Actuarial Reserving System

Technical Architecture Documentation

Version: 1.0
Last Updated: 2026-02-03
Author: Auto-generated from codebase analysis

Last Updated: 2026-02-03
Author: Auto-generated from codebase analysis

1. Project Structure

1.1 Directory Layout

```
reserving-agent/
■■■ app.py                               # Streamlit entry point (landing page)
■■■ pages/
■   ■■■ 1_Reported_Claims.py             # Analysis & model selection UI
■   ■■■ 2_Summary.py                     # Ultimates & scenario analysis UI
■   ■■■ 3_■_AI_Team.py                  # Multi-agent chat interface
■■■ src/
■   ■■■ agents/                          # Multi-agent system
■   ■   ■■■ orchestrator.py              # Request router & workflow coordinator
■   ■   ■■■ methodology.py               # Analysis planning agent
■   ■   ■■■ selection.py                 # LLM method selection agent
■   ■   ■■■ reserving.py                 # Execution agent (calculations)
■   ■   ■■■ validation.py                # Peer review agent
■   ■   ■■■ reporting.py                 # Report generator agent
■   ■   ■■■ qa.py                        # Q&A specialist agent
■   ■   ■■■ schemas.py                   # Pydantic data models
■   ■   ■■■ llm_utils.py                 # OpenAI API wrapper
■   ■   ■■■ settings.py                  # Path configuration
■   ■   ■■■ main.py                      # CLI entry point
■   ■■■ chain_ladder.py                  # Core Chain Ladder implementation
■   ■■■ data_loader.py                   # Triangle loading utilities
■   ■■■ enhanced_workflow.py             # Full analysis pipeline
■   ■■■ visualizer.py                    # Plotting utilities
■   ■■■ stochastic_reserving/
■   ■   ■■■ mack_model.py                # Mack Chain Ladder (parametric)
■   ■   ■■■ bootstrap.py                 # ODP Bootstrap (non-parametric)
■   ■■■ alternative_methods/
■   ■   ■■■ bornhuetter_ferguson.py      # BF method
■   ■   ■■■ cape_cod.py                  # Cape Cod method
■   ■■■ model_selection/
■   ■   ■■■ factor_estimators.py         # 7 factor aggregation strategies
■   ■   ■■■ model_selector.py            # Model comparison framework
■   ■   ■■■ validation_framework.py      # Holdout/rolling-origin CV
■   ■   ■■■ error_metrics.py             # 11 error metrics
■   ■   ■■■ statistical_tests.py         # DM test, t-test, Wilcoxon
■   ■■■ diagnostics/
■   ■   ■■■ diagnostic_tests.py          # 5 Mack assumption tests
■   ■   ■■■ volatility_analysis.py       # Factor stability analysis
■   ■   ■■■ residual_analysis.py         # Residual diagnostics
■   ■■■ tail_fitting/
■   ■   ■■■ tail_estimator.py            # 7 tail curve fitting methods
■   ■■■ ml_models/
■   ■   ■■■ gradient_boosting_factors.py # GBM factor selection
■   ■   ■■■ anomaly_detector.py          # Triangle anomaly detection
■   ■■■ scenario_analysis/
■       ■■■ economic_scenario_generator.py
■       ■■■ stress_testing.py
■       ■■■ tail_risk.py
■■■ data/
■   ■■■ processed/                       # Primary data files
■   ■   ■■■ reported_absolute_losses.csv
■   ■   ■■■ earned_premium.csv
■   ■■■ sample_triangles/                # Benchmark datasets
■■■ outputs/                             # Generated reports
■■■ requirements.txt
■■■ README.md
```

1.2 File Responsibilities

File	Responsibility	Key Classes/Functions
`orchestrator.py`	Request routing, workflow orchestration	`Orchestrator.route_request()`, `stream_workflow()`
`methodology.py`	Analysis type determination	`MethodologyAgent.plan_analysis()`
`selection.py`	LLM-driven method selection	`SelectionAgent.analyze_and_select()`
`reserving.py`	Actuarial calculations	`ReservingExecutionAgent.execute()`
`validation.py`	Result validation	`ValidationAgent.validate()`
`reporting.py`	Report generation	`ReportingAgent.generate_report()`
`qa.py`	Question answering	`QASpecialistAgent.answer_query()`
`schemas.py`	Data models	`ReservingOutput`, `ValidationReport`, etc.
`llm_utils.py`	OpenAI integration	`LLMClient.get_completion()`

1.3 Dependencies

```
# Core
numpy>=1.21.0
pandas>=1.3.0
pydantic>=2.0.0

# UI
streamlit>=1.28.0
plotly>=5.0.0

# Data
openpyxl>=3.0.0

# LLM
openai>=1.0.0
```

2. Core Workflow

2.1 Request Flow Diagram

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                         USER REQUEST                               ■
■                    (via Chat Interface)                            ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                              ■
                              ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                      ORCHESTRATOR                                  ■
■                  determine_intent(query)                           ■
■                                                                    ■
■   ■■■■■■■■■■■■■■■■■■■■            ■■■■■■■■■■■■■■■■■■■■           ■
■   ■  NEW_ANALYSIS    ■          ■    Q_AND_A     ■          ■
■   ■  (run workflow)  ■          ■  (use context) ■          ■
■   ■■■■■■■■■■■■■■■■■■■■■■■■■          ■■■■■■■■■■■■■■■■■■■■■■■■■           ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
             ■                             ■
             ▼                             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■    ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■   ANALYSIS PIPELINE        ■    ■    QA SPECIALIST        ■
■                            ■    ■                          ■
■  1. Methodology Agent      ■    ■  answer_query(context)   ■
■         ↓                  ■    ■          ↓               ■
■  2. Selection Agent        ■    ■  LLM/Fallback response   ■
■         ↓                  ■    ■                          ■
■  3. Execution Agent        ■    ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■         ↓                  ■
■  4. Validation Agent       ■
■         ↓                  ■
■  5. Reporting Agent        ■
■         ↓                  ■
■  6. Final Synthesis        ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
             ■
             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                        RESULT BUNDLE                                ■
■  {                                                                  ■
■    report: string (markdown),                                       ■
■    structured_results: ReservingOutput,                             ■
■    validation: ValidationReport,                                    ■
■    method_selection: dict,                                          ■
■    audit_trail: AgentLog[]                                          ■
■  }                                                                  ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
             ■
             ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                        UI RENDERING                                 ■
■  Tab 1: Final Report (Markdown)                                     ■
■  Tab 2: Method Selection Details                                    ■
■  Tab 3: Charts (Plotly visualizations)                             ■
■  Tab 4: Structured Data (JSON)                                      ■
■  Tab 5: Audit Trail                                                 ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

2.2 Streaming Architecture

L'Orchestrator usa un **generatore Python** per streaming in tempo reale:

```
def route_request(self, message, current_result, inputs):
    intent = self.determine_intent(message, has_context)

    if intent == "NEW_ANALYSIS":
        yield {"step": "router", "message": "Starting analysis..."}

        for update in self.stream_workflow(message, inputs):
            yield update  # Real-time status updates

    elif intent == "Q_AND_A":
        yield {"step": "router", "message": "Answering question..."}
        answer, log = self.qa_agent.answer_query(message, current_result)
        yield {"step": "qa", "status": "done", "data": answer}
```

Update Format:

```
{
    "step": "methodology" | "selection" | "execution" | "validation" | "reporting" | "complete",
    "status": "running" | "done",
    "message": str,
    "data": Any  # Agent-specific output
}
```

3. Agents Analysis

3.1 Orchestrator

File: `src/agents/orchestrator.py`

Classe: `Orchestrator`

Responsabilità:
- Router delle richieste (NEW_ANALYSIS vs Q_AND_A)
- Coordinamento pipeline multi-agente
- Streaming status updates alla UI

Metodi Principali:

Metodo	Input	Output	Descrizione	
`determine_intent()`	query: str, has_context: bool	"NEW_ANALYSIS" \	"Q_AND_A"	Classifica intent via LLM o euristica
`route_request()`	message, current_result, inputs	Generator[dict]	Entry point unificato	
`stream_workflow()`	request, inputs	Generator[dict]	Esegue pipeline 6-step	
`ask_question()`	query, context	Generator[dict]	Path Q&A	

Intent Detection Logic:
```
# LLM prompt
"Classify: NEW_ANALYSIS (new calculations) or Q_AND_A (question about existing)"

# Fallback heuristics
if "run" in query or "calculate" in query or "analyze" in query:
    return "NEW_ANALYSIS"
else:
    return "Q_AND_A"
```

Limiti:
- Intent detection dipende da LLM disponibile
- Nessun supporto per richieste ibride (analisi + domanda)

3.2 Methodology Agent

File: `src/agents/methodology.py`

Classe: `MethodologyAgent`

Responsabilità:
- Analizzare la richiesta utente
- Determinare il tipo di analisi (QUICK/STANDARD/FULL)
- Produrre configurazione per l'Execution Agent

Input: Stringa richiesta naturale

Output: `(ReservingConfigFile, AgentLog)`

Logica di Mapping:

Keyword	Analysis Type	Methods Enabled
"quick", "fast"	QUICK	Chain Ladder only
"stress", "full", "complete"	FULL	All methods + stress testing
default	STANDARD	CL + Mack + Bootstrap

Configurazione Risultante:

```
class ReservingConfigFile:
    analysis_type: AnalysisType
    run_model_selection: bool = True
    run_bootstrap: bool = True
    n_bootstrap_simulations: int = 1000
    run_diagnostics: bool = True
    run_stress_testing: bool = False  # True solo per FULL
```

Limiti:
- Mapping basato su keyword semplici
- Nessun parsing NLU avanzato
- Configurazione non parametrizzabile dall'utente

3.3 Selection Agent (LLM-Driven)

File: `src/agents/selection.py`

Classe: `SelectionAgent`

Responsabilità:
- Calcolare TUTTI i 7 factor estimators
- Eseguire validazione completa (13 metriche)
- Eseguire 5 test diagnostici
- Analizzare volatilità e tail fitting
- Chiedere all'LLM di selezionare il metodo ottimale

Metodi Principali:

Metodo	Descrizione
`analyze_and_select(triangle, premium)`	Entry point principale
`_calculate_all_estimators(triangle)`	Calcola 7 estimatori
`_run_full_validation(triangle)`	13 error metrics via ModelSelector
`_run_diagnostics(triangle)`	5 test Mack
`_analyze_volatility(triangle)`	CV, trend, structural breaks
`_fit_tail(triangle)`	7 curve fitting methods
`_calculate_maturity(triangle)`	% maturity per anno
`_calculate_bf_comparison(triangle, premium)`	BF vs CL
`_prepare_comprehensive_context(all_data)`	Formatta per LLM

Metodo	Descrizione
`_llm_select(context, all_data)`	Chiede all'LLM
`_fallback_selection(all_data)`	Regole se LLM non disponibile

7 Factor Estimators Calcolati:

1. Simple Average
2. Volume-Weighted
3. Geometric Average
4. Median
5. Leverage-Weighted
6. Exclude High/Low
7. Mack Adjusted

LLM Selection Prompt:

```
Based on validation metrics, diagnostic tests, and maturity analysis, decide:
1. Which factor estimator to use as BASE method
2. Which years should use Bornhuetter-Ferguson (<70% maturity)
3. Whether to apply tail factor adjustment

RESPOND IN JSON:
{
    "selected_estimator": "...",
    "estimator_reason": "...",
    "bf_years": ["2022", "2023"],
    "bf_reason": "...",
    "summary": "..."
}
```

Output:

```
{
    "selected_estimator": "Volume-Weighted",
    "estimator_reason": "Lowest MSE in out-of-sample validation",
    "bf_years": ["2022", "2023"],
    "bf_reason": "Years with <70% maturity",
    "all_estimators": {"Simple Average": 1234567, ...},
    "validation_metrics": {"Simple Average": {"MSE": 123, "MAE": 45}, ...},
    "maturity_by_year": {"2018": 95.2, "2019": 87.1, ...}
}
```

Limiti:
- Dipende da LLM per selezione intelligente
- Fallback usa solo MSE (potrebbe non essere ottimale)
- Nessun supporto per preferenze utente

3.4 Execution Agent (Actuary)

File: `src/agents/reserving.py`

Classe: `ReservingExecutionAgent`

Responsabilità:
- Caricare dati dal filesystem
- Eseguire calcoli attuariali via `EnhancedReservingWorkflow`

- Confezionare risultati in `ReservingOutput`

Metodi:

Metodo	Input	Output	
`execute(inputs, config)`	ReservingInput, ReservingConfigFile	ReservingOutput	
`_load_data(path, is_series)`	path: str	DataFrame \	Series
`_package_results(workflow, config, triangle)`	...	ReservingOutput	

Pipeline di Esecuzione:

```
# 1. Load data
triangle = self._load_data(inputs.triangle_path)
premium = self._load_data(inputs.premium_path, is_series=True)

# 2. Create workflow
workflow = EnhancedReservingWorkflow(triangle, premium)

# 3. Run Chain Ladder (sempre)
workflow.run_chain_ladder()

# 4. Run Mack (se STANDARD o FULL)
if config.analysis_type != AnalysisType.QUICK:
    workflow.run_mack_model()

# 5. Run Bootstrap (se abilitato)
if config.run_bootstrap:
    workflow.run_bootstrap(n_simulations=config.n_bootstrap_simulations)

# 6. Run BF e Cape Cod (se FULL)
if config.analysis_type == AnalysisType.FULL:
    workflow.run_alternative_methods()

# 7. Run Diagnostics
if config.run_diagnostics:
    workflow.run_diagnostics()

# 8. Package results
return self._package_results(workflow, config, triangle)
```

Data Enrichment per LLM:

Il metodo `_package_results` popola:
- `triangle_info`: metadata (n_years, periods, range)
- `detailed_data`: triangolo completo, fattori, riserve per anno
- Questi dati permettono all'LLM di rispondere a qualsiasi domanda

Limiti:
- Path hardcoded per alcuni file
- Nessun supporto per formati diversi da CSV
- Errori di caricamento non granulari

3.5 Validation Agent

File: `src/agents/validation.py`

Classe: `ValidationAgent`

Responsabilità:
- Verificare sanity dei risultati
- Confrontare metodi per coerenza
- Assegnare confidence score

Controlli Eseguiti:

Check	Condizione	Severity	Score Impact
Model Adequacy	score < 60	WARNING	-15
Mack CV	cv > 25%	WARNING	-10
CL vs Mack Divergence	diff > 10%	WARNING	-10
Bootstrap vs CL	diff > 15%	INFO	-5

Calcolo Confidence Score:

```
score = 100

if diagnostics.adequacy_score < 60:
    score -= 15
    issues.append(ValidationIssue(severity="WARNING", ...))

if mack.cv > 0.25:
    score -= 10
    issues.append(ValidationIssue(severity="WARNING", ...))

# etc.

return ValidationReport(
    status=ValidationStatus.WARNING if issues else ValidationStatus.PASSED,
    overall_confidence_score=max(0, score),
    issues=issues
)
```

Output: \`ValidationReport\`

Limiti:
- Soglie hardcoded
- Nessun check su BF o Cape Cod
- Manca confronto con benchmark industry

3.6 Reporting Agent

File: \`src/agents/reporting.py\`

Classe: \`ReportingAgent\`

Responsabilità:
- Generare report markdown completo
- Includere tutte le decisioni LLM
- Formattare tabelle e sezioni

Sezioni Generate:

1. **Header**: Data, Status, Methodology
2. **Data Overview**: Triangle metadata

3. ■ **AI Method Selection**:
- Selected estimator + reason
- BF years + reason
- All estimators comparison table
- Validation metrics table
- Maturity by year table
- AI Summary
4. **Executive Summary**: Reserve recommendation
5. **Reserve Estimates by Method**: Comparison table
6. **Bootstrap Confidence Intervals**: Percentiles
7. **Development Factors**: Factor table
8. **Reserves by Accident Year**: Year breakdown
9. **Validation & Diagnostics**: Findings
10. **Footer**: Disclaimer

Output: Markdown string (~3000-5000 chars)

Limiti:
- Solo formato Markdown
- Nessun export PDF/Excel diretto
- Layout non customizzabile

3.7 QA Specialist Agent

File: `` `src/agents/qa.py` ``

Classe: `` `QASpecialistAgent` ``

Responsabilità:
- Rispondere a domande sui risultati
- Usare LLM con contesto completo
- Fallback a pattern matching

Strategia:

```
1. Se LLM disponibile:
   - Serializza ReservingOutput + ValidationReport in JSON
   - Passa come contesto all'LLM
   - LLM genera risposta naturale

2. Se LLM non disponibile:
   - Pattern matching su keyword
   - Estrae valori specifici dal contesto
   - Formatta risposta template
```

Keyword Patterns (Fallback):

Pattern	Risposta
"validation", "issues"	Lista issues da ValidationReport
"ibnr", "reserve"	Total reserve da chain_ladder
"ultimate"	Ultimate loss
"mack", "error"	Standard error e CV
"bootstrap", "percentile"	Percentili distribuzione

Pattern	Risposta
"bornhuetter", "bf"	BF reserve
"cape cod"	Cape Cod reserve
"diagnostic"	Adequacy score e rating

Limiti:
- Fallback molto limitato
- Nessun supporto per domande complesse
- Nessuna memoria conversazionale (oltre session state)

3.8 LLM Client

File: \`src/agents/llm_utils.py\`

Classe: \`LLMClient\`

Configurazione:

```
class LLMClient:
    def __init__(self):
        self.api_key = os.getenv("OPENAI_API_KEY")
        self.client = OpenAI(api_key=self.api_key) if self.api_key else None

    def is_available(self) -> bool:
        return self.client is not None

    def get_completion(self, system_prompt, user_prompt, model="gpt-4o-mini"):
        response = self.client.chat.completions.create(
            model=model,
            messages=[
                {"role": "system", "content": system_prompt},
                {"role": "user", "content": user_prompt}
            ],
            temperature=0.0  # Deterministic
        )
        return response.choices[0].message.content
```

Modello Default: \`gpt-4o-mini\` (veloce, economico)

Graceful Degradation: Tutti i chiamanti hanno logica fallback se LLM non disponibile.

4. Reserving Package Integration

4.1 Chiamate al Package

Il sistema usa internamente queste componenti:

Component	File	Chiamato da
\`ChainLadder\`	\`chain_ladder.py\`	ExecutionAgent, SelectionAgent
\`MackChainLadder\`	\`stochastic_reserving/mack_model.py\`	ExecutionAgent

Component	File	Chiamato da
`ODPBootstrap`	`stochastic_reserving/bootstrap.py`	ExecutionAgent
`BornhuetterFerguson`	`alternative_methods/bornhuetter_ferguson.py`	ExecutionAgent, SelectionAgent
`CapeCod`	`alternative_methods/cape_cod.py`	ExecutionAgent, SelectionAgent
`get_all_estimators()`	`model_selection/factor_estimators.py`	SelectionAgent
`ModelSelector`	`model_selection/model_selector.py`	SelectionAgent
`DiagnosticTests`	`diagnostics/diagnostic_tests.py`	SelectionAgent
`VolatilityAnalyzer`	`diagnostics/volatility_analysis.py`	SelectionAgent
`TailEstimator`	`tail_fitting/tail_estimator.py`	SelectionAgent

4.2 Flow dei Parametri

```
Triangle (DataFrame)
    ■
    ■■■■ ChainLadder(triangle)
    ■     ■■■ .calculate_age_to_age_factors()
    ■     ■■■ .select_development_factors()
    ■     ■■■ .calculate_cumulative_factors()
    ■     ■■■ .project_ultimate_losses()
    ■
    ■■■■ MackChainLadder(triangle)
    ■     ■■■ .fit()
    ■     ■■■ .get_confidence_intervals()
    ■
    ■■■■ ODPBootstrap(triangle, n_simulations=1000)
    ■     ■■■ .fit()
    ■     ■■■ .reserve_distribution
    ■
    ■■■■ BornhuetterFerguson(triangle, premium)
    ■     ■■■ .fit()
    ■     ■■■ .get_comparison()
    ■
    ■■■■ get_all_estimators()
         ■■■ [est.estimate(triangle) for est in estimators]
```

4.3 Gestione Risultati

Ogni metodo produce output specifico che viene mappato in Pydantic models:

```
# Chain Ladder → MethodResult
MethodResult(
    method_name="Chain Ladder",
    total_reserve=cl.ultimate_losses['Reserve'].sum(),
    ultimate_loss=cl.ultimate_losses['Ultimate'].sum()
)

# Mack → StochasticResult
StochasticResult(
    method_name="Mack",
    total_reserve=...,
    standard_error=mack.mse_reserves['SE'].sum(),
    cv=mack.mse_reserves['CV'].mean(),
    percentiles={"75%": ..., "95%": ...}
)
```

5. Data & Artifacts

5.1 Pydantic Models (schemas.py)

Enums:

Enum	Values	Usage
`AgentRole`	ORCHESTRATOR, METHODOLOGY, EXECUTION, VALIDATION, REPORTING	Audit trail
`AnalysisType`	QUICK, STANDARD, FULL	Config
`ValidationStatus`	PASSED, WARNING, REJECTED	Validation

Core Models:

```
ReservingInput
■■■ triangle_path: str
■■■ premium_path: Optional[str]
■■■ loss_ratios_path: Optional[str]

ReservingConfigFile
■■■ analysis_type: AnalysisType
■■■ run_model_selection: bool
■■■ run_bootstrap: bool
■■■ n_bootstrap_simulations: int
■■■ run_diagnostics: bool
■■■ run_stress_testing: bool

ReservingOutput
■■■ timestamp: datetime
■■■ config_used: ReservingConfigFile
■■■ triangle_info: TriangleMetadata
■■■ detailed_data: DetailedData
■■■ method_selection: MethodSelection
■■■ chain_ladder: MethodResult
■■■ mack: StochasticResult
■■■ bootstrap: StochasticResult
■■■ bornhuetter_ferguson: MethodResult
■■■ cape_cod: MethodResult
■■■ diagnostics: DiagnosticsResult

ValidationReport
■■■ status: ValidationStatus
■■■ overall_confidence_score: int (0-100)
■■■ issues: List[ValidationIssue]
■■■ comparison_summary: str

AgentLog
■■■ timestamp: datetime
■■■ agent: AgentRole
■■■ action: str
■■■ details: str
```

5.2 Storage Locations

Artifact	Location	Format
Input Triangle	`data/processed/reported_absolute_losses.csv`	CSV
Premium	`data/processed/earned_premium.csv`	CSV
Final Report	`outputs/agent_runs/final_report.md`	Markdown

Artifact	Location	Format
Session State	In-memory (Streamlit)	Python dict

5.3 Audit Trail

Ogni agente produce un `AgentLog`:

```
AgentLog(
    timestamp=datetime.now(),
    agent=AgentRole.EXECUTION,
    action="Run Chain Ladder",
    details="Completed with reserve = $1,234,567"
)
```

I log vengono aggregati in `audit_trail: List[AgentLog]` e mostrati nella UI.

6. Question & Re-Analysis Loop

6.1 Intent Detection

```
def determine_intent(self, query: str, has_context: bool) -> str:
    # 1. Try LLM
    if self.llm.is_available():
        prompt = f"Classify: NEW_ANALYSIS or Q_AND_A\nQuery: {query}"
        response = self.llm.get_completion(system_prompt, prompt)
        if "NEW_ANALYSIS" in response:
            return "NEW_ANALYSIS"
        return "Q_AND_A"

    # 2. Fallback heuristics
    analysis_keywords = ["run", "calculate", "analyze", "execute", "perform"]
    if any(kw in query.lower() for kw in analysis_keywords):
        return "NEW_ANALYSIS"
    return "Q_AND_A"
```

6.2 Q&A; Flow (Nessun Nuovo Calcolo)

```
User: "What is the CV for the Mack model?"
    ■
    ▼
Orchestrator.determine_intent() → "Q_AND_A"
    ■
    ▼
QASpecialistAgent.answer_query(query, context)
    ■
    ■■■ context = st.session_state.final_result
    ■
    ■■■ LLM: "Based on the Mack model results, the CV is 18.5%..."
    ■   OR
    ■■■ Fallback: "The Mack model shows CV: 18.5%"
    ■
    ▼
Response displayed in chat
```

6.3 Re-Analysis Flow (Nuovi Calcoli)

```
User: "Run a full analysis with stress testing"
    ■
    ▼
Orchestrator.determine_intent() → "NEW_ANALYSIS"
    ■
    ▼
stream_workflow()
    ■
    ■■■ Methodology: analysis_type=FULL, run_stress_testing=True
    ■■■ Selection: Calcola tutti estimatori
    ■■■ Execution: Run tutti i metodi
    ■■■ Validation: Check risultati
    ■■■ Reporting: Genera report
    ■■■ Synthesis: Risposta finale
    ■
    ▼
st.session_state.final_result = new_results
```

6.4 Limitazioni

- **Nessuna Memoria Conversazionale**: Ogni domanda è indipendente (solo session state)
- **Nessun Calcolo On-Demand**: Q&A non può triggerare calcoli specifici
- **Intent Binario**: Solo NEW_ANALYSIS o Q_AND_A, niente di intermedio

7. Example Execution Paths

7.1 Path 1: Analisi Standard

User Input: "Analyze the triangle"

Files Coinvolti:

```
pages/3_■_AI_Team.py
    ■■■ Orchestrator.route_request()
        ■■■ determine_intent() → "NEW_ANALYSIS"
        ■■■ stream_workflow()
            ■■■ methodology.py → MethodologyAgent.plan_analysis()
            ■   ■■■ Returns: AnalysisType.STANDARD
            ■
            ■■■ selection.py → SelectionAgent.analyze_and_select()
            ■   ■■■ model_selection/factor_estimators.py
            ■   ■■■ model_selection/model_selector.py
            ■   ■■■ diagnostics/diagnostic_tests.py
            ■   ■■■ Returns: {"selected_estimator": "Volume-Weighted", ...}
            ■
            ■■■ reserving.py → ReservingExecutionAgent.execute()
            ■   ■■■ data_loader.py
            ■   ■■■ chain_ladder.py
            ■   ■■■ stochastic_reserving/mack_model.py
            ■   ■■■ stochastic_reserving/bootstrap.py
            ■   ■■■ Returns: ReservingOutput
            ■
            ■■■ validation.py → ValidationAgent.validate()
            ■   ■■■ Returns: ValidationReport
            ■
            ■■■ reporting.py → ReportingAgent.generate_report()
```

```
                  ■■■ Returns: Markdown string
```

Output: Report completo con CL, Mack, Bootstrap, 7 estimatori confrontati.

7.2 Path 2: Analisi Full (con Stress Testing)

User Input: "Run full analysis with stress testing"

Differenze da Standard:

```
methodology.py
    ■■■ Keywords "full", "stress" detected
    ■■■ Returns: AnalysisType.FULL, run_stress_testing=True

reserving.py
    ■■■ Additional calls:
        ■■■ alternative_methods/bornhuetter_ferguson.py
        ■■■ alternative_methods/cape_cod.py
        ■■■ scenario_analysis/stress_testing.py
```

Output Aggiuntivo: BF reserve, Cape Cod reserve, stress scenarios.

7.3 Path 3: Domanda Post-Analisi

User Input: "How many accident years are in the triangle?"

Pre-condizione: `st.session_state.final_result` esiste

Files Coinvolti:

```
pages/3_■_AI_Team.py
    ■■■ Orchestrator.route_request()
        ■■■ determine_intent() → "Q_AND_A"
        ■■■ ask_question()
            ■■■ qa.py → QASpecialistAgent.answer_query()
                ■
                ■■■ context = final_result["structured_results"]
                ■
                ■■■ LLM available:
                ■   ■■■ "The triangle contains 10 accident years (2014-2023)"
                ■
                ■■■ LLM not available:
                    ■■■ Keyword match on "years" → extract from triangle_info
```

Output: Risposta testuale senza nuovi calcoli.

8. Gaps & Improvement Opportunities

8.1 Architetturali

Gap	Impatto	Soluzione Proposta
Nessun Code Interpreter	Q&A non può fare calcoli custom	Aggiungere CodeAgent con sandbox Python
Intent Detection Binario	Richieste ibride mal gestite	Classificazione multi-label o chain-of-thought
Nessuna Memoria Conversazionale	Ogni domanda isolata	Aggiungere conversation buffer o RAG
Soglie Hardcoded	Validazione non configurabile	Esternalizzare in config YAML

8.2 Calcoli

Gap	Impatto	Soluzione Proposta
Selection Agent Overkill	Calcola TUTTO ogni volta	Lazy evaluation o caching per estimatori
Bootstrap Lento	1000 simulazioni bloccanti	Async/background job o progressive loading
Nessun Caching Risultati	Re-calcolo su ogni run	Redis/pickle per risultati intermedi

8.3 UI/UX

Gap	Impatto	Soluzione Proposta
Solo Markdown Report	Nessun export professionale	Aggiungere PDF via ReportLab o FPDF
Grafici Non nel Report	Separazione report/charts	Integrare Plotly export in report
Nessun Upload Custom	Solo triangoli predefiniti	Aggiungere file uploader con validazione

8.4 Robustezza

Gap	Impatto	Soluzione Proposta
LLM Single Point of Failure	Fallback limitati	Migliorare fallback con regole più ricche
Nessun Rate Limiting	OpenAI 429 errors	Implementare retry con exponential backoff
Errori Non Granulari	Debug difficile	Structured logging con livelli

8.5 Testing

Gap	Impatto	Soluzione Proposta
Cartella tests/ Vuota	Nessun test automatizzato	Unit test per ogni agent
Nessun Integration Test	Regressioni non catturate	End-to-end test con fixtures

Appendix: Quick Reference

Agent Responsibilities

```
ORCHESTRATOR  ■■■■■■  Router, Coordinator
     ■
     ■■■ METHODOLOGY  ■■  Config Planning (QUICK/STANDARD/FULL)
     ■
     ■■■ SELECTION    ■■  LLM Method Selection (7 estimators)
     ■
     ■■■ EXECUTION    ■■  Actuarial Calculations
     ■
     ■■■ VALIDATION   ■■  Sanity Checks, Scoring
     ■
     ■■■ REPORTING    ■■  Markdown Generation
     ■
     ■■■ QA SPECIALIST ■■ Question Answering
```

Data Flow

```
CSV Files → DataFrame → Calculations → Pydantic Models → JSON/Markdown → UI
```

Key Files to Modify

Per...	**Modifica...**
Aggiungere metodo	\`reserving.py\`, \`schemas.py\`
Nuova validazione	\`validation.py\`
Cambiare report	\`reporting.py\`
Nuovi chart	\`pages/3_■_AI_Team.py\`
Nuovi test diagnostici	\`selection.py\`
Cambiare LLM	\`llm_utils.py\`

Documento generato automaticamente dall'analisi del codice sorgente.