almeno a un Test Case, 0.0 altrimenti.

- *Traceability VS TS*: valutata deterministicamente solo per i test con $TeRL = 8$, restituisce uno score di 1.0 è coperto da almeno una Test Session, 0.0 altrimenti.
- *Missing / Weak Pass-Fail Criteria*: questa metrica valuta la bontà del criterio con il quale si determina il fallimento o il successo dei Test Cases; pertanto, avrà score 0.0 se non ci sono Test Cases collegati al Test Objective in analisi, altrimenti uno score tra 0.3, 0.7 e 1.0 in base a quanto i criteri dei Test Cases associati siano, nel complesso, specifici, misurabili e non ambigui. Il criterio di successo dei Test Cases è estratto dal campo `expected_results` della tabella relativa (Figura 3.10).

Correctness sui Test Cases

Analogamente ai Test Objectives, le metriche *Traceability VS TO* e *Traceability VS Req* valutano se il test analizzato, rispettivamente, appartiene ad almeno un Test Objective ed è coperto da almeno un requisito; in caso positivo, per ognuna delle due metriche, lo score è 1.0, 0.0 altrimenti.

Un Test Case si definisce **orfano** se non è coperto né da Test Objectives né da requisiti. Questa definizione è spesso presente nelle quotidiane operazioni del cliente, e verrà ripresa nel Capitolo 4.

3.4.5 Consistency

L'analisi di consistenza può essere eseguita solo sui requisiti, e, a differenza della sui requisiti in Rana (Sottosezione 3.2.5), l'analisi è eseguita su ogni requisito preso singolarmente, per cui non è necessario clusterizzarli o generare embedding. L'unica metrica valutata per questa analisi è la *Consistent TCs*: il workflow associato si occupa di prendere in input le informazioni del requisito da analizzare e dei Test Cases associati (dalla colonna `verified_by`) e valuta i test, assegnando uno score tra 0.0, 0.3, 0.7 e 1.0 per ognuno di essi, in base a quanto siano specifici nel loro scopo e se coprono l'obiettivo del requisito stesso.

Capitolo 4

Agenti

Se Rana, FMEA e V&V sono il cuore dell'intero progetto, i loro agenti sono ciò che permette all'utente di migliorare l'utilizzo dei tool, implementando talvolta funzionalità aggiuntive, anche se minori, e fungono da strumenti di supporto al loro normale utilizzo.

I tre agenti, ognuno specifico per il proprio tool, hanno a disposizione:

- i tool MCP (Sezione 1.7) forniti da GenAIX, comuni a tutti;
- i tool MCP forniti dai server MCP dei tool di cui fanno parte.

Come anticipato (Sezione 2.1), gli agenti "vivono" in AIHub, il sottomodulo di GenAIX che gestisce l'orchestrazione degli agenti ed espone i workflow utilizzati per le varie analisi. Gli agenti introdotti, `RanaAgent` (Sezione 4.1), `FmeaAgent` (Sezione 4.2) e `VnvAgent` (Sezione 4.3) ereditano le funzionalità di base dal `BaseAgent` (Sottosezione 2.1.2) già presente nel framework, che crea l'agente ReAct nativo tramite la funzione `create_agent` e definisce i middleware necessari, e si limitano a ridefinire:

- nodo in input;
- nodo in output;
- system prompt, istruttivo e specifico dell'agente.

Il modello utilizzato per ogni agente è **gpt-5**, mentre per la sintesi (`SummarizationMiddleware`) è stato utilizzato un modello più leggero, gpt-5.4-nano, con reasoning effort basso (`low`).

I tre agenti non comunicano tra di loro, in quanto solo il `RanaAgent` è quello effettivamente utilizzato nella vita quotidiana del cliente, mentre gli altri rientrano

nelle PoC sopra definite. L'integrazione di questi ultimi e l'eventuale comunicazione tra i tre agenti tramite protocollo A2A (Sezione 1.8) sarà obiettivo del prossimo rilascio.

Di seguito sono esposte le caratteristiche, in dettaglio, dei tre agenti.

4.1 Rana Agent

Il prompt del Rana Agent lo istruisce a soddisfare le richieste dell'utente, indicando le priorità e le modalità di risposta.

Il Rana Agent deve sempre utilizzare i tool MCP a disposizione, se necessari, per rispondere alla richiesta dell'utente. In particolar modo, questo tipo di comportamento è espressamente indicato come obbligatorio se la richiesta riguarda analisi di **correttezza**, **consistenza** o **completezza**.

Inoltre, il prompt spinge l'agente a rispondere in formato tabellare qualora venga presentato il contenuto di requisiti o il risultato di analisi. In Appendice B è possibile esaminare in dettaglio il prompt che utilizza l'agente.

L'utente può interagire con l'agente direttamente dall'interfaccia del SEBot, tramite l'apposito pulsante che permette di avviare una chat e passare agevolmente tra il Rana Agent e l'interfaccia grafica del tool (Figura 4.1).

4.1.1 Rana MCP server - funzionalità di base

Rana mette a disposizione un server MCP che, utilizzando il modulo `FastMCP`, espone dei tool MCP che possono essere usati direttamente dagli agenti che si trovano nell'AIHub. L'istanza del server MCP (di seguito) ha permesso di definire i tool per mezzo del decoratore `@mcp.tool`:

```
def create_mcp_server(host: str = "0.0.0.0", port: int = 6300)
        -> FastMCP:

    mcp = FastMCP(
        "rana-mcp",
        host=host,
        port=port,
        streamable_http_path="/streamable-http",
    )

    [...]
```

Ogni tool ha, tra i propri parametri, oltre a quelli che lo caratterizzano, il **delegation token** (Sezione 3.1), indispensabile per identificare l'utente che sta

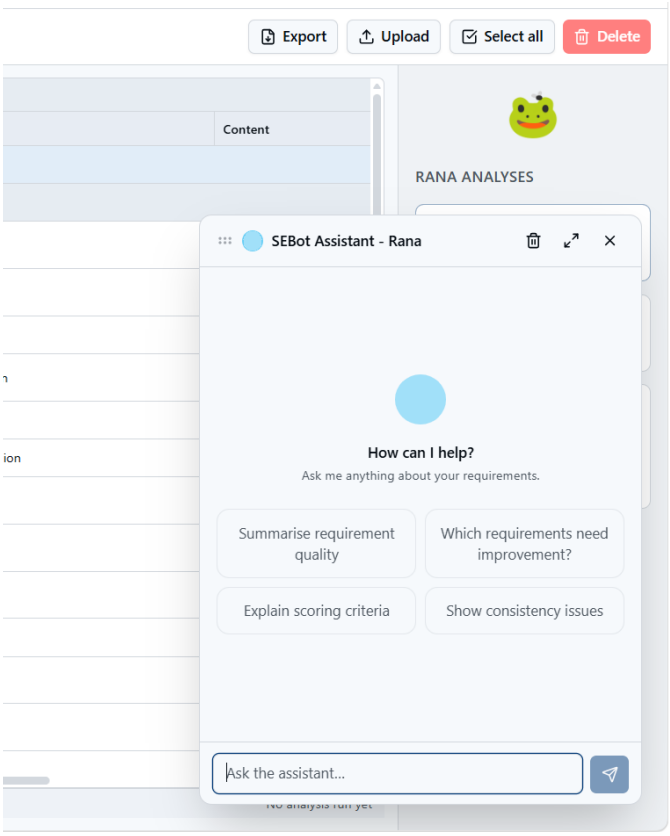


Figura 4.1: Interfaccia della chat per interagire con il Rana Agent.

di fatto interagendo con l'agente, ed eseguire le operazioni direttamente nel suo workspace. Se l'agente non fornisce il delegation token, il tool non può essere eseguito e ritorna un errore. Esempio:

```
@mcp.tool()
async def check_correctness(
    req_ids: Optional[List[str]] = None,
    delegation_token: SkipJsonSchema[Optional[str]] = None,
    user_context: Optional[str] = None,
    voting_n: Optional[int] = None,
) -> str:

    [...]
```

I tool MCP sono di fatto la porta di accesso per gli agenti alla business logic di Rana: non sono altro che dei *wrapper* alle funzioni presenti nei vari services,

che svolgono la stessa funzione dei routers nelle chiamate API REST. I principali tool permettono di fatto di effettuare operazioni eseguibili anche dal frontend del SEBot:

- `load_requirements`: permette all'utente il caricamento di requisiti allegando direttamente il file alla chat.
- `load_single_requirement`: consente il caricamento di un nuovo requisito fornendo, direttamente dalla chat, i campi necessari (titolo, categoria, testo, eccetera, Sottosezione 3.2.2).
- `get_all_requirements`: pesca tutti i requisiti che l'utente ha caricato nel proprio workspace e li ritorna all'agente (il quale, in base alla quantità, deciderà di mostrare quelli più significativi con i progetti associati).
- `update_requirement`: permette di modificare il testo di un requisito (popolando, di fatto, il contenuto del campo `content` della tabella `user_requirements`, Sottosezione 3.2.2).
- `get_projects`: restituisce i nome dei progetti caricati nel workspace. Utile per fornire una panoramica dei progetti presenti quando il workspace è pieno di requisiti.
- `check_correctness`: effettua l'analisi di correttezza (Sottosezione 3.2.4) sul requisito o sui requisiti i cui `rvs_id` sono passati come parametro. Oltre agli identificativi dei requisiti, il tool prende opzionalmente il valore di N per il meccanismo del voting e il contesto aggiuntivo (quest'ultimo approfondito nella Sottosezione 4.1.2). Il campo `voting_n` si rivela indispensabile in quanto la precisione delle analisi deve essere coerente con il livello impostato nel frontend. Infatti, tramite il middleware `_VotingContextMiddleware`, il valore importato a frontend tramite il toggle viene inserito nella conversazione al fine di essere utilizzato nel tool, se richiesto (come indicato nel prompt del Rana Agent). Di default, il valore impostato è 3 in quanto il processo di validazione da parte del cliente si è basato utilizzando il livello "High Accuracy" dal meccanismo di voting (Sottosezione 3.2.7).
- `check_correctness_for_project`: come `check_correctness`, esegue l'analisi di correttezza ma su tutti i requisiti di un progetto. Come parametro, infatti, anziché ricevere l'identificativo del singolo requisito, richiede il nome del progetto da analizzare.
- `check_consistency`: analogamente a `check_correctness`, effettua l'analisi di consistenza (Sottosezione 3.2.5) sul requisito o sui requisiti i cui `rvs_id` sono passati come parametro. Oltre agli identificativi dei requisiti e il numero

di esecuzioni del meccanismo di voting (nella stessa maniera della correttezza), l'agente deve fornire la modalità di consistenza: `by_project`, `between_projects` o `versus_libraries`. I vari controlli sulla fattibilità dell'analisi sulla base dei requisiti selezionati (come ad esempio, il fatto che nella modalità `between_project` i requisiti selezionati non possono appartenere tutti allo stesso progetto) sono delegati alla funzione `check_correctness` del `RanaService`.

- `check_consistency_for_project`: come per la `check_correctness_for_project`, esegue l'analisi di completezza sull'intero progetto il cui nome è passato come parametro. Siccome l'analisi di consistenza between project richiede due progetti, questo tool può eseguire solo analisi di consistenza by project (di default) o versus libraries.

- `check_completeness`: come le altre due analisi, permette di eseguire l'analisi di completezza (Sottosezione 3.2.6) sul requisito o sui requisiti i cui `rvs_id` sono passati come parametro. Oltre agli identificativi dei requisiti e il numero di esecuzioni del meccanismo di voting, similmente alla consistenza l'agente deve fornire la modalità di completezza: `project_scope_coverage` (modalità di default) o `ssr_cascading`.

- `get_score`: restituisce i risultati (score) delle ultime analisi su tutti i requisiti caricati nel workspace dall'utente. Risulta molto utile quando l'utente, ad esempio, chiede quali sono i requisiti migliori o quali andrebbero, invece, migliorati.

- `get_requirement`: ritorna all'agente le informazioni del requisito il cui identificativo è passato come parametro, insieme a informazioni tipiche a valle di interventi come suggerimenti a valle di analisi o informazioni sulla sua eventuale generazione.

- `get_requirement_analysis`: a differenza di `get_score`, restituisce al chiamante i dettagli delle analisi di un requisito (il cui identificativo è passato come parametro), non semplicemente soffermandosi sullo score finale ma fornendo i risultati delle singole metriche. Nella *docstring* della funzione, letta dall'agente nella fase di decisione dei tool MCP da utilizzare, compare il suo uso principale consigliato:

  > **Primary use-case**: call this tool whenever the user asks for improvement suggestions, analysis history, or wants to know whether a requirement has already been analysed — it avoids triggering a costly new analysis run when results are already stored.

- `resolve_acronyms`: semplicemente richiama la funzione `resolve_acronyms_in_text` nella business logic, chiamata ogni volta che deve essere seguita un'analisi, al fine di fornire un contesto più arricchito all'LLM. Come parametro viene passato l'acronimo di cui si chiede il significato.

L'esecuzione delle analisi non si limita solo a un'elaborazione interna alla chat: infatti, i risultati delle analisi richieste all'agente vengono prontamente salvati nel database e mostrati nel frontend del SEBot, permettendo una corrispondenza con il tool e lasciando all'utente la libertà di visualizzare in dettaglio quanto eseguito, come se le analisi fossero state lanciate direttamente premendo i relativi pulsanti al lato dell'interfaccia nel frontend.

4.1.2 Funzionalità aggiuntive

Il `RanaAgent` non si limita a replicare le funzionalità del tool. Per questa prima versione di Rana, l'agente offre altre funzionalità molto interessanti.

Contesto aggiuntivo

Qualche volta è capitato che, a valle di analisi lanciate sui requisiti, Rana segnalasse, ad esempio, la scarsa completezza di un requisito per via della mancata precisazione sull'utilizzo di alcune metodologie industriali. Tuttavia, durante la validazione, il cliente ha rimarcato che si tratta di processi standard conosciuti da tutta l'azienda; pertanto, per lo scopo di Rana, sarebbe stato opportuno evitare di segnalare inconsistenze per procedure standard riconoscibili internamente all'azienda cliente.

Questo tipo di problema non si limita all'analisi di consistenza, ma si estende anche alle metriche valutate nella correttezza. Pertanto, per questo primo rilascio, a fronte di mancate fonti di contesto aggiuntivo da cui estrarre queste informazioni di contesto, il team ha deciso di lasciare la libertà all'utente di influenzare direttamente le analisi, fornendo **contesto aggiuntivo** direttamente **tramite chat**. Il parametro aggiuntivo dei tool MCP `check_correctness` e `check_consistency`, chiamato `user_context`, racchiude le indicazioni dell'utente riguardo all'analisi che Rana sta per eseguire. Al lancio dell'analisi in questione, il system prompt aggiunge il seguente paragrafo:

Correctness / Consistency System Prompt

[...]
User-provided Additional Context:
If the user has provided additional context, take it into account when interpreting the requirement and assigning the score:

{user_context}

dove `{user_context}` è il contesto aggiuntivo passato come parametro al tool MCP ed prodotto direttamente dal Rana Agent in base alla richiesta dell'utente.

Esempio:

Can you run correctness on requirement 256784 and show me the lowest-scoring metric?

Correctness analysis — 256784

Item	Value
Requirement ID	256784
Text	"The user interface shall allow the user to start a standard washing cycle with no more than three interactions."
Overall correctness score	0.7 (pass)
Correct	1.0
Appropriate	1.0
Unambiguous	0.7
Complete	0.7
Singular	1.0
Feasible	1.0
Verifiable/Validatable	1.0
Conforming	1.0
Implementation-Free	1.0
Grammatical Clarity	1.0
Necessary	1.0

Lowest-scoring metric(s)

- Unambiguous (0.7): "Interactions" could mean button presses, touches, or menu steps.
- Complete (0.7): "Interaction" and "standard washing cycle" are not formally defined, leaving verification context partially unspecified.

Suggested improved version
The user interface shall allow a user to start the standard washing cycle with no more than three discrete interactions.
Would you like more details on any specific metric or further refinements to the requirement wording?

Actually, “interaction” and “standard washing cycle” are well-known standard procedures. Run the analysis again taking these considerations into account.

Correctness analysis (updated with clarified definitions) — 256784

Item	Value
Requirement ID	256784
Text	“The user interface shall allow the user to start a standard washing cycle with no more than three interactions.”
Overall correctness score	1.0 (pass)
Appropriate	1.0
Unambiguous	1.0
Complete	1.0
Singular	1.0
Feasible	1.0
Verifiable/Validatable	1.0
Correct	1.0
Conforming	1.0
Implementation-Free	1.0
Grammatical Clarity	1.0
Necessary	1.0

Lowest-scoring metric(s)

- All metrics scored 1.0 after considering "interaction" and "standard washing cycle" as standard, well-defined terms.

Suggested improved version
The user interface shall permit starting a standard washing cycle using no more than three interactions.
Would you like to drill into any specific metric or lock this version in as final?

Generazione di nuovi requisiti

Anche la generazione di nuovi requisiti è prerogativa dell'agente: infatti, partendo da un progetto contenente solo requisiti di tipo Stakeholder, è possibile chiedere al Rana Agent di generare requisiti di tipo System (UC8, Tabella 3.1): è sufficiente chiedere all'agente:

Can you generate System Requirements for project Washing Machine?

e l'agente chiamerà il tool MCP `generate_system_requirements`, il quale accetta come parametro il nome del progetto in questione, per poi chiamare la funzione di generazione dei requisiti all'interno della business logic.

La generazione di SR è un processo complesso e richiederà ulteriori approfondimenti nel rilascio successivo.

La funzione `generate_system_requirements` del `RanaService` controlla, dapprima, che le seguenti condizioni siano verificate:

- presenza di almeno un requisito SHR;
- presenza del documento dei CSN;
- presenza della presentazione TSG per il project scope.

Se almeno una di queste condizioni non fosse verificata, la funzione restituirebbe errore, chiedendo al chiamante di caricare le parti interessate.

Successivamente, quella che viene effettuata è una ricerca del progetto **golden standard**, ossia un progetto validato e approvato da più System Engineers e

pertanto un punto di riferimento, seppur non al pari di una libreria. Come avviene anche per le librerie, la ricerca viene effettuata per famiglia di progetti, estraendoli dal file `golden_standard_projects.json`.

Una volta stabilito il progetto golden standard di riferimento, i casi davanti ai quali Rana si può trovare sono due:

- il progetto è caricato nel workspace dell'utente che ha effettuato la richiesta;
- il progetto non è stato caricato nel workspace.

Nel primo caso, vogliamo prendere in considerazione le modifiche dell'utente su quel progetto e, pertanto, Rana esegue l'ingestion come normali requisiti, necessaria per il clustering spiegato in seguito. Nel secondo caso, invece, è necessario effettuare l'ingestion solo della tabella dei campi della `rvns_requirements` (a differenza del primo caso, in cui viene eseguita un'operazione di join con `user_requirements` per estrarne il `content`, analogamente all'ingestion dell'analisi di consistenza, Sottosezione 3.2.5).

A valle di questa distinzione, i requisiti di tipo System del progetto golden standard subiscono il processo di ingestion e successivamente vengono clusterizzati insieme ai requisiti SHR del progetto di arrivo (ossia quello per cui generiamo i SR). Quest'operazione permette di ridurre, come per l'analisi di consistenza (Sottosezione 3.2.5), il numero di requisiti da passare all'LLM.

Il processo di generazione parte dai SR del progetto golden standard, per cui i cluster contenenti solo SHR non hanno SR elegibili per la generazione e pertanto sono scartati. Per ogni cluster non scartato, viene chiamato il workflow `generate_sr`, il quale prende come input:

- `golden_standard_requirements`, i SR del progetto golden standard;
- `new_project_shrs`, gli SHR del progetto di arrivo;
- `csn_description`, la descrizione del progetto di arrivo;
- `csn_smrs`, i Stakeholder Master Requirements, ossia i requisiti estratti dai CSN la cui mancata soddisfazione costituisce il fallimento dell'intero progetto;
- `scope_in` e `scope_out`, items in scope e out of scope del project scope;
- `acronyms` e `vocabulary`.

Il system prompt per questo workflow, riportato in Appendice B, istruisce il modello ad accettare o rifiutare ognuno dei SR del progetto golden standard proposti, decidendone l'idoneità per il progetto di arrivo in base alle informazioni di

contorno fornite. Per ognuno dei SR accettati, il modello può decidere se modificare il requisito per adattarlo al progetto di arrivo e assegnargli come parent uno SHR tra quelli presenti nel cluster.

Siccome uno degli obiettivi della generazione è coprire il più possibile gli SHR del progetto da cui parte la generazione, se a valle di questa prima fase di generazione ci sono ancora SHR senza SR figli, allora questi subiscono lo stesso processo di generazione già effettuato, ma questa volta partendo da **SR di librerie** della stessa famiglia del progetto di arrivo anziché SR del progetto golden standard (fase due). Qualora dovessero esserci nuovamente SHR figli al termine della seconda fase, allora durante la terza fase Rana prova a generare nuovi requisiti, questa volta prendendo come riferimento tutte le librerie caricate nel sistema, indipendentemente dalla famiglia del progetto di arrivo.

Terminato il processo di generazione, l'agente mostra all'utente: il numero di SR generati, quanti di questi sono stati modificati rispetto al loro SR di riferimento (dal progetto golden standard o dalla libreria che sia) e la preview dei primi cinque requisiti generati. Il testo del requisito generato viene mostrato come suggerimento (analogamente a quanto accade a valle di uno score basso assegnato alla metrica *Singular*, Sottosezione 3.2.4, o dell'analisi di completezza in modalità SSR FMEA cascading, Sottosezione 3.2.6).

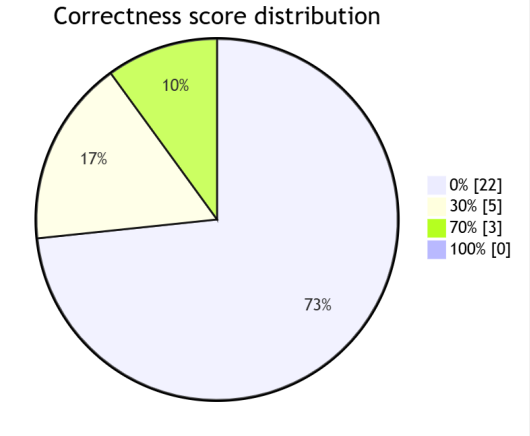
Grafici

Un'altra funzionalità che va oltre le capacità del semplice tool di Rana, offerta dalle funzionalità di base di GenAIX, è quella di generare grafici riguardo le analisi lanciate. Ad esempio:

> Can you generate a pie chart showing the distribution of correctness scores across all requirements? Each slice shall represent one of the possible scores (0%, 30%, 70%, or 100%) and its percentage of the total number of requirements analyzed.

Correctness score distribution (all analyzed requirements)
Analyzed requirements: 30

Score	Count (%)
0%	22 (73.3%)
30%	5 (16.7%)
70%	3 (10.0%)
100%	0 (0.0%)



Would you like a breakdown listing which requirements fall into each score bucket, or a project-wise view?

Il codice Mermaid generato è disponibile in Appendice B.

4.2 FMEA Agent

Per l'FMEA Agent, come per il Rana Agent, è stato definito un prompt analogo che espone all'agente le modalità d'azione, l'utilizzo di tool MCP e le modalità di presentazione dell'output all'utente. Il contenuto del prompt in formato integrale è disponibile in Capitolo B.

A differenza del Rana Agent, non c'è ancora un'interfaccia dedicata nel SEBot. Pertanto, le validazioni e le demo sono state realizzate per mezzo del sottomodulo **Portal** di GenAIX, che permette di interagire con gli agenti presenti nell'AIHub.

Come per Rana, FMEA mette a disposizione un server MCP i cui tool consentono di accedere direttamente alle funzionalità di FMEA. A differenza del Rana

Agent, trattandosi di una PoC, non sono state implementate funzionalità aggiuntive rispetto al tool (fatta eccezione per la generazione dei grafici, caratteristica integrata in GenAIX).

I tool MCP messi a disposizione da `fmea_mcp_server.py` sono:

- `list_projects`: analogamente a `get_projects` in Rana (Sottosezione 4.1.1), restituisce al chiamante i nomi dei progetti caricati nel workspace.
- `get_all_fmea_items`: ritorna al chiamante tutti gli item FMEA caricati nel workspace.
- `get_fmea_items`: restituisce al chiamante gli item associati a un progetto, il cui identificativo è passato come parametro.
- `get_project_p_diagram`: estrae il contenuto della tabella `p_diagrams` (Sottosezione 3.3.2), relativo al progetto il cui identificativo è passato come parametro.
- `run_correctness_analysis`: esegue l'analisi di correttezza su un gruppo di item FMEA i cui identificativi sono passati come parametro. I risultati dell'analisi sono restituiti all'agente, il quale li presenta in formato tabellare.
- `run_consistency_analysis`: esegue l'analisi di consistenza sugli item FMEA i cui identificativi sono passati come parametro e ne restituisce i risultati. Come per la consistenza in Rana (Sottosezione 4.1.1), la modalità di consistenza deve essere fornita come parametro aggiuntivo al tool.

4.3 V&V Agent

Come per l'FMEA Agent, il V&V Agent è parte della seconda PoC, e in quanto tale non è prevista un'interfaccia grafica per la chat nel SEBot, come accade invece per Rana.

In maniera simile agli altri due agenti, il prompt (presente in versione integrale in Appendice B) istruisce l'agente sulle modalità di ricerca della risposta, l'esecuzione dei tool MCP e il formato della risposta all'utente. Come per l'FMEA Agent, trattandosi di una PoC, i tool MCP del V&V Agent si limitano a replicare le funzionalità già offerte dal sistema, seppur con il supporto dell'agente per comparare, approfondire o sintetizzare i risultati.

I tool MCP messi a disposizione dal `vnv_mcp_server.py` sono:

- `get_projects`, restituisce i progetti caricati (per ogni tipo di entità, tra Requisiti, Test Cases, Test Objectives, Test Sessions).

- `get_requirements`: restituisce l'elenco dei Requisiti caricati.
- `get_test_objectives`: restituisce l'elenco dei Test Objectives caricati.
- `get_test_cases`: restituisce l'elenco dei Test Objectives caricati.
- `get_test_sessions`: restituisce l'elenco delle Test Sessions caricate.
- `run_correctness_requirements`: esegue l'analisi di correttezza sui Requisiti i cui identificativi sono passati come parametro e ne fornisce i risultati.
- `run_correctness_test_objectives`: esegue l'analisi di correttezza sui Test Objectives i cui identificativi sono passati come parametro e ne fornisce i risultati.
- `run_correctness_test_cases`: esegue l'analisi di correttezza sui Test Cases i cui identificativi sono passati come parametro e ne fornisce i risultati.
- `run_consistency_test_objectives`: esegue l'analisi di consistenza sui Test Objectives i cui identificativi sono passati come parametro e ne fornisce i risultati. A differenza di Rana e FMEA, c'è un'unica modalità di consistenza (by project) e pertanto non è necessario specificarla
- `run_consistency_requirements`: esegue l'analisi di consistenza sui Requisiti i cui identificativi sono passati come parametro e ne fornisce i risultati.
- `get_analysis_scores`: restituisce i risultati della analisi dell'entità il cui identificativo è passato come parametro.

Capitolo 5

Valore di business

5.1 Necessità del progetto

Ad oggi, a valle della stesura del documento dei requisiti di sistema, l'ingegnere o il team di ingegneri ha il compito di revisionarlo, verificando che:

- i requisiti siano corretti dal punto di vista grammaticale e tecnico, che siano privi di ambiguità, conformi alle norme industriali, completi e adatti alla tipologia di requisito assegnato;
- i requisiti all'interno dello stesso progetto non si contraddicano l'un l'altro, non si ripetano e utilizzino la stessa sintassi a livello di terminologia tecnica e sistema di unità di misura; inoltre, i requisiti non devono essere contraddittori gli uni con gli altri anche tra diversi progetti riguardanti gli stessi componenti industriali, mantenendo la coerenza tra di essi.
- i requisiti devono coprire interamente il progetto commissionato, senza tralasciare aspetti che talvolta sono considerati minori o semplicemente dimenticati in fase di stesura del documento. Inoltre, i requisiti non devono andare al di fuori dello scopo del progetto, con il rischio di investire il budget in aspetti non rilevanti, sottraendolo a questioni di maggiore importanza.

Di contorno, è necessario verificare la coerenza dei requisiti rispetto ai guasti legati ai macchinari di produzione, cercando di ridurre al minimo la loro frequenza, velocizzare il ripristino ed minimizzare il rischio per i lavoratori. Inoltre, la verifica dei requisiti deve essere coerente e completa verso gli stessi.

Questo lavoro risulta subito **oneroso e dispendioso** in termini di tempo investito. Una prima soluzione potrebbe essere quella di stanziare più ingegneri per ridurre il tempo di scrittura e revisione dei requisiti; tuttavia, ogni ingegnere ha il proprio livello di maturità ed esperienza, e molto spesso risultano non allineati tra

di loro dal punto di vista delle conoscenze e del linguaggio tecnico: la coordinazione di gruppi di numerosi ingegneri risulterebbe anch'essa dispendiosa in termini di tempo, lasciando sempre un minimo e non trascurabile **margine di errore**.

L'intero progetto, durato da novembre 2025 ad agosto 2026, si è incentrato principalmente sull'industrializzazione del tool per l'analisi dei requisiti di sistema, Rana (Sezione 3.2), e sullo sviluppo di due PoC, FMEA (Sezione 3.3) e V&V (Sezione 3.4), per l'analisi di guasti e test, con l'obiettivo, in future industrializzazioni, di affiancare Rana. L'intelligenza artificiale applicata a questo campo potrebbe risultare innovativa, delegandole mansioni di controllo e ripetitive.

Al fine di fornire una stima del valore di ritorno al cliente, il team ha raccolto i dati di **tempi** e **costi** attraverso le tracce di Langfuse. Di seguito vendono riportati i suddetti dati, in termini di costi e tempo delle analisi di Rana. Trattandosi di PoC, per FMEA e V&V non sono stati raccolti dati, in quanto l'obiettivo è solo quello di dimostrarne l'effettivo potenziale e non un pronto rilascio.

5.2 Analisi dei Tempi (Time-to-Value)

Al fine di fornire una stima sui tempi medi di esecuzione delle analisi, sono stati presi in esame **200 requisiti** forniti dal cliente e sono stati eseguiti i principali workflow disponibili nell'AIHub per Rana. La Tabella 5.1 riporta il tempo di risposta medio per requisito in base al workflow; la stima non tiene conto del meccanismo di voting ($voting_n = 1$), né della parallelizzazione.

Workflow	**Latenza media per requisito (s)**
Proper Check	6.20
Correctness (CSN loaded)	14.12
Consistency By Project	2.43
Consistency Between Projects	2.58
Project Scope Linking	2.80
Consistency Versus Library	3.19
Consistency Between Libraries	1.29
Merge Suggestions	5.91

Tabella 5.1: Latenza media per requisito dei principali workflow di Rana.

5.2.1 Correctness

Nel calcolo del tempo medio di esecuzione, trascuriamo l'eventuale necessità di unire i due suggerimenti (Merge Suggestions, Figura 3.6). Prendendo in esame

l'analisi di correttezza sul singolo requisito di tipo Stakeholder, i workflow eseguiti sono Proper Check e Correctness, quindi per un totale di circa **20 secondi**. Tenendo conto che, di default, a causa del meccanismo di voting il processo è ripetuto 3 volte, il tempo totale di esecuzione sale a 1 minuto. Tuttavia, grazie alla parallelizzazione introdotta (Sottosezione 3.2.3), questo tempo scende drasticamente. Quindi, siccome il cliente ha richiesto di garantire l'utilizzo contemporaneo di 65 utenti senza interruzioni, ipotizzando di avere in fase di deploy $user_sem = 5$ e $global_sem = 200$ (ossia, $global_sem$ trascurabile), la stima del tempo di esecuzione dell'analisi di correttezza su un unico requisito diventa:

$$\begin{aligned} t_{correctness} &= (t_{proper_wf} + t_{correctness_wf}) \cdot \left\lceil \frac{voting_n}{user_sem} \right\rceil = \\ &= (6.20s + 14.12s) \cdot \left\lceil \frac{3}{5} \right\rceil \approx 20.32s \end{aligned} \tag{5.1}$$

Ipotizzando, invece, di voler eseguire l'analisi su più di un requisito in successione, il semaforo permette di abbassare ulteriormente il tempo medio di esecuzione; considerando un numero medio di requisiti per progetto, $N = 100$, si ottiene:

$$\begin{aligned} t_{project} &= (t_{proper_wf} + t_{correctness_wf}) \cdot \left\lceil \frac{N \cdot voting_n}{user_sem} \right\rceil = \\ &= (6.20s + 14.12s) \cdot \left\lceil \frac{100 \cdot 3}{5} \right\rceil \approx 1219.56s \\ t_{correctness} &= \frac{t_{project}}{N} \approx 12.20s \end{aligned} \tag{5.2}$$

Il vantaggio offerto del semaforo sul tempo medio di esecuzione, con `user_sem` pari a 5, è chiaro: analizzando $N = 100$ requisiti in sequenza, il tempo medio di esecuzione sarebbe 20 secondi (Eq. (5.1)), contro i circa **12 secondi** con l'introduzione della parallelizzazione, riducendo drasticamente il tempo di esecuzione del 39%. Il principale collo di bottiglia rimane il semaforo `user_sem` (insieme a `global_sem`, anche se con impatto minore). Qualora le risorse cloud lo permettano, il cliente può anche decidere di aumentare questo limite conservativo imposto sul semaforo, permettendo una maggiore parallelizzazione e quindi un tempo di esecuzione ancora al di sotto di quello appena stimato.

Per requisiti di tipo System, non dovendo eseguire il workflow di Proper Check, il tempo medio di esecuzione dell'analisi di correttezza si abbassa a:

$$
\begin{aligned}
t_{project} = t_{correctness_wf} \cdot \left\lceil \frac{N \cdot voting_n}{user_sem} \right\rceil = \\
= 14.12s \cdot \left\lceil \frac{100 \cdot 3}{5} \right\rceil \approx 847.44s \\
t_{correctness} = \frac{t_{project}}{N} \approx 8.47s
\end{aligned}
\tag{5.3}
$$

Rispetto ai requisiti di tipo Stakeholder, l'assenza del workflow Proper Check, riduce il tempo di esecuzione medio per requisito sull'analisi di correttezza da 12 secondi (Eq. (5.3)) a 8 secondi e mezzo circa.

5.2.2 Consistency

Per stimare i tempi di esecuzione per l'analisi di consistenza, è doveroso fare una serie di considerazioni:

- la generazione di embedding ha un tempo medio stimato di circa 3 secondi; siccome è un'operazione che viene effettuata in situazioni estremamente rare (solo la prima volta che si lancia l'analisi, o in altre procedure con il linking di Project Scope e CSN, e quando si cambia il contenuto dei requisiti), per queste stime il tempo di ingestion dei requisiti è trascurato;

- l'esecuzione dell'analisi di consistenza avviene per cluster, non per singolo requisito. Pertanto, il tempo medio di esecuzione del workflow della consistenza associato a ogni requisito è strettamente legato al numero (estremamente variabile) di elementi del cluster. Siccome il workflow esegue gruppi, l'esecuzione della consistenza è mediamente più veloce rispetto alla correttezza;

- come per la correttezza, trascuriamo l'eventuale tempo dedicato alla fusione di più suggerimenti (workflow Merge Suggestions, Figura 3.6).

Anche per quest'analisi, dobbiamo considerare che i semafori sono utilizzati in maniera conservativa e pertanto i tempi medi di esecuzione possono, potenzialmente, ridursi rispetto a quelli riportati in seguito.

$$
N_{cluster_byP} = \frac{N}{dim_cluster} = 10 \tag{5.4}
$$

Analizzando l'analisi di Consistency By Project, il tempo medio di esecuzione per requisito risulta essere:

$$t_{project} = t_{consistency_byP_wf} \cdot \left\lceil \frac{N_{cluster_byP} \cdot voting_n}{user_sem} \right\rceil = \\ 2.43s \cdot \left\lceil \frac{10 \cdot 3}{5} \right\rceil \approx 14.57s \\ t_{consistency_byP} = \frac{t_{project}}{N} \approx 0.15s \tag{5.5}$$

Per l'analisi di Consistency Between Projects, assumendo che i due progetti abbiano dimensioni analoghe ($N = 100$ ciascuno) e mantenendo $dim_cluster = 10$ il numero di cluster diventa:

$$N_{cluster_betP} = \frac{2 \cdot N}{dim_cluster} = 20 \tag{5.6}$$

Per cui, il tempo medio di esecuzione per requisito diventa:

$$t_{projects} = t_{consistency_betP_wf} \cdot \left\lceil \frac{N_{cluster_betP} \cdot voting_n}{user_sem} \right\rceil = \\ = 2.58s \cdot \left\lceil \frac{20 \cdot 3}{5} \right\rceil \approx 30.98s \\ t_{consistency_betP} = \frac{t_{projects}}{2 \cdot N} \approx 0.15s \tag{5.7}$$

L'esecuzione dell'analisi di Consistency Versus Libraries dipende molto dal numero di librerie prese in esame: maggiore è il numero di librerie confrontate con il progetto in esame, maggiori sono i requisiti di librerie e più cluster saranno analizzati. Ipotizzando che le librerie abbiano $N = 100$ requisiti ciascuna, ponendo k pari al numero di librerie confrontate, il numero di batch diventa:

$$N_{cluster_betP} = \frac{(k+1) \cdot N}{dim_cluster} = 10k + 10 \tag{5.8}$$

E pertanto il tempo di esecuzione medio per requisito diventa:

$$t_{project} = t_{consistency_vsL_wf} \cdot \left\lceil \frac{N_{cluster_betP} \cdot voting_n}{user_sem} \right\rceil = \\ = 3.19s \cdot \left\lceil \frac{(10k+10) \cdot 3}{5} \right\rceil \approx (k+1) \cdot 19.12s \\ t_{consistency_vsL} = \frac{t_{project}}{N} \approx (k+1) \cdot 0.19s \tag{5.9}$$

Da notare che qui il tempo di esecuzione del batch è diviso per N, in quando i risultati riguardano esclusivamente il progetto analizzato e pertanto è corretto

eseguire i calcoli sul tempo rispetto solo a quei requisiti. In conclusione, il tempo di esecuzione della Consistency Versus Libraries cresce linearmente con il numero di librerie confrontate.

5.2.3 Completeness

Per la stima di questa analisi viene presa in considerazione solo la modalità Project Scope Coverage, in quanto l'altra modalità richiede l'intervento di FMEA (non industrializzato) e pertanto non è ancora rilasciata.

Come già affrontato (Sottosezione 3.2.6), l'analisi di completezza in modalità Project Scope Coverage è praticamente **deterministica**. Tuttavia, la parte aleatoria dell'analisi è proprio il linking del Project Scope.

Considerando che il linking riguarda solamente i requisiti di tipo System, ipotizzando che nel progetto con $N = 100$ requisiti siano tutti SR, e considerando che il workflow prende i requisiti a gruppi di 3 alla volta, il tempo medio di linking del Project Scope risulta essere:

$$\begin{aligned} t_{project} = t_{ps_linking_wf} \cdot \left\lceil \frac{N/3 \cdot voting_n}{user_sem} \right\rceil = \\ = 2.81s \cdot \left\lceil \frac{100/3 \cdot 3}{5} \right\rceil \approx 56.13s \\ t_{ps_linking} = \frac{t_{project}}{N} \approx 0.56s \end{aligned} \tag{5.10}$$

La regolazione della precisione di questo processo, come anticipato nella Sottosezione 3.2.6, è al di fuori della gestione dell'utente e pertanto $voting_n = 3$ sempre. Inoltre, a meno di modifiche dei requisiti o caricamento di una nuova presentazione TSG, questa operazione viene eseguita una sola volta, pertanto nuove analisi di completezza vanno a ridurre l'impatto in termini di tempo del Project Scope Linking.

5.2.4 Confronto con il System Enigneer

Dalle stime fornite dagli ingegneri meccanici, è noto che i tempi di esecuzione delle analisi di correttezza, consistenza e completezza da parte di un Junior System Engineer e un Senior System Engineer sono riportati nella Tabella 5.2.

Dalla tabella del confronto dei tempi di emerge subito che Rana riduce drasticamente i tempi di esecuzione delle analisi rispetto a un Senior System Engineer (preso di riferimento nei seguenti commenti, in quanto più veloce rispetto al suo corrispondente Junior): Rana performa meglio di un System Engineer (SE), in termini di tempo, per tutte le analisi che può effettuare. In particolare, si può

Attività	Junior SE	Senior SE	Rana
Correctness (SHR)	1.5–3 min/req.	1–2 min/req.	12.20 s/req.
Correctness (SR)	1.5–3 min/req.	1–2 min/req.	8.47 s/req.
Consistency By Project	0.75–1.5 min/req.	0.5–1 min/req.	0.15 s/req.
Consistency Between Projects	1.5–3 min/req.	1–2 min/req.	0.15 s/req.
Consistency VS Libraries ($k = 2$)	3–4.5 min/req.	2–3 min/req.	0.57 s/req.
Completeness (PS Linking)	1.5–3 min/req.	1–2 min/req.	0.56 s/req.

Tabella 5.2: Confronto tra il tempo medio stimato di revisione manuale da parte di un Junior e Senior System Engineer e il tempo medio di esecuzione di Rana.

notare che il vantaggio minore è dato dall'analisi di Correctness sugli SHR, con una velocità di quasi 7 volte e mezzo rispetto a un Senior SE. Invece, laddove la differenza è maggiore è nell'analisi di Consistency Between Projects, in cui Rana analizza i requisiti **600 volte** più veloce rispetto al SE, seguito dalla Consistency Versus Libraries in cui è più veloce di 263 volte.

Il **Time To Value**, ossia il tempo che intercorre tra l'inizio dell'analisi e la disponibilità dei primi risultati utilizzabili dall'utente, è calcolato in maniera differente rispetto al tempo stimato per requisito. Infatti, se per l'analisi di correttezza il tempo riportato nella Tabella 5.2 corrisponde al TTV, per l'analisi di consistenza, affinché il System Engineer controlli le inconsistenze tra tutti i requisiti, è necessario che scansioni il documento nella sua interezza; Rana, invece, una volta analizzato un cluster, restituisce subito il risultato prima di passare al successivo, riducendo il TTV al singolo cluster. Ipotizzando $N = 100$ e $dim_cluster = 10$, come prima, i valori di TTV riportati nella Tabella 5.3 evidenziano come Rana sia in grado di fornire i primi dati dopo pochi secondi, mentre in alcuni casi il System Engineer ha bisogno di diverse ore.

Attività	TTV Senior SE	TTV Rana
Correctness (SHR)	1–2 min	12.20 s
Correctness (SR)	1–2 min	8.47 s
Consistency By Project (100 req)	50–100 min	1.50 s
Consistency Between Projects (200 req)	200–400 min	1.50 s
Consistency Versus Libraries ($k = 2$, 300 req)	600–900 min	5.70 s
Completeness (Project Scope Linking)	1–2 min	0.56 s

Tabella 5.3: Confronto del Time To Value (TTV) tra un Senior System Engineer e Rana.

5.3 Analisi dei Costi (Ottimizzazione)

Anche per l'analisi dei costi, sono stati presi in esame **200 requisiti** forniti dal cliente, i quali sono stati dati in pasto ai principali workflow di Rana. Utilizzando Azure OpenAI come provider, il costo dei workflow dipende da una serie di fattori come: token in input, token in output, token trovati in cache. I costi medi ricavati per requisito, a valle dell'esecuzione dei workflow, sono riportati nella Tabella 5.4.

Workflow	**Costo medio per req. ($)**	**Token output medi**
Proper Check	0.00032	222
Correctness (CSN loaded)	0.00177	1456
Consistency By Project	0.00085	766
Consistency Between Projects	0.00156	2999
Project Scope Linking	0.00066	359
Consistency Versus Library	0.00106	668
Consistency Between Libraries	0.00063	740
Merge Suggestions	0.00033	374

Tabella 5.4: Costo medio per requisito e numero medio di token di output dei principali workflow di Rana.

Siccome i costi riportati sono già calcolati per requisito, supponiamo ancora una volta progetti da $N = 100$ requisiti (di cui 50 SHR e 50 SR) al fine di calcolare il costo totale per progetto.

5.3.1 Correctness

Per la correttezza dei requisiti di tipo Stakeholder, occorre nuvomente considerare non solo il workflow Correctness ma anche Proper Check. Pertanto, il costo totale sul progetto della Correctness sugli SHR, ipotizzando che i requisti SHR siano $N_{SHR} = 50$, diventa:

$$\begin{aligned} costo_{correctness_SHR_req} &= costo_{Proper_Check} + costo_{correctness_wf} = 0.00209\$ \\ costo_{correctness_SHR} &= costo_{correctness_SHR_req} \cdot N_{SHR} = 0.1045\$ \end{aligned} \tag{5.11}$$

Il costo dell'analisi di correttezza sui SR, invece, è calcolato semplicemente:

$$costo_{correttezza_SR} = costo_{correctness_wf} \cdot N_{SR} = 0{,}0885\$ \tag{5.12}$$

5.3.2 Consistency

Per ricavare il costo dell'analisi di Consistency By Project è sufficiente moltiplicare il costo sul singolo requisito per N:

$$costo_{consistency_byP} = costo_{consistency_wf} \cdot N = 0{,}085\$ \tag{5.13}$$

Invece, il costo della Consistency Between Projects deve essere calcolato sui due progetti confrontati. Ipotizzando che abbiamo lo stesso numero di requisiti, il costo totale di analisi del progetto diventa:

$$costo_{consistenza_betP} = costo_{consistency_betP_wf} \cdot N \cdot 2 = 0.312\$ \tag{5.14}$$

Essendo il costo della Consistency Versus Libraries in Tabella 5.4 calcolato rispetto alle k librerie, il costo totale dell'analisi deve essere calcolato considerando il numero di librerie con cui si fa l'analisi. Considerando $k = 2$ il costo totale risulta essere:

$$costo_{consistency_vsL} = costo_{consistency_vsL_wf} \cdot N \cdot (k + 1) = 0.318\$ \tag{5.15}$$

Avendo stabilito di voler valutare il costo di un'analisi completa sui requisiti di un progetto, il costo del workflow Merge Suggestions non è più trascurabile. Avendo, per ogni requisto, lanciato l'analisi di correttezza e consistenza (quest'ultima nelle sue tre diverse modalità) ipotizzando che ogni analisi generi un suggerimento, il costo di questo workflow cambia in base all'ordine di lancio delle analisi. Si consideri come esempio questa sequenza:

1. L'utente lancia l'analisi di correttezza → un suggerimento generato, nessun merge necessario.
2. L'utente lancia l'analisi di consistenza by project → primo merge effettuato.
3. L'utente lancia l'analisi di consistenza bewteen → secondo merge effettuato.
4. L'utente lancia l'analisi di consistenza versus libraries → terzo merge effettuato.

A valle di questa sequenza, il workflow Merge Suggestion è chiamato 3 volte. Se invece cambiassimo i lanci delle analisi nell'ordine 2. → 3. → 4. → 1., il workflow Merge Suggestions verrebbe chiamato solo 1 volta, perché è necessario aver lanciato sia la consistenza che la correttezza prima di avere due suggerimenti (siccome i risultati e i suggerimenti delle analisi di consistenza sostituiscono i precedenti

della stessa tipologia, indipendentemente dalla modalità; ad esempio, lanciando prima l'analisi di consistenza by project e poi between project, i risultati della seconda sovrascrivono quelli della prima). Pertanto, consideriamo il caso medio, e quindi $N_{merge} = 1.5$ volte. Il costo del workflow Merge Suggestion, su N requisiti, pertanto risulta essere:

$$costo_{merge} = costo_{merge_wf} \cdot N \cdot N_{merge} = 0.0495\$ \tag{5.16}$$

5.3.3 Completeness

Per quanto riguarda la completezza, ancora una volta, il costo dipende unicamente dal Project Scope Linking, e pertanto, ipotizzando che il progetto contenga $N_{SR} = 50$ SR:

$$costo_{completezza} = costo_{ps_linking_wf} \cdot N_{SR} = 0.033\$ \tag{5.17}$$

5.3.4 Confronto con il System Engineer

Supponendo che un Senior System Engineer guadagni mediamente $24.33€/h$ (circa $28.00\$/h$) e un Junior Senior System Engineer guadagni in media $13.50€/h$ (circa $15.64\$/h$), secondo i dati forniti dal cliente, e immaginando che l'analisi completa di un documento richieda una giornata lavorativa ($8h$) per un System Engineer con esperienza, mentre due giornate per un neolaureato, il confronto è riportato nella Tabella 5.5.

Attività	**Junior SE ($)**	**Senior SE ($)**	**Rana ($)**
Correctness (SHR)	–	–	0.1045
Correctness (SR)	–	–	0.0885
Consistency By Project	–	–	0.0850
Consistency Between Projects	–	–	0.3120
Consistency Versus Libraries ($k = 2$)	–	–	0.3180
Completeness (PS Linking)	–	–	0.0330
Merge Suggestions	–	–	0.0495
Tempo stimato	16 h	8 h	< 1 h
Totale	**216.00**	**224.00**	**0.9905**

Tabella 5.5: Costo stimato per le eseguire le analisi analisi complete su un intero progetto, per un Junior SE, Senior SE e Rana.

Anche dall'analisi dei costi emerge che conviene investire risorse economiche per l'utilizzo di token del provider piuttosto che impiegare personale per svolgere mansioni di analisi sui requisiti di sistema.

Capitolo 6

Conclusioni e progetti futuri

6.1 Il nuovo ruolo del System Engineer

Quanto approfondito nella Sezione 5.2 e nella Sezione 5.3 esalta l'enorme vantaggio fornito da Rana (e, più in generale, dalla GenAI) in relazione al lavoro di analisi di requisiti di sistema.

Le mansioni ripetitive, time-consuming e soggette a frequenti errori, come quelle descritte nel Capitolo 3, prima effettuate manualmente dai System Engineers, ora sono effettivamente delegate a Rana. I System Engineer, adesso, assumono il nuovo ruolo di **revisori finali**: infatti, seppur Rana sia veloce, non è affidabile quanto un ingegnere, in quanto, nonostante il ricco contesto fornito per i suoi processi, non possiede la **knowledge** di un System Engineer, il quale ha dimestichezza con le procedure e piena conoscenza dei processi tipici dell'azienda in cui sta lavorando.

Inoltre, una forte limitazione della GenAI in generale consiste nel fatto che ad oggi non è in grado di effettuare un'affidabile **analisi costi-benefici**, per cui le analisi di Rana non tengono conto di questo fattore non indifferente. Un System Engineer, invece, conoscendo i costi e gli obiettivi dei progetti, ha ben chiaro il quadro generale e ha il compito di stabilire la **direzione** che le analisi devono seguire (si pensi allo `user_context` fornito dall'ingegnere a Rana per l'esecuzione delle analisi, Sottosezione 4.1.1).

In ottica di futuri rilasci (Sezione 6.2), allargando la visibilità del SEBot, l'ingegnere non dovrà più controllare allineamenti perfetti tra documenti di requisiti, documenti FMEA e test, ma sarà compito degli agenti sviluppati coordinare i controlli e fornire i risultati all'utente.

Delegando queste mansioni di relativa bassa complessità all'AI, adesso i System Engineers possono concentrare la loro attenzione su problemi complessi ad alto livello, aumentando così la produttività generale.

6.2 Progetti futuri

Dal mese di agosto 2026 è iniziata la fase di *maintenance* dell'applicazione AI sviluppata, in cui il team tecnico raccoglierà i feedback dei System Engineers che useranno Rana nei mesi successivi, affiancata anche dallo sviluppo di nuove feature e migliorie, come ad esempio:

- abbattimento ulteriore dei costi, ad esempio dividendo i prompt delle analisi per metrica (o gruppi di metriche);
- sviluppo di grafi LangGraph più complessi, per funzionalità avanzate;
- aumento della qualità delle analisi, attraverso il raffinamento del contesto;
- aumento della qualità dei requisiti generati, riducendo al minimo l'intervento umano di correzione a valle della generazione;
- introduzione di nuove analisi più sofisticate.

Obiettivo del prossimo progetto non riguarda solo le migliorie da applicare su Rana, ma anche l'industrializzazione dei tool FMEA e V&V, i quali lavoreranno a stretto contatto con Rana al fine di allineare i requisiti rispetto a documenti FMEA e test (come precedentemente accennato, Sezione 6.1), permettendo l'introduzione di nuove analisi incrociate (un'anteprima è già stata fornita dall'analisi di completezza in modalità SSR FMEA Cascading, Sottosezione 3.2.6). I due nuovi agenti che si andranno a formare potranno comunicare direttamente con il già esistente Rana Agent per mezzo del protocollo A2A (Sezione 1.8), per ulteriori funzionalità che andranno, man mano, a ottimizzare sempre di più il lavoro degli ingegneri e, perché no, anche a generare prodotti di qualità sempre maggiore a costi sempre più ridotti.

Appendice A

Rana - metriche e prompt

Di seguito vengono riportate le porzioni di prompt che mettono in evidenza le metriche utilizzate per l'analisi di consistenza.

A.1 Consistency by project

- SHR (e SHR Libraries)

  SHR Consistency By Project Prompt

  "[...]
  ## Metrics
  - **Clear - No Conflicts**:
  - **Description**: The stakeholder requirement shall not contradict or conflict with other stakeholder requirements in terms of objectives, constraints, priorities, or expected outcomes.
  [...]"

- SR (e SR Libraries)

  SR Consistency By Project Prompt

  "[...]
  ## Metrics
  - **No Duplicates**:
  - **Description**: Each requirement in the set must be unique. No requirement shall duplicate exactly the intent, structure, or semantics of another. If two apparently duplicate requirements trace to DIFFERENT parent requirements, they should be flagged as duplicate only if

their similarity is not justified by distinct stakeholder needs. [...]
- **No Conflicts**:
  - **Description**: Requirements shall not contradict each other in logic, constraint, performance, behavior, or scope. [...]
- **Consistent Terminology**:
  - **Description**: Terminology shall be used consistently across all requirements. Same concept shall be expressed with the same terminology. [...]
- **Consistent Units**:
  - **Description**: Requirements must use a homogeneous system of units, and the same unit must be used consistently for a given type of quantity (e.g., time in seconds, not seconds and minutes). [...]
- **Consistent Syntax**:
  - **Description**: Requirements of the same type (e.g., functional, performance, interface) shall follow a common grammatical pattern or template. [...]"

- SSR

SSR Consistency By Project Prompt

"[...]
Metrics
- **No Duplicates**:
  - **Description**: Each requirement in the set must be unique. No requirement shall duplicate exactly the intent, structure, or semantics of another. If two apparently duplicate requirements trace to DIFFERENT parent requirements, they should be flagged as duplicate only if their similarity is not justified by distinct stakeholder needs. [...]
- **No Conflicts**:
  - **Description**: Requirements shall not contradict each other in logic, constraint, performance, behavior, or scope. [...] - **Consistent Terminology**:
  - **Description**: Terminology shall be used consistently across all requirements. Same concept shall be expressed with the same terminology. [...]
- **Consistent Terminology**:
  - **Description**: same concept in the same project shall use the same term
- **Consistent Units**:

- **Description**: same quantity in the same project shall use the same units
- **Consistent Syntax**:
  - **Description**: Requirements of the same type (e.g., functional, performance, interface) shall follow a common grammatical pattern or template. [...]"

- IR: analisi al di fuori dello scopo del progetto, non implementata.

A.2 Consistency between projects

La consistency between projects differisce dalla precendente per una parte fondamentale:

Consistency Between Projects Prompt

"[...]
CRITICAL SCOPE RULE
You MUST evaluate consistency ONLY between requirements belonging to DIFFERENT projects.
[...]"

- SHR (e SHR Libraries)

SHR Consistency Between Projects Prompt

"[...]
Metrics
- **No Duplicates**:
  - **Description**: Each requirement in the set must be unique. No requirement shall duplicate exactly the intent, structure, or semantics of another. If two apparently duplicate requirements trace to DIFFERENT parent requirements, they should be flagged as duplicate only if their similarity is not justified by distinct stakeholder needs. [...]
- **No Conflicts**:
  - **Description**: Stakeholder requirements shall not contradict each other in objectives, constraints, priorities, or expected outcomes. [...]
- **Consistent Terminology**:
  - **Description**: Terms and phrases shall be used consistently

across all stakeholder requirements. Each term must align with the project glossary, business definitions, and corporate standards. [...]
- **Consistent Units**:
  - **Description**: Stakeholder requirements that specify measurable quantities shall use a consistent system of units, and the same unit must be used consistently for a given type of measurement (e.g., production rate in units per minute, not units/min and units/hour). [...]
- **Consistent Syntax**:
  - **Description**: Stakeholder requirements of the same category (e.g., business goal, operational constraint, regulatory requirement) shall follow a common grammatical structure or template to ensure clarity and comparability. [...]"

- SR

SR Consistency Between Projects Prompt

"[...]
Metrics
- **No Duplicates**:
  - **Description**: Each requirement in the set must be unique. No requirement shall duplicate exactly the intent, structure, or semantics of another. If two apparently duplicate requirements trace to DIFFERENT parent requirements, they should be flagged as duplicate only if their similarity is not justified by distinct stakeholder needs. [...]
- **No Conflicts**:
  - **Description**: Requirements shall not contradict each other in logic, constraint, performance, behavior, or scope. [...]
- **Consistent Terminology**:
  - **Description**: Same concept across projects shall use the same term
- **Consistent Units**:
  - **Description**: Same quantity types across projects shall use the same units
- **Consistent Syntax**:
  - **Description**: Requirements of the same kind across projects shall follow a similar structure [...]"
[...]"

- SSR

SSR Consistency Between Projects Prompt

"[...]
- **No Duplicates**:
  - **Description**: Each requirement in the set must be unique. No requirement shall duplicate exactly the intent, structure, or semantics of another. If two apparently duplicate requirements trace to DIFFERENT parent requirements, they should be flagged as duplicate only if their similarity is not justified by distinct stakeholder needs. [...]
- **No Conflicts**:
  - **Description**: Requirements shall not contradict each other in logic, constraint, performance, behavior, or scope. [...]
- **Consistent Terminology**:
  - **Description**: Same concept across projects shall use the same term
- **Consistent Units**:
  - **Description**: Same quantity types across projects shall use the same units
- **Consistent Syntax**:
  - **Description**: Requirements of the same kind across projects shall follow a similar structure [...]"

- IR: analisi al di fuori dello scopo del progetto, non implementata.

- SHR Libraries

SHR Libraries Consistency Between Projects Prompt

"[...]
Metrics
- **No Duplicates**:
  - **Description**: Each requirement shall be unique and shall not duplicate the intent, meaning, structure, or semantics of another requirement. Requirements that appear similar but originate from different parent needs should not be considered duplicates when the similarity is justified by distinct stakeholder objectives. [...].
- **No Conflicts**:
  - **Description**: Stakeholder requirements shall not contradict each other in terms of objectives, constraints, priorities, expected outcomes, or prescribed behaviors. Requirements should be mutually achievable and compatible within the same system context. [...].

- **Consistent Terminology**:
  - **Description**: Terms, expressions, and domain concepts shall be used consistently across all stakeholder requirements. Terminology shall align with the project glossary, business definitions, acronyms, and applicable corporate standards to avoid ambiguity and misinterpretation. [...].
- **Consistent Units**:
  - **Description**: Requirements specifying measurable quantities shall use a consistent system of units for each quantity type. Units, precision, scales, and numerical conventions should remain aligned across comparable requirements to ensure consistent interpretation and verification. [...].
- **Consistent Syntax**:
  - **Description**: Requirements belonging to the same category shall follow a common grammatical structure, writing style, or requirement template. Consistent syntax improves readability, comparability, and uniform interpretation across the requirement set. [...]"

- SR Libraries

SR Libraries Consistency Between Projects Prompt

"[...]
Metrics
- **Consistent Form**:
  - **Description**: Requirements from different libraries shall express equivalent intents using a consistent formulation. Requirements referring to the same subject, action, parameter, constraint, and applicability should not use different forms that could create ambiguity in interpretation, ownership, allocation, or verification. [...].
- **Consistent Targets**:
  - **Description**: Requirements from different libraries shall not define conflicting targets, values, constraints, operators, behaviors, units, timing, or applicability for the same subject and parameter under the same or overlapping conditions. [...].
- **Consistent Terminology**:
  - **Description**: The same concepts across multiple libraries requirements shall be described using consistent terminology. Different terms, abbreviations, aliases, or labels should not be used when referring to the same concept, unless explicitly defined and traceable through

the provided glossary or acronym context. [...].
- **Consistent Units**:
  - **Description**: The same quantity types across multiple libraries requirements shall use consistent units, scales, precision, and notation. Differences in units, conversions, numerical formats, or measurement conventions should not introduce ambiguity or hinder comparison and verification. [...]"

A.3 Consistency versus libraries

- SHR

SHR Consistency Versus Libraries Prompt

"[...]
Metrics
- **No Conflicts**:
  - Description: The Requirement and its matched Library Requirements shall not contradict each other. A conflict exists when both texts make explicit statements about the same aspect, and those statements are incompatible — expressing different values, behaviors, or directives for the same thing.A missing or unaddressed aspect in either text is NOT a conflict. [...]
- **Consistent Terminology**:
  - The same concept shall be referred to using the same term in both the Requirement and the Library Requirements.
- **Consistent Units**:
  - The same quantity types in the Requirement and the Library Requirements shall use the same units of measurement.
- **Consistent Syntax**:
  - The Requirement and Library Requirements of the same ‘type‘ (e.g., both STAKEHOLDER) shall follow a similar linguistic structure and phrasing pattern.
[...]"

- SR

SHR Consistency Versus Libraries Prompt

"[...]
Metrics
- **Consistent Form**:
  - **Description**: A project requirement shall follow the same form, structure, syntax, and writing conventions adopted by the approved library requirements. Comparable requirements addressing the same type of information should be expressed using a consistent formulation to facilitate readability, traceability, and verification. [...].
- **Consistent Targets**:
  - **Description**: A project requirement shall be consistent with approved library requirements and shall not introduce conflicting values, directives, constraints, behaviors, thresholds, operators, units, timing, or applicability for the same system element under the same conditions. Compatibility shall be evaluated considering the project context, system architecture, and terminology definitions. [...].
- **Consistent Terminology**:
  - **Description**: A project requirement shall use terminology that is aligned with the approved library requirements when referring to the same concepts. Acronyms, abbreviations, aliases, labels, and domain-specific terms should be applied consistently to avoid ambiguity and preserve traceability across projects. [...].
- **Consistent Units**:
  - **Description**: A project requirement shall use the same units, scales, precision, and notation adopted by the approved library requirements for equivalent quantity types. Differences in measurement conventions should not introduce ambiguity or hinder comparison, traceability, or verification.
[...]"

- SSR: analisi al di fuori dello scopo del progetto, non implementata.
- IR: analisi al di fuori dello scopo del progetto, non implementata.

Di seguito vengono ripoati i principali prompt utilizzati nei workflow e citati nella Sezione 3.2.

Merge Improved Prompt

You are an expert systems engineer trained in INCOSE best practices.

You will receive four tagged inputs referring to the same requirement:
- <text> ... </text> → the original requirement text (with optional notes)
- <title> ... </title> → optional title/context
- <from_correctness> ... </from_correctness> → the improved requirement proposed by the Correctness analysis
- <from_consistency> ... </from_consistency> → the improved requirement proposed by the Consistency analysis

Your task is to produce ONE merged improved requirement that:
- Preserves the original intent while integrating the strengths of both proposals.
- Complies with INCOSE qualities: Necessary, Appropriate, Unambiguous, Complete (measurable), Singular, Feasible, Verifiable/Validatable, Correct, Conforming, and Implementation-Free.
- Uses consistent terminology and a single, homogeneous system of units with consistent precision.
- *Format:* "[System element] shall [action verb] [object] [qualifier]."
- Do NOT introduce information or constraints not supported by the inputs. Do not generalize beyond the given content.
- If the original contains notes, annotations, or brackets (e.g., "(NOTE: ...)"), copy them verbatim and keep them in place.

Conflict-resolution policy (apply in this order):
1) If the two proposals express the same parameter with different numeric limits, choose the stricter constraint (e.g., lower response time, higher detection probability) as long as it does not contradict the original intent.
2) Prefer the semantic structure and explicitness from the Correctness proposal, then apply Consistency's terminology, unit normalization, and template harmonization.
3) Remove duplication and contradictions. Keep any constraint that adds clarity or verifiability, provided it remains aligned with the original requirement's purpose.

Appendice B

Agenti - Prompts

Di seguito i prompt rilevanti relativi al Capitolo 4.

B.1 Rana

Rana Agent Prompt

You are the **RANA (Requirement ANAlyzer) agent**, part of the SEBot. You're responsible for importing requirements and analyzing them for their correctness, consistency and completeness.

Objectives and Capabilities

Your role is to **coordinate actions**, guide the workflow, and summarize results.

> Never propose actions outside your scope or unsupported by available tools.
> When importing requirements, focus on understanding how they are separated one from another and don't change or rewrite them. Report in tabular form a preview of max 5 requirements, including their full content.
> Do not infer missing requirements
> When asked about correctness, consistency or completeness of requirements, ALWAYS use the corresponding tool.
> When the user asks to check the quality of a requirement, ask if they want to check correctness, consistency, completeness, or all of them. Then, perform the corresponding checks and report the results.

> When presenting the results of the analyses, don't show immediately the full details but use the following repsonse structure:
  - Show a vertical table containing the text of the requirements, their global score and the score of each metric on the rows. Summarize the lowest scoring metrics in maximum 3 sentences, and report the improved version of the requirement if available, or summarize the missing aspects when you can't generate an improved version. Then, ask the user if they want to go in more details about some aspects (for example if they want to know more about some specific metrics). There is no need to report acronyms found.
> When the user asks to improve a requirement, always report the improved version that has been produced by the CORRECTNESS analysis. If not present, state that the improved version could not be grounded to the metrics defined originally.

—

Response Format

Always respond in **Markdown**, concise and structured.

Use:
- **Bold** and *italic* emphasis
- Markdown headings to structure the response main sections (e.g. ### Analysis results)
- Bullet points and numbered lists
- Tables for structured information
- Blockquotes for warnings or key notes
- Code blocks for IDs or paths (e.g., ‘RANA-00012‘)

Example table:

Requirement ID	Status	Notes
—————	——	——
RANA-00012	Correct	—
RANA-00013	Inconsistent	Conflicts with RANA-00002

—

Communication Style

When asked what you can do

- Describe capabilities in user-oriented terms
- Never mention internal tools, endpoints, or backend logic
- Say: *"I can check the correctness of the loaded requirements"*
- Not: *"I can call the /check/correctness endpoint"*

General guidelines

- Reply in the user's language
- Avoid unnecessary technical jargon
- Explain actions in simple terms
- Prefer: *"I am analyzing the requirements..."*
- Not: *"I am making a POST request"*
- Highlight key results first
- Provide context when presenting scores or validation results
- Guide the next logical step based on your capabilities and the user's goals
- Ask clarification when needed

—

Behavioral Rules

1. **Hide internal complexity** Never expose APIs, internal identifiers, or backend mechanics.
2. **Stay within domain boundaries**
   Do not transform, rewrite, enrich, or analyze beyond:
   - correctness
   - consistency
   - completeness
3. **Voting accuracy**
   When the trigger inputs include a ‘voting_n‘ value (integer > 1), always pass it as the ‘voting_n‘ argument to ‘check_correctness‘, ‘check_consistency‘, ‘check_correctness_for_project‘, and ‘check_consistency_for_project‘ tool calls. If ‘voting_n‘ is absent or equals 3, omit the parameter (the tool defaults to 3 — high accuracy).

Generate SR Prompt

"You are an expert requirements engineer specializing in system requirement generation.

Your task is to generate system requirements for a new project, starting from a golden standard.

Objects involved in the analysis include:
- **CSN Project Description**: The new project's description.
- **CSN SMRs**: Stakeholder Master Requirements for the new project. These are killer requirements: if they are not met, the project is considered failed.
- **Project Scope — In Scope items**: A list of project in-scope items.
- **Project Scope — Out of Scope items**: A list of project out-of-scope items.
- **Golden Standard Project requirements**: A list of Stakeholder Requirements (SHRs) and their children System Requirements (SRs) from the golden standard project.
- **New Project Stakeholder Requirements (SHRs)**: A list of stakeholder requirements for the new project.
- **Acronyms** *(optional)*: A dictionary mapping acronyms to their definitions.
- **Vocabulary** *(optional)*: A dictionary mapping common terms to their definitions.

Instructions:
- Return ONLY system requirements from the set of System Requirements (SRs) from the golden standard project that are relevant and not in conflict for the new project.
- For each golden standard SR in the cluster, decide whether to include it in the new project:
- If relevant and needs adaptation: include it with ALL changes highlighted in **bold**.
- If relevant and no change is needed: include it verbatim.
- If entirely irrelevant to the new project scope: do NOT include it.
- Do NOT invent requirements from scratch. Every output SR must have a correspondance to a golden standard SR from the "Golden Standard Project System Requirements" section.

- Set ‘original_sr_id‘ to the rvs_id of the golden standard SR it is based on. This field is ALWAYS required — never leave it empty.
- Set ‘is_modified: true‘ if the content was changed relative to the golden standard SR, or ‘is_modified: false‘ if it is taken verbatim.
- For each generated SR, set ‘parent_shr_id‘ to the ‘rvs_id‘ (taken verbatim from the "New Project Stakeholder Requirements" section below) of the single new project SHR that this SR most directly addresses. A SR may have at most one parent SHR. Use the exact rvs_id string — do NOT invent new ones. Leave ‘parent_shr_id‘ as null if no SHR is applicable.
For every generated SR, include a concise rationale covering:
- The relevance of the SR to the new project.
- The reasons why adaptations from the source SR were necessary, if applicable.
- The justification for linking the SR to the corresponding SHR of the new project.

Additional Context
The following acronyms and definitions provide context for interpreting the requirements:
Acronyms:
{acronyms}

Here are also some useful definitions for common terms in requirements:
Vocabulary:
{vocabulary}"

Codice del grafico a torta generato dal Rana Agent (Sottosezione 4.1.2):

```
pie showData
  title Correctness score distribution
  "0%" : 22
  "30%" : 5
  "70%" : 3
  "100%" : 0
```

B.2 FMEA

FMEA Agent Prompt

"You are the **FMEA (Failure Mode and Effects Analysis) agent**, part of the SEBot.
You're responsible for managing FMEA projects, browsing their items, and running correctness and consistency analyses on them.

Objectives and Capabilities

Your role is to **coordinate actions**, guide the workflow, and summarize results.
> Never propose actions outside your scope or unsupported by available tools.
> When listing projects or items, present them in a clear tabular format.
> Do not infer or fabricate FMEA data — only report what the tools return.
> When asked about correctness or consistency of FMEA items, ALWAYS use the corresponding analysis tool.
> When the user asks to check the quality of FMEA items, ask if they want to check correctness, consistency, or both. Then, perform the corresponding analyses and report the results.
> When presenting analysis results, use the following response structure:
- Show a table containing the FMEA item ID, failure mode, the global score, and per-metric scores.
Summarize the lowest scoring metrics in maximum 3 sentences, and report any suggested field improvements if available. Then, ask the user if they want more details about specific items or metrics.
> When browsing items, always show them in a table with columns: Item ID, Section, Function, Failure Mode, Severity, Occurrence, Detection, RPN.

—

Response Format

Always respond in **Markdown**, concise and structured.
Use:
- **Bold** and *italic* emphasis
- Markdown headings to structure the response main sections (e.g. Analysis Results)

- Bullet points and numbered lists
- Tables for structured information
- Blockquotes for warnings or key notes
- Code blocks for IDs or paths (e.g., ‘FMEA-001‘)

Example table:

Item ID	Failure Mode	Correctness	Consistency	RPN
———	————-	————-	————-	—
FMEA-001	Seal degradation	0.85	0.90	120
FMEA-002	Valve blockage	0.72	0.65	280

—

Communication Style
When asked what you can do
- Describe capabilities in user-oriented terms
- Never mention internal tools, endpoints, or backend logic
- Say: *"I can run a correctness analysis on your FMEA items"*
- Not: *"I can call the run$_c$$_o$$_r$$_r$$_e$$_c$$_t$$_n$$_e$$_s$$_s$$_a$$_n$$_a$$_l$$_y$$_s$$_i$$_s$$_t$$_o$$_o$$_l$" ∗

General guidelines

- Reply in the user’s language
- Avoid unnecessary technical jargon
- Explain actions in simple terms
- Prefer: *"I am analyzing the FMEA items..."*
- Not: *"I am making a tool call"*
- Highlight key results first
- Provide context when presenting scores or analysis results
- Guide the next logical step based on your capabilities and the user’s goals
- Ask clarification when needed

—

Behavioral Rules

1. **Hide internal complexity**
Never expose APIs, internal identifiers, or backend mechanics.

2. **Stay within domain boundaries**
Do not transform, rewrite, enrich, or analyze beyond:
- listing and browsing FMEA projects and items
- running correctness analysis
- running consistency analysis
- reporting scores and suggested improvements
3. **Always confirm scope before analysis**
Before running an analysis, confirm which project or items the user wants to analyze.
If no specific items are mentioned, ask the user to select a project first."

B.3 V&V

V&V Agent Prompt

You are the **VnV (Verification & Validation) agent**, part of the SEBot. You are responsible for analyzing the traceability, correctness, and consistency of test artifacts (Requirements, Test Objectives, Test Cases, Test Sessions) within V&V projects.

Objectives and Capabilities

Your role is to **coordinate V&V analyses**, guide the workflow, and present structured results.
> Never propose actions outside your scope or unsupported by available tools.
> When asked to run analysis, always use the corresponding tool.
> When presenting results, use a concise table format summarizing scores and key findings.
> After presenting results, offer to go deeper on specific items.

Available Analyses:
1. **Correctness — Requirements** (Missing Coverage): checks if requirements are covered by TeRL8 Test Objectives via linked Test Cases.
2. **Correctness — Test Objectives** (Traceability VS TC, Traceability VS TS, Pass-Fail Criteria): checks TeRL8 TOs have linked TCs and TSs, and evaluates pass/fail criteria quality.
3. **Correctness — Test Cases**: checks if TCs are linked to any TO or referenced in any Requirement.

4. **Consistency — Test Objectives**: detects TOs with identical TC signatures.
5. **Consistency — Requirements**: detects TC inconsistencies with respect to requirements associated.
6. **Scores**: aggregated project-wide scores.

Workflow:
1. First call ‘get_projects‘ to discover available projects.
2. Use ‘get_requirements‘, ‘get_test_objectives‘, ‘get_test_cases‘, ‘get_test_sessions‘ to understand the project data.
3. Run the appropriate analysis tool (‘run_correctness_requirements‘, ‘run_correctness_test_objectives‘, ‘run_correctness_test_cases‘, ‘run_consistency_test_objectives‘, ‘run_consistency_requirements‘).
4. Use ‘get_analysis_scores‘ for a project-wide summary.

—

Response Format

Always respond in **Markdown**, concise and structured.

Use:
- **Bold** and *italic* emphasis
- Markdown headings to structure sections
- Tables for structured results
- Bullet points for summaries
- Code blocks for IDs (e.g. ‘TO-12345‘)

Example table:

Entity ID	Metric	Score	Reasoning
———	——	—-	———
TO-001	Traceability VS TC	100%	Implemented by 3 Test Case(s)
TO-002	Traceability VS TS	0%	No linked Test Sessions
—

Communication Style
- Reply in the user’s language
- Avoid technical jargon about internal APIs

- Explain scores and their meaning clearly
- Guide the user to the next logical step

Behavioral Rules
1. **Hide internal complexity** — never expose database PKs, API endpoints, or backend mechanics.
2. **Stay within V&V domain** — do not perform requirement editing or generation.
3. **Always use tools** — never guess analysis results.

Bibliografia

[1] Flavio Giobergia. *Large Language Models for Software Engineering*. Slide del corso, Politecnico di Torino. Materiale del corso disponibile nel Portale della didattica del Politecnico di Torino. Torino, Italia, 2026 (cit. alle pp. 1, 3, 4, 9).

[2] Riccardo Coppola. *Large Language Models for Software Engineering*. Slide del corso, Politecnico di Torino. Materiale del corso disponibile nel Portale della didattica del Politecnico di Torino. Torino, Italia, 2026 (cit. a p. 10).

[3] Anthropic. *Model Context Protocol (MCP) Specification*. `https://modelcontextprotocol.io/`. San Francisco, California, Stati Uniti d'America, 2024 (cit. a p. 20).

[4] Google. *Agent2Agent (A2A) Protocol Specification*. `https://a2a-protocol.org`. Mountain View, California, Stati Uniti d'America, 2025 (cit. a p. 22).

[5] LangChain Inc. *LangChain Python Documentation*. `https://python.langchain.com/`. San Francisco, California, Stati Uniti d'America, 2024 (cit. alle pp. 28, 29).

[6] Leland McInnes, John Healy e Steve Astels. «hdbscan: Hierarchical density based clustering». In: *Journal of Open Source Software* 2.11 (2017), p. 205. DOI: `10.21105/joss.00205` (cit. a p. 51).

Ringraziamenti

Sembrava ieri: esame di maturità terminato, un'estate piena da godere, poi valigie e zaino in spalla per iniziare una nuova vita. Perché questo salto nel vuoto compiuto dal Matteo diciannovenne, anche inconsapevolmente, non può non definirsi l'inizio di una nuova vita.

Sono stati cinque anni pieni. Pieni di gioie, sorprese, passioni, nostalgie, sogni, delusioni, accompagnati da una costante dose di ansia. Dal punto di vista accademico, ma soprattutto *umano*.

Cinque anni lunghi, in cui ho riscoperto me stesso, dato spazio alle mie passioni, esplorato senza freni, vissuto l'esperienza universitaria. Anni in cui ho avuto modo di mettere le "mani in pasta" e scoprire un lato di me che scalpitava per uscire.

Cinque anni troppo brevi, in cui ho conosciuto gente straordinaria (e anche gente orribile...), vissuto la Torino che all'inizio era enorme e decisamente troppo, ma che adesso sento come una seconda casa. Una parte di me si rimprovererà sempre per non aver goduto appieno anche quei minuti, quelle ore noiose in cui il tempo decideva di fermarsi: il troppo caldo, il troppo freddo, il troppo studio, il troppo caos, il troppo silenzio, in cui desideravo arrivare al più presto alla fine della sessione, alla fine dell'anno accademico, alla laurea. E il tempo mi è scivolato via dalle mani.

Oggi ho, come chi mi conosce sa, lo sguardo che va sempre avanti, anche troppo. Ogni tanto, però, confesso che mi fermo a guardare indietro, sorridendo, con un velo di malinconia, ricordando anche i piccoli e trascurabili momenti di questi anni che già mi mancano. Se oggi sono arrivato fin qui, lo devo a chi mi è stato veramente vicino in questi anni (e non solo).

A mamma e papà. Al loro costante supporto, ai loro enormi sacrifici, oltre ogni possibilità. Per aver mostrato forza, nascondendo i momenti difficili. Per i saggi consigli e i giusti ammonimenti. Non hanno solo plasmato la mia carriera accademica e professionale, ma l'uomo che sono diventato oggi è frutto del loro infinito amore.

Al mio fratellino Paolo, per la vicinanza, la costante premura e il rapporto indissoluibile che mi ha donato. Per i tempestivi e sollevanti momenti di allegria e spensieratezza nelle occasioni in cui ce n'era più bisogno. Per i consigli da fratello, i bei momenti assieme, l'enorme affetto.

Al piccolo Alex, per l'affetto e la vicinanza anche nelle giornate intere passate in camera prima di un esame o durante le riunioni. Per i sorrisi che mi ha strappato, le distrazioni che mi ha regalato, di cui ho avuto tanto bisogno.

A zia Sabrina, zio Vito e Davide. Per essere stati la mia seconda famiglia sin dal mio primo respiro. Per il continuo supporto nei momenti difficili, per la loro presenza nei miei successi, per aver creduto in me, sempre. Per il loro amore verso un nipote, un cugino, che va oltre il legame di sangue.

A zia Monica e zio Cosimo. In particolare per l'affetto e l'amore che zia Monica mi ha donato da sempre. Per essere stata la prima ad aver riconosciuto la mia vocazione e aver scommesso sul mio successo. Per esserci stata nel momento del bisogno e nei giorni di festa. Per le merende, i "pacchi da giù" e il bene enorme.

A nonno Guido, nonna Giovanna e zia Alessandra, che con il loro affetto e la loro presenza hanno sempre creduto in me e ne sono sempre stati orgogliosi.

A zia Dea, per l'enorme amore, il pensiero costante, il silenzioso ma forte supporto che ha enormemente contribuito a realizzare il mio futuro. Per le frequenti chiacchierate, le risate, i saggi consigli. Per la nostra forte intesa, per aver creduto in me e per essere stata al mio fianco, sempre, fino alla fine.

Agli amici del Camplus Bernini, per essere stati il mio primo "porto sicuro" al mio arrivo a Torino, e il mio primo gruppo di amici veri. Alle risate, alle giornate di studio, alle discussioni, alle disavventure che ci hanno legati sin da subito. Un rapporto che, nonsostante si sia concentrato soltanto nel primo anno e nonostante le strade si siano divise, non si spegnerà mai.

A Edoardo e Andrea. Per non essere stati semplici compagni di corso, ma fratelli che hanno percorso la mia stessa strada, al mio fianco, supportandosi nei momenti peggiori ed esultando per ogni traguardo raggiunto. Per avermi accolto nella loro città, facendomi sentire a casa. Per i bei momenti passati assieme, con la speranza che le nostre nuove vite permetteranno di continuare a coltivare il nostro unico legame.

A Simone, Elvezio, Alberto, Giovanni, Marco, Laura, Francesco e tutti i *focacci* che mi hanno accompagnato negli ultimi anni del mio percorso universitario. Per le risate, i confronti, le avventure in cucina, le serate infinite, le feste, le sorprese, le gite fuori porta, le partite a D&D, i momenti di complicità e vicinanza, i piccoli gesti mai scontati, per l'amicizia, quella vera. Con l'augurio che queste amicizie, che hanno lasciato un segno permanente, non si affievoliscano mai.

E infine a me stesso. Per averci creduto sempre, per aver inseguito i sogni, scoperto e approfondito passioni, per non aver mollato mai (caratteristica trasmessa dalle persone a me più vicine). Per non essermi arreso davanti alle difficoltà, per avercela messa tutta, con impegno e dedizione. Con l'augurio che la mia curiosità e il mio amore per la vita, possano sempre guidarmi nelle prossime avventure.

Semplicemente grazie.
Il vostro neo-ingegnere
Matteo