Contents

Dimensioni di Design e Trade-off Architetturali

Overview

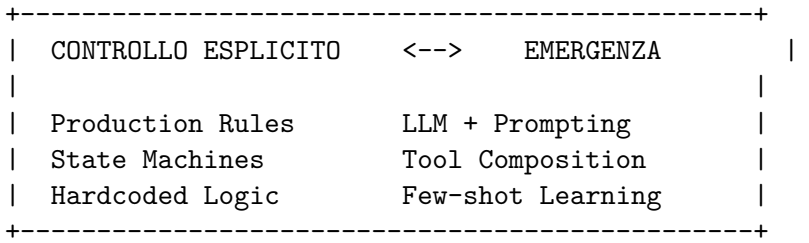
Ogni architettura agentica naviga uno spazio multi-dimensionale di trade-off. Non esistono scelte universalmente “migliori” - solo scelte appropriate per specifici contesti.

Questo documento analizza le **6 dimensioni fondamentali** e i relativi trade-off, fornendo decision framework per scelte architetturali informate.

Le 6 Dimensioni Fondamentali

1. Controllo vs Flessibilità

Definizione: Quanto comportamento è esplicitamente programmato vs emergente da capacità LLM.



Estremo: Controllo Massimo Approccio: - Ogni decisione da regole esplicite - State machines deterministiche - Planning algorithms formali (HTN, PDDL)

Vantaggi: - [OK] Comportamento completamente prevedibile - [OK] Facile formal verification - [OK] Debugging deterministico - [OK] Performance garantite

Svantaggi: - [NO] Rigido su input non previsti - [NO] Richiede engineering estensivo - [NO] Difficile generalizzazione - [NO] Costoso mantenimento

Quando Scegliere: - Safety-critical systems - Regulatory compliance richiesta - Domain molto ristretto e stabile - Zero tolleranza per errori

Estremo: Emergenza Massima Approccio: - Comportamento emerge da LLM reasoning - Minimal hardcoded logic - Prompting come primary control mechanism

Vantaggi: - [OK] Altissima flessibilità - [OK] Generalizzazione naturale - [OK] Adattamento a scenari nuovi - [OK] Minimal engineering overhead

Svantaggi: - [NO] Comportamento meno prevedibile - [NO] Difficult formal verification - [NO] Debugging di comportamenti emergenti difficile - [NO] Performance variabile

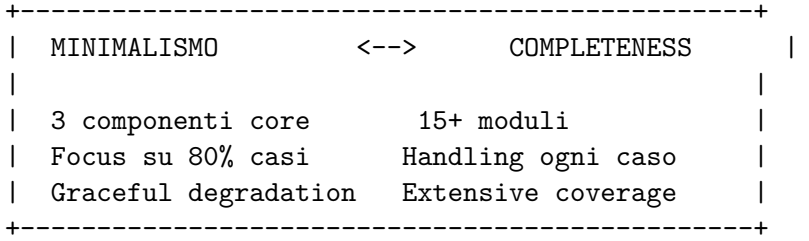
Quando Scegliere: - Domain ampi e variabili - Innovazione e creatività richieste - Best-effort acceptable - Rapida iterazione prioritaria

Zona Intermedia: Hybrid Approccio: - Core logic esplicita per critical paths - Emergenza per handling variabilità - Guardrails espliciti + flessibilità entro bounds

Trade-off Bilanciato: - Control where it matters - Flexibility where beneficial - Esempio: Planning framework esplicito + LLM per adaptive execution

2. Semplicità vs Completeness

Definizione: Numero di componenti e capacità architetturali.



Approccio Minimalista Filosofia: Identificare nucleo essenziale, lasciare resto emergere o degradare.

Architettura Tipo:

```
Core:
- Execution loop
- Memory (context + retrieval)
- Tool interface

Extensions added as needed
```

Vantaggi: - [OK] Facile comprensione - [OK] Rapido sviluppo - [OK] Basso overhead - [OK] Flessibile per modifiche

Svantaggi: - [NO] Coverage incompleta - [NO] Performance sub-ottimale per edge cases - [NO] Alcuni task potrebbero fallire

Quando Scegliere: - Rapid prototyping - Domains in evoluzione - 80% coverage sufficiente - Overhead importante bottleneck

Approccio Completo Filosofia: Design per ogni scenario possibile, extensive capabilities.

Architettura Tipo:

```
Components:
- Multiple reasoning strategies
- 4 memory types
- Metacognition layer
- Extensive error taxonomy
- Multiple planning modes
- Learning mechanisms
...
```

Vantaggi: - [OK] Coverage estensiva - [OK] Ottimizzazione per ogni caso - [OK] Robustezza a edge cases - [OK] Capabilities ricche

Svantaggi: - [NO] Alta complessità - [NO] Lento sviluppo - [NO] Difficile mantenimento - [NO] Overhead significativo

Quando Scegliere: - Production systems maturi - Domain ben compreso e stabile - Coverage completa richiesta - Risorse di sviluppo abbondanti

3. Generalità vs Specializzazione

Definizione: Ampiezza domain vs ottimizzazione per specifico.

```
+-----------------------------------------------+
|  GENERAL-PURPOSE     <-->    DOMAIN-SPECIFIC     |
|                                                |
|  Ogni task tipo         Un domain eccellente   |
|  Broad capability       Deep capability         |
```

```
|  Few-shot adaptation   Engineered excellence    |
+-----------------------------------------------+
```

General-Purpose Agent Approccio: - No assunzioni su domain specifico - Capabilities generiche e componibili - Adaptation via prompting e few-shot

Vantaggi: - [OK] Single system per molti use case - [OK] Adattabile a nuovi domain - [OK] Riutilizzo massimo

Svantaggi: - [NO] Performance sub-ottimale per ogni domain - [NO] Nessun advantage di specializzazione - [NO] Può essere "mediocre a tutto"

Quando Scegliere: - Multi-domain requirements - Domain in evoluzione - Unknow future use cases - Sviluppo single system preferito

Domain-Specific Agent Approccio: - Optimized per specific domain - Custom tools e knowledge - Specialized reasoning

Vantaggi: - [OK] Performance ottimale per domain - [OK] Domain expertise integrata - [OK] Efficient per use case target

Svantaggi: - [NO] Non generalizza fuori domain - [NO] Richiede multiple systems per domains diversi - [NO] Re-engineering per nuovo domain

Quando Scegliere: - Single domain chiaro - Performance critical - Domain stable - Risorse per specializzazione disponibili

4. Latenza vs Qualità

Definizione: Trade-off tra velocità di risposta e qualità output.

```
+-----------------------------------------------+
|  FAST/GOOD-ENOUGH   <-->    SLOW/EXCELLENT      |
|                                               |
|  Simple prompts         Extensive reasoning   |
|  Small models           Large models           |
|  Cached responses       Fresh generation       |
|  Single-pass            Multi-pass refinement  |
+-----------------------------------------------+
```

Trade-off Curve:

```
Quality
  |      --------
  |          (diminishing returns)
  |
  |
  | ______________ Latency

  Zone A: <2s, 70% quality - Interactive
  Zone B: 2-30s, 85% quality - Responsive
```

```
Zone C: 30s-5min, 95% quality - Batch
Zone D: >5min, 99% quality - Offline
```

Design Choices per Zone:

Zone A (Interactive): - Small/medium models - Simple prompting - Aggressive caching - Pre-computed responses when possible

Zone B (Responsive): - Large models acceptable - Moderate reasoning depth - Strategic caching - Parallel tool execution

Zone C (Batch): - Largest models - Extensive reasoning - Multi-pass refinement - Comprehensive validation

Zone D (Offline): - Multiple model calls - Tree exploration - Self-critique loops - Extensive verification

5. Cost vs Capability

Definizione: Trade-off tra costo operativo e capacità system.

```
+-----------------------------------------------+
|  LOW-COST          <-->      HIGH-CAPABILITY   |
|                                                |
|  Small models           Large models           |
|  Cached responses       Fresh generation       |
|  Simple tools           Rich toolset           |
|  Minimal memory         Extensive memory       |
+-----------------------------------------------+
```

Cost Drivers: 1. **Model Size**: Small ($0.001/1K) -> Large ($0.10/1K) = 100x 2. **Call Frequency**: Cached vs ogni volta = 10-100x 3. **Tool Usage**: Free vs paid APIs = Varia 4. **Memory Storage**: Context vs vector DB = Varia

Optimization Strategies:

Low-Cost Optimization: - Model routing (small per simple, large per complex) - Aggressive caching - Prompt optimization (fewer tokens) - Lazy loading capabilities

High-Capability Optimization: - Always largest/best model - Fresh generation ogni volta - Extensive toolset - Comprehensive memory

Balanced Approach: - Model routing basato su complexity detection - Cache for common patterns - Essential tools only - Tier

ed memory (hot/warm/cold)

6. Autonomy vs Human-in-Loop

Definizione: Grado di supervisione umana richiesta.

```
+-----------------------------------------------+
|  FULL-AUTO          <-->    HUMAN-SUPERVISED   |
|                                                |
```

```
|  No human needed        Human approval needed  |
|  End-to-end execution   Human verification     |
|  Self-correction        Human correction       |
+------------------------------------------------+
```

Spectrum:

Level 0: Full Autonomy - Agent decide ed esegue tutto - No approval gates - Self-recovery from errors - Esempio: Content generation

Level 1: Notify-Only - Agent agisce autonomamente - Human notificato post-facto - Human può rollback se needed - Esempio: Automated responses

Level 2: Approval Gates - Agent propone azioni - Human approva prima esecuzione - Critical actions only gated - Esempio: Code deployment

Level 3: Co-Pilot Mode - Agent suggerisce - Human decide sempre - Agent esegue su comando - Esempio: Medical diagnosis

Level 4: Tool-Only - Agent solo analizza - Human fa tutte decisioni ed azioni - Agent è research/analysis assistant - Esempio: Legal analysis

Design Implications:

Full Autonomy Requirements: - Alta affidabilità - Extensive testing - Error recovery robust - Logging per forensics

Human-in-Loop Requirements: - Clear approval interfaces - Explainable recommendations - Easy rollback - Audit trail

Framework di Decisione

Decision Matrix: Dimension -> Choice

Context	Dim 1	Dim 2	Dim 3	Dim 4	Dim 5	Dim 6
Startup MVP	Emergenza	Minimal	General	Responsive	Low-cost	Full-auto
Enterprise Prod	Hybrid	Complete	Specialized	Batch	Balanced	Approval
Safety-Critical	Control	Complete	Specialized	Variable	High-cap	Co-pilot
Research Tool	Emergenza	Minimal	General	Offline	High-cap	Tool-only

Step-by-Step Decision Process

```
1. Classificare problema (usa 01-problem-taxonomy.md)
   |
2. Identificare vincoli hard
   - Safety requirements -> Dim 1 (Control)
   - Latency requirements -> Dim 4 (Fast)
```

```
   - Budget constraints -> Dim 5 (Low-cost)
   |
3. Prioritizzare dimensioni
   - Quali sono critiche?
   - Quali sono nice-to-have?
   |
4. Per ogni dimensione, selezionare punto su spectrum
   - Considera trade-off
   - Verifica compatibilità cross-dimensional
   |
5. Validare coerenza
   - Low-cost + High-capability = Incoerente
   - Fast + Extensive reasoning = Incoerente
   |
6. Selezionare pattern (vedi 03-architecture-patterns.md)
```

Trade-off Multi-Dimensionali

Alcune scelte su una dimensione influenzano altre:

Trade-off Compatibility Matrix

	Controllo	Minimal	General	Fast	Low-cost	Auto
Controllo	-	[!]	[v]	[v]	[v]	[!]
Minimal	[!]	-	[v]	[v]	[v]	[v]
General	[v]	[v]	-	[!]	[v]	[!]
Fast	[v]	[v]	[!]	-	[v]	[v]
Low-cost	[v]	[v]	[v]	[v]	-	[v]
Auto	[!]	[v]	[!]	[v]	[v]	-

[v] = Naturalmente compatibile
[!] = Richiede design attento
[x] = Fondamentalmente incompatibile

Esempi di Incompatibilità

Fast + Extensive Reasoning: - [x] Impossibile ragionamento profondo in <2s - Resolution: Accettare latenza OR semplificare reasoning

Low-Cost + Always Fresh: - [x] Caching disabled = alto costo - Resolution: Accettare costo OR usare caching

Full-Auto + Safety-Critical: - [!] Rischioso senza human verification - Resolution: Add approval gates OR increase affidabilità drasticamente

Patterns Emergenti

Dalla analisi di sistemi esistenti, emergono **archetipi ricorrenti**:

Archetipo 1: "Lean Startup Agent"

- Emergenza + Minimal + General + Responsive + Low-cost + Auto
- **Use case**: MVP rapido, iterate velocemente
- **Trade-off accettato**: Coverage imperfetta, variabilità output

Archetipo 2: "Enterprise Production Agent"

- Hybrid + Complete + Specialized + Batch + Balanced + Approval
- **Use case**: Production critical ma non safety
- **Trade-off accettato**: Complessità sviluppo, costo moderato

Archetipo 3: "Safety-First Agent"

- Control + Complete + Specialized + Variable + High-cap + Co-pilot
- **Use case**: Medical, legal, financial critical
- **Trade-off accettato**: Rigidità, alto costo, latenza

Archetipo 4: "Research Assistant"

- Emergenza + Minimal + General + Offline + High-cap + Tool-only
- **Use case**: Supporto ricerca/analisi
- **Trade-off accettato**: Human fa decisioni, slow

Archetipo 5: "Real-Time Controller"

- Control + Minimal + Specialized + Fast + Balanced + Auto
- **Use case**: Robotics, real-time systems
- **Trade-off accettato**: Reasoning limitato, domain ristretto

Evoluzione nel Tempo

Architecture spesso evolve lungo dimensioni:

```
MVP -> Production
+- Minimal -> Complete (add capabilities as needed)
+- General -> Specialized (optimize for main use cases)
+- Emergenza -> Hybrid (add control for critical paths)
+- Low-cost -> Balanced (invest in reliability)

Production -> Maturity
+- Complete -> Streamlined (remove unused)
+- Specialized -> Re-generalized (new use cases)
+- Hybrid -> More control (lessons learned)
+- Approval -> More auto (trust built)
```

Anti-Patterns Comuni

Anti-Pattern: "Over-Control su Problem Semplice"

- Dimensioni: Full control + Complete per Classe A problem
- Sintomo: Months di development per chatbot semplice
- Correzione: Riconoscere che emergenza è sufficiente

Anti-Pattern: "Over-Emergence su Problem Critico"

- Dimensioni: Full emergenza per Classe C problem
- Sintomo: Comportamento imprevedibile in medical context
- Correzione: Add controlli e verifiche esplicite

Anti-Pattern: "Swiss Army Knife"

- Dimensioni: Tentare everything excellently
- Sintomo: Complex system che è mediocre a tutto
- Correzione: Specializzare OR accettare trade-off

Anti-Pattern: "Premature Optimization"

- Dimensioni: Complete + Specialized + High-cap per MVP
- Sintomo: Over-engineering prima di validation
- Correzione: Start minimal, evolve based on data

Metriche per Validazione Scelte

Per ogni dimensione, metriche appropriate:

Dim 1 (Controllo): % decisioni da regole vs LLM, predicibilità output **Dim 2 (Semplicità)**: # componenti, lines of code, time to understand **Dim 3 (Generalità)**: # domains supported, transfer learning success **Dim 4 (Latenza)**: p50/p95/p99 response time **Dim 5 (Costo)**: $ per task, $ per user/month **Dim 6 (Autonomy)**: % tasks needing human, time to human intervention

Conclusione

Non esistono scelte "giuste" universalmente - solo scelte appropriate per contesto.

Il framework di design dimensions permette: - [OK] Decisioni architetturali informate - [OK] Trade-off espliciti e comunicabili - [OK] Comparazione oggettiva alternative - [OK] Evoluzione incrementale chiara

Next: 03-architecture-patterns.md per pattern concreti per combinazioni dimensionali comuni.