Contents

Flussi di Dati: Pattern di Interazione e Sequenze

Panoramica

Questo documento descrive come i dati fluiscono attraverso l'architettura durante diverse operazioni, mostrando le interazioni tra componenti e le trasformazioni dei dati.

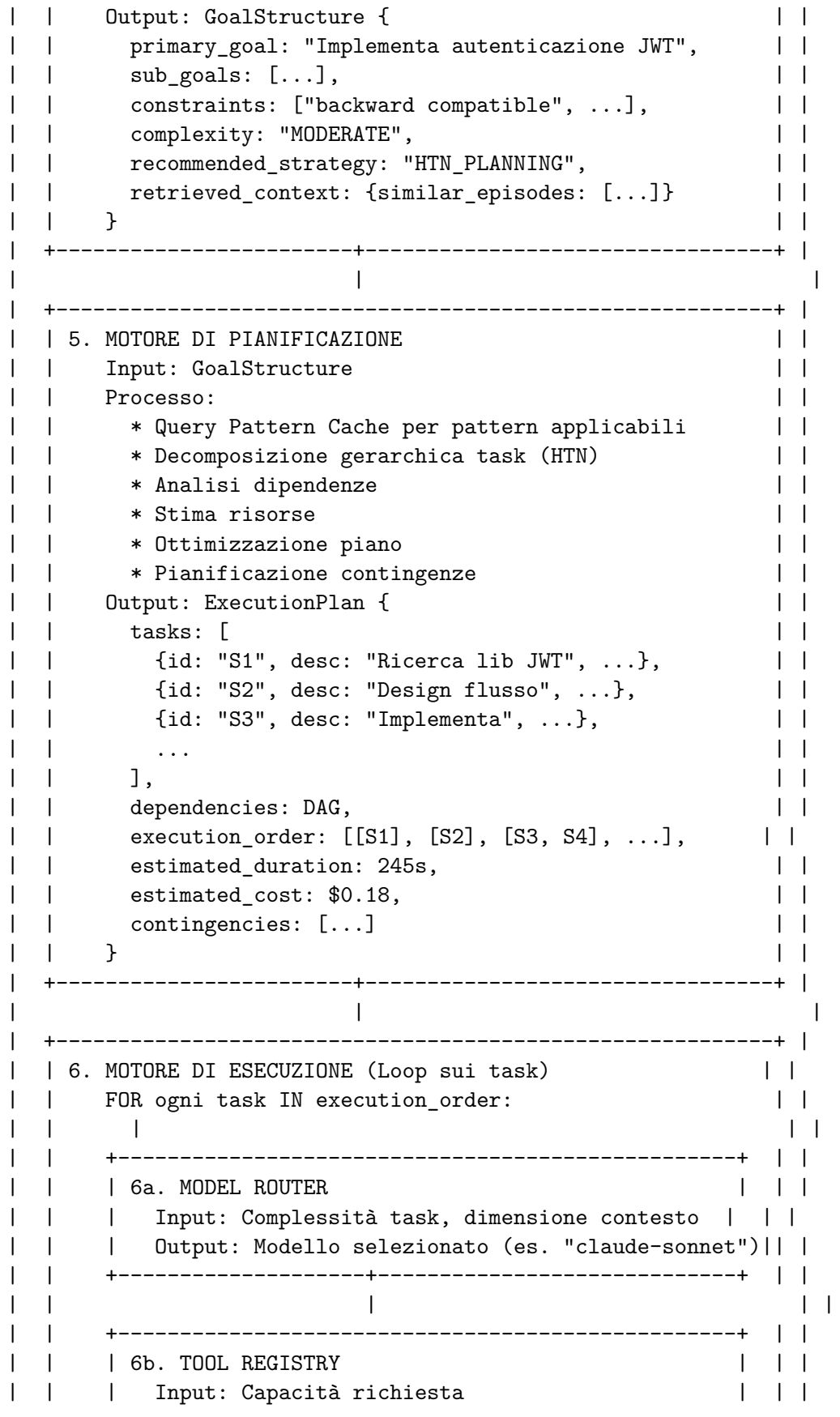
1. Flusso Completo di Esecuzione dei Task

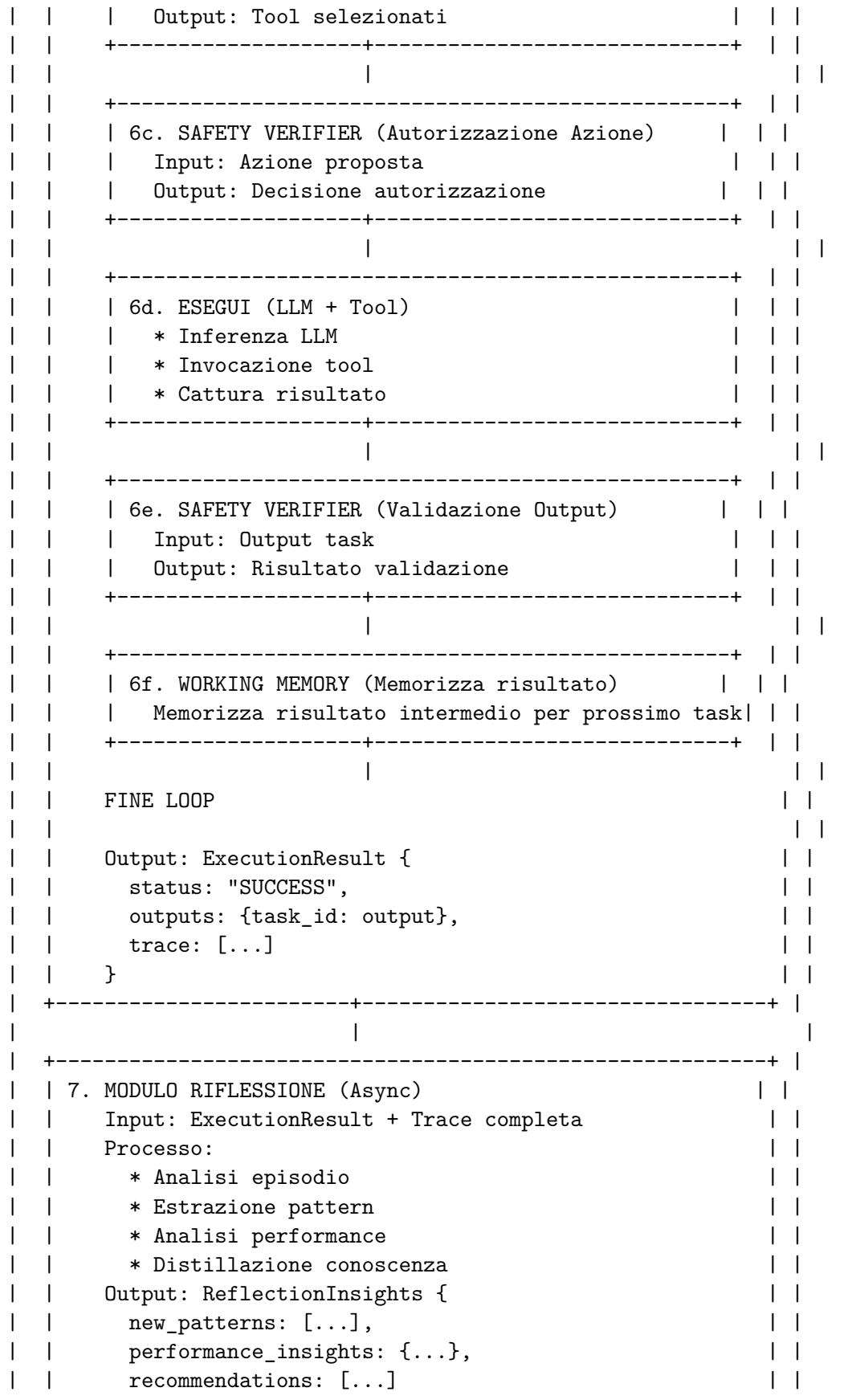
1.1 Flusso di Dati End-to-End

Percorso completo dalla richiesta utente al risultato finale:

```
+------------------------------------------------------------------+
|          FLUSSO DATI ESECUZIONE TASK COMPLETA                    |
|                                                                  |
|  UTENTE                                                          |
|    |                                                             |
|    | Task: "Aggiungi autenticazione JWT all'API"                 |
|    |                                                             |
|  +-------------------------------------------------------------+ |
|  | 1. API GATEWAY                                              | |
```

```
|  |    Input: Task in linguaggio naturale                    | |
|  |    Output: TaskRequest {                                 | |
|  |      task_id: "T-12345",                                 | |
|  |      user_id: "U-789",                                   | |
|  |      input_text: "...",                                  | |
|  |      timestamp: "2024-01-15T10:00:00Z",                  | |
|  |      context: {session_id, preferences, ...}             | |
|  |    }                                                     | |
|  +-----------------------+----------------------------------+ |
|                          |                                    |
|  +-------------------------------------------------------+ |
|  | 2. SAFETY VERIFIER (Validazione Input)                | | |
|  |    Input: TaskRequest                                  | |
|  |    Processo:                                           | |
|  |      * Rilevamento injection                            | |
|  |      * Validazione schema                              | |
|  |      * Filtraggio contenuti                            | |
|  |    Output: ValidationResult {                          | |
|  |      valid: true,                                      | |
|  |      sanitized_input: "...",                           | |
|  |      detected_issues: []                               | |
|  |    }                                                   | |
|  +-----------------------+----------------------------------+ |
|                          |                                    |
|  +-------------------------------------------------------+ |
|  | 3. RESOURCE MANAGER (Controllo Budget)                | | |
|  |    Input: TaskRequest + Budget Utente                  | |
|  |    Processo:                                           | |
|  |      * Controlla budget rimanente                      | |
|  |      * Stima costo task                                | |
|  |      * Alloca risorse                                  | |
|  |    Output: ResourceAllocation {                        | |
|  |      approved: true,                                   | |
|  |      allocated_budget: $1.00,                          | |
|  |      time_limit: 600s,                                 | |
|  |      priority: "normal"                                | |
|  |    }                                                   | |
|  +-----------------------+----------------------------------+ |
|                          |                                    |
|  +-------------------------------------------------------+ |
|  | 4. MODULO ANALISI GOAL                                 | |
|  |    Input: Task sanitizzato + Contesto                  | |
|  |    Processo:                                           | |
|  |      * Parsing semantico                               | |
|  |      * Estrazione goal                                 | |
|  |      * Identificazione vincoli                         | |
|  |      * Recupero contesto da Memoria Episodica          | |
|  |      * Classificazione complessità                     | |
```


Output: GoalStructure {
primary_goal: "Implementa autenticazione JWT",
sub_goals: [...],
constraints: ["backward compatible", ...],
complexity: "MODERATE",
recommended_strategy: "HTN_PLANNING",
retrieved_context: {similar_episodes: [...]}
}
5. MOTORE DI PIANIFICAZIONE
Input: GoalStructure
Processo:
* Query Pattern Cache per pattern applicabili
* Decomposizione gerarchica task (HTN)
* Analisi dipendenze
* Stima risorse
* Ottimizzazione piano
* Pianificazione contingenze
Output: ExecutionPlan {
tasks: [
{id: "S1", desc: "Ricerca lib JWT", ...},
{id: "S2", desc: "Design flusso", ...},
{id: "S3", desc: "Implementa", ...},
...
],
dependencies: DAG,
execution_order: [[S1], [S2], [S3, S4], ...],
estimated_duration: 245s,
estimated_cost: $0.18,
contingencies: [...]
}
6. MOTORE DI ESECUZIONE (Loop sui task)
FOR ogni task IN execution_order:
6a. MODEL ROUTER
Input: Complessità task, dimensione contesto
Output: Modello selezionato (es. "claude-sonnet")
6b. TOOL REGISTRY
Input: Capacità richiesta


Output: Tool selezionati
6c. SAFETY VERIFIER (Autorizzazione Azione)
Input: Azione proposta
Output: Decisione autorizzazione
6d. ESEGUI (LLM + Tool)
* Inferenza LLM
* Invocazione tool
* Cattura risultato
6e. SAFETY VERIFIER (Validazione Output)
Input: Output task
Output: Risultato validazione
6f. WORKING MEMORY (Memorizza risultato)
Memorizza risultato intermedio per prossimo task
FINE LOOP
Output: ExecutionResult {
status: "SUCCESS",
outputs: {task_id: output},
trace: [...]
}
7. MODULO RIFLESSIONE (Async)
Input: ExecutionResult + Trace completa
Processo:
* Analisi episodio
* Estrazione pattern
* Analisi performance
* Distillazione conoscenza
Output: ReflectionInsights {
new_patterns: [...],
performance_insights: {...},
recommendations: [...]

```
|  |    }                                                    | |
|  |    Effetti Collaterali:                                 | |
|  |      * Memorizza episodio in Memoria Episodica          | |
|  |      * Aggiorna Pattern Cache                           | |
|  +-------------------------+-------------------------------+ |
|                            |                                   |
|  +-------------------------+-------------------------------+ |
|  | 8. RESOURCE MANAGER (Registra uso effettivo)           | |
|  |    Aggiorna budget con costo effettivo                  | |
|  +-------------------------+-------------------------------+ |
|                            |                                   |
|  +-------------------------+-------------------------------+ |
|  | 9. SISTEMA OSSERVABILITÀ (Logging & Metriche)          | |
|  |    * Log trace esecuzione completa                      | |
|  |    * Aggiorna metriche (tasso successo, latenza, costo) | |
|  |    * Memorizza trace distribuita                        | |
|  +-------------------------+-------------------------------+ |
|                            |                                   |
|  UTENTE                                                       |
|    <- FinalResult {                                           |
|        success: true,                                         |
|        output: "Autenticazione JWT implementata",             |
|        cost: $0.19,                                           |
|        duration: 265s                                         |
|      }                                                        |
+---------------------------------------------------------------+
```

1.2 Trasformazioni dei Dati

Trasformazioni chiave attraverso il flusso:

```
+-----------------------------------------------------+
|           STADI DI TRASFORMAZIONE DATI              |
|                                                     |
|  Stadio 1: LINGUAGGIO NATURALE                      |
|  "Aggiungi autenticazione JWT all'API"              |
|    | (Analisi Goal)                                 |
|                                                     |
|  Stadio 2: GOAL STRUTTURATO                         |
|  GoalStructure {                                    |
|    primary_goal: {...},                             |
|    constraints: [...],                              |
|    complexity: "MODERATE"                           |
|  }                                                  |
|    | (Pianificazione)                               |
|                                                     |
|  Stadio 3: PIANO DI ESECUZIONE                      |
|  ExecutionPlan {                                    |
```

```
|    tasks: [T1, T2, ...],                                    |
|    dependencies: DAG,                                       |
|    execution_order: [[T1], [T2, T3], ...]                   |
|  }                                                          |
|    | (Esecuzione)                                           |
|                                                             |
|  Stadio 4: RISULTATI INTERMEDI                              |
|  {                                                          |
|    T1_output: "PyJWT scelto",                               |
|    T2_output: "Doc design creato",                          |
|    T3_output: "Codice implementato",                        |
|    ...                                                      |
|  }                                                          |
|    | (Aggregazione)                                         |
|                                                             |
|  Stadio 5: OUTPUT FINALE                                    |
|  ExecutionResult {                                          |
|    status: "SUCCESS",                                       |
|    primary_output: "Sistema autenticazione con JWT",        |
|    artifacts: [file_codice, test, docs],                    |
|    metadata: {duration, cost, ...}                          |
|  }                                                          |
|    | (Riflessione - Async)                                  |
|                                                             |
|  Stadio 6: ARTEFATTI DI APPRENDIMENTO                       |
|  ReflectionInsights {                                       |
|    new_patterns: ["JWT-implementation-pattern"],            |
|    performance_insights: {...},                             |
|    stored_episode: episode_id                               |
|  }                                                          |
+-------------------------------------------------------------+
```

2. Interazioni con il Sistema di Memoria

2.1 Pattern di Lettura Memoria

Pattern: Recupero Contesto per Pianificazione

```
+-------------------------------------------------------------------+
|         RECUPERO MEMORIA PER PIANIFICAZIONE                       |
|                                                                   |
|  Il Motore di Pianificazione necessita di contesto                |
|    |                                                              |
|    +---> Query MEMORIA EPISODICA                                   |
|    |    Input: Embedding descrizione task corrente               |
|    |    Processo:                                                |
|    |      * Ricerca similarità vettoriale                        |
|    |      * Filtra per punteggio rilevanza > 0.7                 |
```

```
|    |      * Classifica per recenza e successo                  |
|    |    Output: Top-5 episodi simili [E1, E2, E3, E4, E5]      |
|    |      Ognuno contiene:                                     |
|    |        - Descrizione task                                 |
|    |        - Strategia usata                                  |
|    |        - Risultato                                        |
|    |        - Apprendimenti chiave                             |
|    |    Latenza: ~80ms                                         |
|    |                                                            |
|    +---> Query PATTERN CACHE                                    |
|    |    Input: Feature task {domain: "auth", type: "impl"}     |
|    |    Processo:                                              |
|    |      * Lookup basato su feature                           |
|    |      * Filtra pattern applicabili                         |
|    |      * Classifica per confidenza                          |
|    |    Output: Top-3 pattern [P1, P2, P3]                     |
|    |      Ognuno contiene:                                     |
|    |        - Template pattern                                 |
|    |        - Guida applicazione                               |
|    |        - Benefici attesi                                  |
|    |    Latenza: ~20ms (cached)                                |
|    |                                                            |
|    +---> Carica in WORKING MEMORY                               |
|         Contesto combinato: {episodes: [...], patterns: [...]} |
|         Usato dal Motore di Pianificazione per decisioni       |
|         informate                                              |
+----------------------------------------------------------------+
```

2.2 Pattern di Scrittura Memoria

Pattern: Memorizzazione Episodio Dopo Completamento

```
+----------------------------------------------------------------+
|            FLUSSO MEMORIZZAZIONE EPISODIO                      |
|                                                                |
|  Task completato -> Inizia riflessione                          |
|    |                                                           |
|    +---> Costruisci Oggetto Episodio                            |
|    |    Da: ExecutionResult + Trace + Contesto                 |
|    |    Output: Struttura Episodio completa                    |
|    |                                                            |
|    +---> Genera Embedding                                       |
|    |    * Descrizione task -> embedding (vettore 768-dim)      |
|    |    * Episodio completo -> embedding                       |
|    |    * Risultato -> embedding                               |
|    |    Usando: Modello embedding (es. text-embedding-3)       |
|    |                                                            |
|    +---> Memorizza in MEMORIA EPISODICA                         |
```

```
|    |    Scritture parallele:                                    |
|    |      * Vector DB <- embeddings + episode_id                |
|    |      * Document DB <- documento episodio completo          |
|    |      * Object Storage <- artefatti grandi (se presenti)    |
|    |    Transazione: Atomica (tutto o niente)                   |
|    |    Latenza: ~150ms                                         |
|    |                                                            |
|    +---> Aggiorna Indici                                         |
|         * Aggiungi a indici di ricerca                           |
|         * Aggiorna statistiche (conteggio episodi, successo      |
|           medio, ecc)                                           |
|         * Invalida cache rilevanti                               |
|                                                                  |
|  Episodio ora disponibile per futuri recuperi                    |
+------------------------------------------------------------------+
```

2.3 Aggiornamento Pattern Cache

Pattern: Aggiornamento Pattern da Riflessione

```
+------------------------------------------------------------------+
|            FLUSSO AGGIORNAMENTO PATTERN CACHE                    |
|                                                                  |
|  La Riflessione scopre nuovo pattern                             |
|    |                                                             |
|    +---> Controlla se pattern esiste già                         |
|    |    Query Pattern Cache per similarità                       |
|    |    SE esiste: Aggiorna evidenza                             |
|    |    SE non esiste: Crea nuovo pattern candidato              |
|    |                                                             |
|    +---> SE NUOVO PATTERN:                                       |
|    |    |                                                        |
|    |    +--> Crea Oggetto Pattern                                |
|    |    |   Status: CANDIDATE                                    |
|    |    |   Evidence: [episodio corrente]                        |
|    |    |   Confidence: Bassa (singolo episodio)                 |
|    |    |                                                        |
|    |    +--> Memorizza in Pattern Cache                          |
|    |        Monitora per ulteriore evidenza                      |
|    |                                                             |
|    +---> SE PATTERN ESISTENTE:                                   |
|         |                                                        |
|         +--> Aggiungi Episodio all'Evidenza                      |
|         |   pattern.evidence.supporting_episodes.append(...)     |
|         |                                                        |
|         +--> Ricalcola Statistiche                               |
|         |   * Tasso di successo                                  |
|         |   * Significatività statistica                        |
```

```
|         |   * Punteggio confidenza                                  |
|         |                                                           |
|         +--> Controlla per Promozione                               |
|         |   SE confidenza > soglia AND significatività < 0.05:|
|         |     pattern.status = "VALIDATED"                          |
|         |                                                           |
|         +--> Aggiorna in Cache                                      |
|             Persisti modifiche                                      |
|                                                                     |
|  Pattern ora ha più evidenza, possibilmente validato                |
+---------------------------------------------------------------------+
```

3. Interazioni con Model Router

3.1 Flusso di Selezione Modello

```
+---------------------------------------------------------------------+
|                FLUSSO SELEZIONE MODELLO                             |
|                                                                     |
|  Motore di Esecuzione: Necessita LLM per subtask                    |
|    |                                                                |
|    +---> Prepara Richiesta al Model Router                          |
|    |    ModelRequest {                                             |
|    |      task_description: "...",                                 |
|    |      complexity: "MODERATE",                                  |
|    |      context_size: 12K token,                                 |
|    |      requires_creativity: false,                              |
|    |      budget_remaining: $0.50                                  |
|    |    }                                                          |
|    |                                                               |
|    |                                                               |
|  +-------------------------------------------------------------+ |
|  | MODEL ROUTER                                                | |
|  |                                                             | |
|  |  1. Classificazione Complessità                             | |
|  |     Punteggio task -> complexity_level = MODERATE           | |
|  |                                                             | |
|  |  2. Controllo Finestra Contesto                             | |
|  |     12K token -> Si adatta a Tier 2 (fino a 128K)           | |
|  |                                                             | |
|  |  3. Applica Policy di Routing                               | |
|  |     SE complessità == MODERATE:                             | |
|  |       Default -> Tier 2 (Modello medio)                     | |
|  |                                                             | |
|  |  4. Controllo Costi                                         | |
|  |     Costo stimato: $0.08                                    | |
|  |     Budget rimanente: $0.50                                 | |
|  |     [v] Approvato                                           | |
```

```
|  |                                                          | |
|  |  5. Selezione Modello                                    | |
|  |     Selezionato: "claude-sonnet-3.5"                     | |
|  |     Catena fallback: ["gpt-4o-mini", "gpt-4-turbo"]     | |
|  |                                                          | |
|  +------------------------+---------------------------------+ |
|                           |                                   |
|  +--------------------------------------------------------+ |
|  | LIVELLO INVOCAZIONE MODELLO                            | |
|  |                                                        | |
|  |  1. Bilanciamento Carico                               | |
|  |     Seleziona istanza: claude-sonnet-instance-3        | |
|  |                                                        | |
|  |  2. Chiamata API                                       | |
|  |     POST /v1/chat/completions                          | |
|  |     Body: {model, messages, max_tokens, ...}           | |
|  |                                                        | |
|  |  3. Gestione Risposta                                  | |
|  |     Analizza risposta                                  | |
|  |     Estrai: text, tokens_used, finish_reason           | |
|  |                                                        | |
|  |  4. Fallback (se errore)                               | |
|  |     SE errore: Prova prossimo modello in catena fallback | |
|  |                                                        | |
|  +------------------------+---------------------------------+ |
|                           |                                   |
|  Ritorna al Motore di Esecuzione                              |
|    ModelResponse {                                            |
|      text: "...",                                             |
|      model_used: "claude-sonnet-3.5",                         |
|      tokens: {input: 12234, output: 1456},                    |
|      cost: $0.082,                                            |
|      latency: 4.2s                                            |
|    }                                                          |
+---------------------------------------------------------------+
```

3.2 Tracciamento Performance Modello

Flusso di dati per miglioramento continuo:

```
+---------------------------------------------------------------+
|         FLUSSO TRACCIAMENTO PERFORMANCE MODELLO                |
|                                                                |
|  Ogni invocazione modello ->                                    |
|    |                                                           |
|    +---> Registra Metriche                                      |
|    |    {                                                      |
|    |      model_id: "claude-sonnet-3.5",                       |
```

```
|    |      task_type: "code_generation",                          |
|    |      complexity: "MODERATE",                                |
|    |      latency: 4.2s,                                         |
|    |      cost: $0.082,                                          |
|    |      tokens: {in: 12234, out: 1456},                        |
|    |      success: true,                                         |
|    |      quality_score: 0.89  // Da verifica                  |
|    |    }                                                        |
|    |                                                              |
|    +---> Aggrega in Time-Series DB                                |
|    |    * Bucket di 1 minuto per monitoraggio real-time          |
|    |    * Bucket di 1 ora per trend                              |
|    |    * Aggregati giornalieri per report                       |
|    |                                                              |
|    +---> Aggiorna Statistiche Modello                             |
|    |    model_stats["claude-sonnet-3.5"] {                       |
|    |      avg_latency: 4.5s (media mobile),                      |
|    |      avg_cost: $0.085,                                      |
|    |      success_rate: 94.2%,                                   |
|    |      quality_score: 0.87,                                   |
|    |      total_invocations: 15,432                              |
|    |    }                                                        |
|    |                                                              |
|    +---> Attiva Analisi (Settimanale)                            |
|         * Confronta performance modelli                          |
|         * Identifica opportunità di ottimizzazione               |
|         * Aggiusta pesi di routing se necessario                  |
|         * Genera raccomandazioni                                 |
|                                                                   |
|  Output Esempio:                                                  |
|    "Il modello claude-sonnet-3.5 mostra un tasso di successo     |
|     superiore del 15% rispetto a gpt-4o-mini per task di         |
|     generazione codice. Si raccomanda di aumentare il peso di    |
|     routing per questo modello."                                 |
+------------------------------------------------------------------+
```

4. Flussi di Verifica Sicurezza

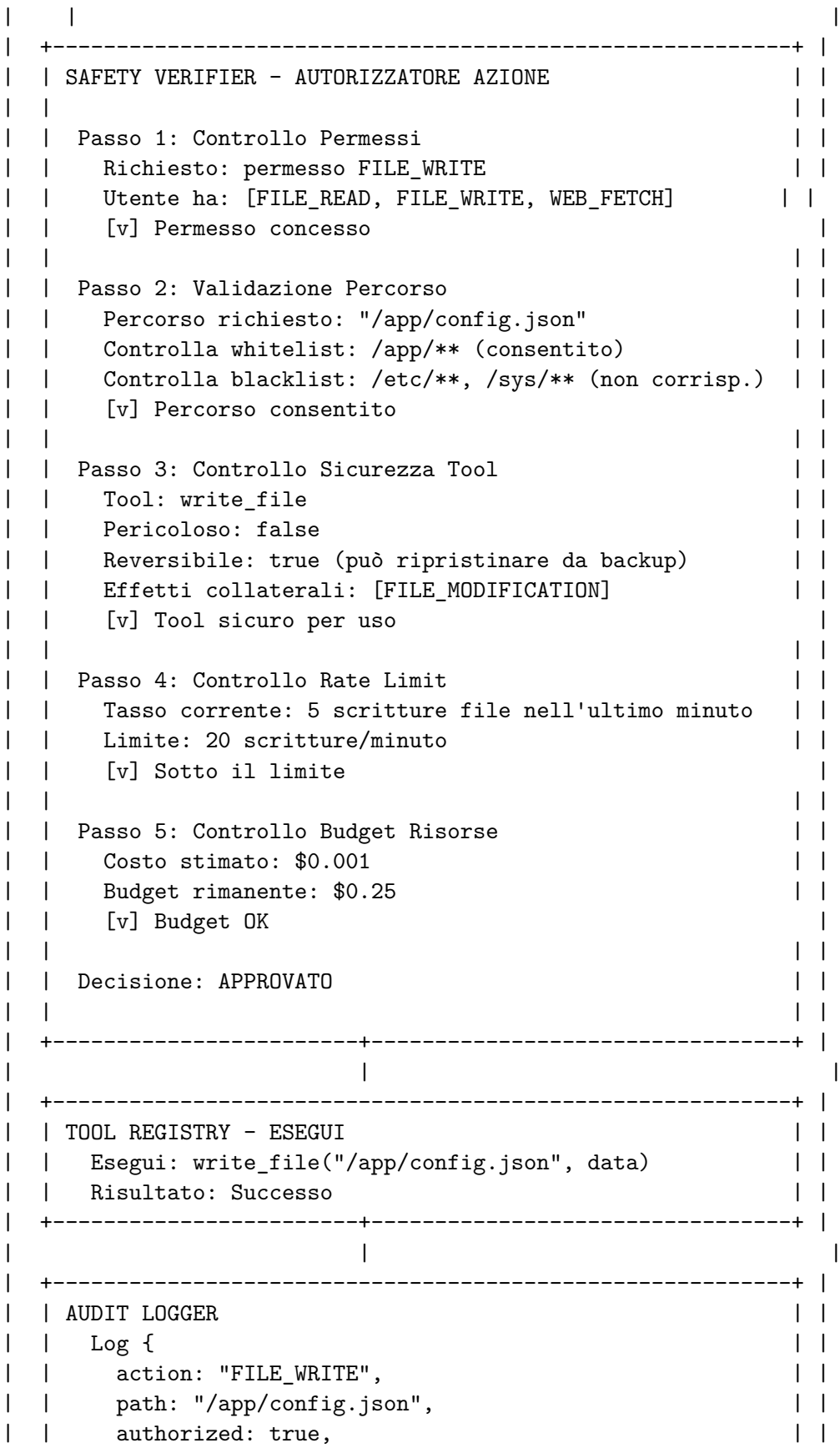
4.1 Flusso di Validazione Input

```
+------------------------------------------------------------------+
|              FLUSSO VALIDAZIONE INPUT                            |
|                                                                   |
|  Input Utente: "Elimina tutti i file in /important/data"         |
|    |                                                              |
|    |                                                              |
|  +-----------------------------------------------------------+  |
|  | SAFETY VERIFIER - VALIDATORE INPUT                        | |
```

```
|  |                                                          | |
|  |  Livello 1: Validazione Schema                           | |
|  |    [v] Input è stringa valida                              | |
|  |    [v] Lunghezza entro limiti                              | |
|  |                                                          | |
|  |  Livello 2: Rilevamento Injection                        | |
|  |    Esegui: CHECK_COMMAND_INJECTION(input)                | |
|  |    Rilevato: Potenziale comando pericoloso "elimina"     | |
|  |    Punteggio: 0.85 (alto rischio)                        | |
|  |                                                          | |
|  |  Livello 3: Policy Contenuti                             | |
|  |    Controllo: Richiesta coinvolge operazione distruttiva | |
|  |    Percorso: /important/data (posizione sensibile)       | |
|  |    Flag: Richiede approvazione esplicita                 | |
|  |                                                          | |
|  |  Decisione: RICHIEDE_APPROVAZIONE_UMANA                  | |
|  |                                                          | |
|  +-----------------------+----------------------------------+ |
|                          |                                    |
|  +-------------------------------------------------------+ |
|  | RICHIESTA APPROVAZIONE A UTENTE                       | |
|  |                                                       | |
|  |  "Questa azione eliminerà file in /important/data.    | |
|  |   Si tratta di un'operazione distruttiva.             | |
|  |   Vuoi procedere?"                                    | |
|  |                                                       | |
|  |   [Approva] [Nega] [Modifica Richiesta]               | |
|  |                                                       | |
|  +-----------------------+-------------------------------+ |
|                          |                                    |
|  SE APPROVATO:                                                |
|    Procedi con analisi goal                                  |
|    Registra approvazione in audit trail                      |
|                                                               |
|  SE NEGATO:                                                   |
|    Ritorna errore a utente                                   |
|    Registra negazione                                        |
|    Suggerisci alternative più sicure                         |
+---------------------------------------------------------------+
```

4.2 Flusso di Autorizzazione Azione

```
+---------------------------------------------------------------+
|            FLUSSO AUTORIZZAZIONE AZIONE                       |
|                                                               |
|  Il Motore di Esecuzione vuole: write_file("/app/config.json",|
|  data)                                                        |
|    |                                                          |
```


SAFETY VERIFIER - AUTORIZZATORE AZIONE
Passo 1: Controllo Permessi
Richiesto: permesso FILE_WRITE
Utente ha: [FILE_READ, FILE_WRITE, WEB_FETCH]
[v] Permesso concesso
Passo 2: Validazione Percorso
Percorso richiesto: "/app/config.json"
Controlla whitelist: /app/** (consentito)
Controlla blacklist: /etc/**, /sys/** (non corrisp.)
[v] Percorso consentito
Passo 3: Controllo Sicurezza Tool
Tool: write_file
Pericoloso: false
Reversibile: true (può ripristinare da backup)
Effetti collaterali: [FILE_MODIFICATION]
[v] Tool sicuro per uso
Passo 4: Controllo Rate Limit
Tasso corrente: 5 scritture file nell'ultimo minuto
Limite: 20 scritture/minuto
[v] Sotto il limite
Passo 5: Controllo Budget Risorse
Costo stimato: $0.001
Budget rimanente: $0.25
[v] Budget OK
Decisione: APPROVATO
TOOL REGISTRY - ESEGUI
Esegui: write_file("/app/config.json", data)
Risultato: Successo
AUDIT LOGGER
Log {
action: "FILE_WRITE",
path: "/app/config.json",
authorized: true,

```
|  |     executed: true,                                    | |
|  |     result: "success"                                  | |
|  |   }                                                    | |
|  |  +-----------------------------------------------------+ |
+-------------------------------------------------------------+
```

4.3 Flusso di Validazione Output

```
+-------------------------------------------------------------+
|             FLUSSO VALIDAZIONE OUTPUT                       |
|                                                             |
|  Task completato, output generato:                          |
|    Codice: (Funzione Python per encoding JWT)               |
|    |                                                        |
|    |                                                        |
|  +-----------------------------------------------------+ |
|  | SAFETY VERIFIER - VALIDATORE OUTPUT                  | |
|  |                                                      | |
|  |  Controllo 1: Validazione Schema                     | |
|  |    Atteso: Codice Python (stringa)                   | |
|  |    Effettivo: Stringa contenente codice Python       | |
|  |    [v] Schema valido                                   | |
|  |                                                      | |
|  |  Controllo 2: Validazione Sintassi                   | |
|  |    Analizza con parser AST Python                    | |
|  |    Risultato: Sintassi Python valida                 | |
|  |    [v] Sintassi valida                                 | |
|  |                                                      | |
|  |  Controllo 3: Scansione Sicurezza                    | |
|  |    Scansione per pattern pericolosi:                 | |
|  |      * eval() -> Non trovato [v]                       | |
|  |      * exec() -> Non trovato [v]                       | |
|  |      * __import__() -> Non trovato [v]                 | |
|  |      * os.system() -> Non trovato [v]                  | |
|  |    [v] Nessun problema di sicurezza                  | |
|  |                                                      | |
|  |  Controllo 4: Rilevamento Segreti                    | |
|  |    Scansione per pattern segreti:                    | |
|  |      * Chiavi API -> Non trovato [v]                   | |
|  |      * Password -> Non trovato [v]                     | |
|  |      * Chiavi private -> Non trovato [v]               | |
|  |    [v] Nessun segreto trapelato                      | |
|  |                                                      | |
|  |  Controllo 5: Soglie Qualità                         | |
|  |    Valuta qualità codice:                            | |
|  |      * Ha docstring: [x] (avviso)                    | |
|  |      * Ha type hints: [v]                            | |
|  |      * Gestione errori: [v]                          | |
```

```
|  |    Qualità complessiva: 0.85 (sopra soglia 0.7)         | |
|  |    [v] Qualità accettabile                                 | |
|  |                                                          | |
|  |  Decisione: APPROVATO (con 1 avviso)                     | |
|  |                                                          | |
|  +-------------------------+--------------------------------+ |
|                            |                                  |
|  Ritorna al Motore di Esecuzione:                             |
|    ValidationResult {                                         |
|      approved: true,                                          |
|      warnings: ["Docstring mancante"],                        |
|      validated_output: (output originale)                     |
|    }                                                          |
+---------------------------------------------------------------+
```

5. Flussi di Gestione Errori

5.1 Rilevamento Errori e Recovery

```
+---------------------------------------------------------------+
|        FLUSSO RILEVAMENTO ERRORI E RECOVERY                   |
|                                                               |
|  Esecuzione: tool_invocation("web_search", query="...")       |
|    |                                                          |
|    +---> Esecuzione tool fallisce                              |
|    |    Errore: TimeoutError("Richiesta scaduta dopo 30s")    |
|    |                                                          |
|    |                                                          |
|  +-------------------------------------------------------+ |
|  | ERROR HANDLER - CLASSIFICATORE                        | |
|  |                                                       | |
|  |  Analizza Errore:                                     | |
|  |    Tipo: TimeoutError                                 | |
|  |    Componente: Tool (web_search)                      | |
|  |    Categoria: TRANSIENT (problema rete)               | |
|  |    Gravità: MEDIA                                     | |
|  |    Recuperabile: SI (è probabile che retry funzioni)  | |
|  |                                                       | |
|  +-------------------------+-----------------------------+ |
|                            |                                  |
|  +-------------------------------------------------------+ |
|  | ERROR HANDLER - MOTORE DI RECOVERY                    | |
|  |                                                       | |
|  |  Seleziona Strategia: RETRY con backoff esponenziale  | |
|  |                                                       | |
|  |  Tentativo 1: Retry immediato                         | |
|  |    -> FALLITO (timeout di nuovo)                       | |
|  |    Attesa: 2s                                         | |
```

```
|  |                                                          | |
|  |  Tentativo 2: Retry dopo 2s                              | |
|  |    -> FALLITO (timeout di nuovo)                          | |
|  |    Attesa: 4s                                            | |
|  |                                                          | |
|  |  Tentativo 3: Retry dopo 4s                              | |
|  |    -> SUCCESSO!                                           | |
|  |    Risultato: Risultati ricerca restituiti               | |
|  |                                                          | |
|  |  Recovery: RIUSCITO                                      | |
|  |                                                          | |
|  +------------------------+---------------------------------+ |
|                           |                                   |
|  +--------------------------------------------------------+ |
|  | SISTEMA OSSERVABILITÀ - LOGGING                          | |
|  |                                                          | |
|  |  Voce Log {                                              | |
|  |    level: "WARNING",                                     | |
|  |    event: "tool_timeout_recovered",                      | |
|  |    tool: "web_search",                                   | |
|  |    attempts: 3,                                          | |
|  |    total_delay: 6s,                                      | |
|  |    outcome: "success"                                    | |
|  |  }                                                       | |
|  |                                                          | |
|  +------------------------+---------------------------------+ |
|                           |                                   |
|  Ritorna al Motore di Esecuzione:                             |
|    RecoveryResult {                                          |
|      success: true,                                          |
|      output: (risultati ricerca),                            |
|      recovery_applied: "retry_with_backoff",                 |
|      attempts_needed: 3,                                     |
|      delay_incurred: 6s                                      |
|    }                                                         |
|                                                               |
|  Il task continua con risultato riuscito                      |
+---------------------------------------------------------------+
```

5.2 Flusso di Escalation Errori

```
+---------------------------------------------------------------+
|              FLUSSO ESCALATION ERRORI                         |
|                                                               |
|  Fallimenti multipli tool -> Recovery non funziona             |
|    |                                                         |
|    |                                                         |
|  +--------------------------------------------------------+ |
```


ERROR HANDLER - GESTORE ESCALATION
Valuta Situazione:
* 3 fallimenti tool consecutivi
* Tutti i tentativi di recovery esauriti
* Il task non può procedere
* Gravità: ALTA
Decisione: ESCALA a utente
CREA INCIDENT
Incident {
id: "INC-123",
type: "TASK_FAILURE",
severity: "HIGH",
task_id: "T-12345",
error_summary: "Fallimenti multipli tool",
recovery_attempts: [...]
}
NOTIFICA UTENTE
Messaggio all'Utente:
"Il task 'Aggiungi autenticazione JWT' ha incontrato
un errore e non ha potuto essere completato
automaticamente.
Errore: Tentativi multipli di ricerca documentazione
falliti a causa di timeout di rete.
Opzioni:
1. Riprova task (potrebbe fallire di nuovo)
2. Salta step ricerca documentazione
3. Fornisci input manuale per questo step
4. Annulla task
ID Incident: INC-123"

```
|  | ALLERTA ENGINEERING (se critico)                          | |
|  |                                                           | |
|  |  SE gravità == CRITICAL:                                  | |
|  |    Invia allerta a ingegnere di guardia                   | |
|  |    Include: dettagli incident, stato sistema, log         | |
|  |                                                           | |
|  +-------------------------+---------------------------------+ |
|                            |                                   |
|  Attendi decisione utente o intervento engineering             |
+----------------------------------------------------------------+
```

6. Flussi di Dati di Osservabilità

6.1 Flusso di Logging

```
+----------------------------------------------------------------+
|                  FLUSSO DATI DI LOGGING                        |
|                                                                |
|  Ogni componente genera voci di log ->                          |
|    |                                                           |
|    +---> LOG_INFO("Task iniziato", {task_id: "T-123", ...})    |
|    +---> LOG_DEBUG("Contesto recuperato", {...})               |
|    +---> LOG_WARNING("Fallback usato", {...})                  |
|    +---> LOG_ERROR("Operazione fallita", {...})                |
|    |                                                           |
|    |                                                           |
|  +-------------------------------------------------------+ |
|  | AGGREGATORE LOG                                       | |
|  |  * Raccoglie da tutte le fonti                        | |
|  |  * Aggiunge correlation_id (trace_id, task_id)        | |
|  |  * Arricchisce con contesto (host, componente, timestamp)| |
|  |  * Buffer (100 voci o 5s, qualunque arrivi prima)     | |
|  +-------------------------+-----------------------------+ |
|                            |                                  |
|  +-------------------------------------------------------+ |
|  | SCRITTURA BATCH A STORAGE                             | |
|  |  * Elasticsearch (se self-hosted)                     | |
|  |  * CloudWatch Logs (se AWS)                           | |
|  |  * Ritenzione: 30 giorni hot, 1 anno archivio         | |
|  +-------------------------+-----------------------------+ |
|                            |                                  |
|  +-------------------------------------------------------+ |
|  | INDICIZZAZIONE PER RICERCA                            | |
|  |  * Indice full-text su messaggio                      | |
|  |  * Indici campo su: level, component, user_id, task_id | |
|  |  * Partizionamento basato su tempo per efficienza     | |
|  +-------------------------+-----------------------------+ |
|                            |                                  |
```

```
|  Disponibile per:                                                  |
|    * Ricerca real-time (Kibana, CloudWatch Insights)              |
|    * Debugging (traccia tutti gli eventi per task specifico)      |
|    * Analytics (trend tasso errori, analisi performance)          |
|    * Compliance (audit trail)                                      |
+--------------------------------------------------------------------+
```

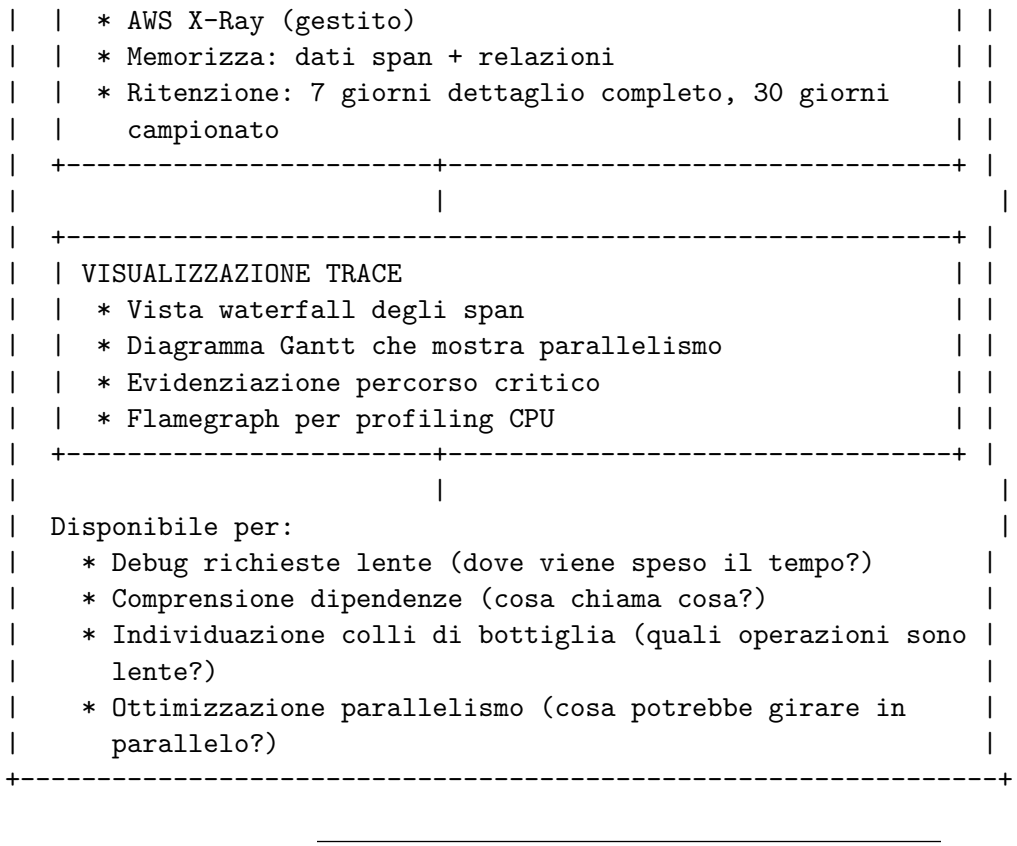
6.2 Flusso di Metriche

```
+--------------------------------------------------------------------+
|                 FLUSSO DATI DI METRICHE                            |
|                                                                    |
|  Operazioni componenti generano metriche ->                         |
|    |                                                               |
|    +---> COUNTER("tasks_started", labels={user_id, type})          |
|    +---> HISTOGRAM("task_duration_seconds", value=245.3)           |
|    +---> GAUGE("active_tasks", value=12)                           |
|    +---> COUNTER("llm_cost_dollars", value=0.18)                   |
|    |                                                               |
|    |                                                               |
|  +-------------------------------------------------------------+   |
|  | COLLETTORE METRICHE                                         |   |
|  |  * Aggregazione in memoria (finestra 1 minuto)              |   |
|  |  * Calcola statistiche: count, sum, min, max, percentili    |   |
|  +-------------------------+-----------------------------------+   |
|                            |                                       |
|  +-------------------------------------------------------------+   |
|  | SCRITTURA A TIME-SERIES DB                                  |   |
|  |  * Prometheus (self-hosted)                                 |   |
|  |  * CloudWatch Metrics (AWS)                                 |   |
|  |  * InfluxDB (alternativa)                                   |   |
|  |  * Ogni 1 minuto per metriche dettagliate                   |   |
|  |  * Downsampled a 5min, 1ora, 1giorno per lungo termine      |   |
|  +-------------------------+-----------------------------------+   |
|                            |                                       |
|  +-------------------------------------------------------------+   |
|  | VISUALIZZAZIONE & ALERTING                                  |   |
|  |  * Dashboard Grafana (real-time)                            |   |
|  |  * Valutazione regole alert (ogni 1min)                     |   |
|  |  * Rilevamento anomalie (basato su ML, opzionale)           |   |
|  +-------------------------+-----------------------------------+   |
|                            |                                       |
|  +-------------------------------------------------------------+   |
|  | ROUTING ALERT (se soglia superata)                          |   |
|  |  * PagerDuty (alert critici)                                |   |
|  |  * Slack (avvisi)                                           |   |
|  |  * Email (info)                                             |   |
|  +-------------------------------------------------------------+   |
```

```
|                                                                  |
|  Usato per:                                                      |
|    * Monitoraggio real-time (salute sistema)                     |
|    * Ottimizzazione performance (identifica colli di bottiglia)  |
|    * Pianificazione capacità (prevede necessità risorse)         |
|    * Tracciamento SLA (tasso successo, target latenza)           |
+------------------------------------------------------------------+
```

6.3 Flusso di Tracing

```
+------------------------------------------------------------------+
|                FLUSSO TRACING DISTRIBUITO                        |
|                                                                  |
|  Arriva richiesta -> trace_id generato                           |
|    |                                                             |
|    +---> Span Root creato                                        |
|    |    trace_id: "abc-123"                                      |
|    |    span_id: "span-1"                                        |
|    |    operation: "handle_task"                                 |
|    |                                                             |
|    +---> Man mano che la richiesta fluisce attraverso sistema:   |
|         |                                                        |
|         +--> Analisi Goal crea span figlio                       |
|         |   span_id: "span-2", parent: "span-1"                  |
|         |                                                        |
|         +--> Pianificazione crea span figlio                     |
|         |   span_id: "span-3", parent: "span-1"                  |
|         |   |                                                    |
|         |   +--> Query Pattern Cache crea span                   |
|         |   |   span_id: "span-4", parent: "span-3"              |
|         |   |                                                    |
|         |   +--> Query memoria crea span                         |
|         |       span_id: "span-5", parent: "span-3"              |
|         |                                                        |
|         +--> Esecuzione crea span multipli                       |
|             (uno per subtask)                                    |
|             span_id: "span-6..N", parent: vari                   |
|    |                                                             |
|  +-------------------------------------------------------------+ |
|  | COLLEZIONE SPAN                                             | |
|  |  * Ogni span inviato a collector quando completato          | |
|  |  * Include: timing, tag, eventi, stato                      | |
|  |  * In batch per efficienza                                  | |
|  +-----------------------+-------------------------------------+ |
|                          |                                       |
|  +-------------------------------------------------------------+ |
|  | STORAGE TRACE                                               | |
|  |  * Jaeger (open source)                                     | |
```



Prossimo: 08-deployment.md -> Modelli di deployment, strategie di scaling, considerazioni operative