

HT2 ACADEMY: VOLUME 1

# Intelligenza che lavora

Dal modello al sistema, dall'hype alla produzione

HT2 TEAM

Prima edizione — 2026



**Intelligenza che lavora**

*Dal modello al sistema, dall'hype alla produzione*

HT2 Team

Copyright © 2026 HT2 [www.ht2.it](http://www.ht2.it)

Tutti i diritti riservati.

Prima edizione: 2026

Composto in L<sup>A</sup>T<sub>E</sub>X con font Palatino.



# Ringraziamenti

Desidero esprimere la mia più profonda gratitudine a tutte le persone che hanno reso possibile la realizzazione di questo volume sull'Agentic AI. Un progetto di questa complessità non avrebbe mai visto la luce senza il contributo sinergico, la visione e la competenza tecnica di un team eccezionale.

## ALLA VISIONE STRATEGICA E TECNICA DI HT2

---

Un ringraziamento speciale va a **Michele Russo**, **Asya Pesaresi**, **Luca Fuligni** e **Paola Spinelli**, membri del Comitato Strategico di HT2, per aver fornito la direzione e il supporto necessari a esplorare le frontiere degli agenti autonomi sia dal punto di vista tecnico che organizzativo.

Allo stesso modo, la solidità scientifica di questo lavoro è frutto dell'impegno di **Natalia Miccini**, e **Dmitry Mingazov**, membri del Comitato Tecnico, il cui rigore nell'analisi delle architetture AI è stato il pilastro su cui si fonda ogni capitolo.

## ALLA NOSTRA COMMUNITY E AI NOSTRI PARTNER

---

Un riconoscimento sentito va alla Community di HT2 nella sua interezza. Siete voi, con il vostro entusiasmo e il continuo scambio di idee, a dare un senso concreto alla nostra ricerca e a spingerci a superare i limiti dell'innovazione.

Un ringraziamento fondamentale va inoltre a **DeepLearning.AI**. Siamo onorati di poter operare come ponte verso il loro straordinario ecosistema di corsi e know-how. Questa sinergia ci permette di democratizzare l'accesso a una formazione di livello mondiale e di trasferire le competenze più avanzate direttamente nel cuore dei nostri progetti.

Paolo Alessandroni  
*Presidente HT2 e Ambassador DeepLearning.AI*



# Prefazione

C'è un momento preciso in cui ogni progetto AI smette di essere entusiasmante e diventa un problema ingegneristico. Il proof-of-concept ha funzionato. Il board è convinto. E poi arriva la produzione: ventimila documenti invece di venti, casi limite che nessuno aveva previsto, un carico reale che non somiglia a nessun benchmark. E il sistema, lentamente o di colpo, smette di reggere.

L'abbiamo visto accadere più volte di quante vorremmo ammettere. E abbiamo smesso di credere che la soluzione fosse più tecnologia. La soluzione era capire meglio *quale* tecnologia, *quando*, e *perché*.

Il panorama dell'AI applicata è cambiato in modo radicale negli ultimi tre anni. Non nella direzione che tutti si aspettavano: non è arrivata l'intelligenza generale, non si è smesso di parlare di allucinazioni, non è sparita la complessità ingegneristica. Quello che è cambiato è la *praticabilità*: oggi è possibile costruire sistemi che ragionano, decidono e agiscono su problemi reali, con un impatto misurabile sul business, senza un laboratorio di ricerca alle spalle.

Ma praticabile non significa semplice. Significa che la complessità si è spostata: dal *funziona?* al *regge in produzione?* Dal *è possibile?* al *vale la pena?* Dal *abbiamo un modello?* al *abbiamo un sistema?*

Questo libro affronta quella complessità spostata. Non spiega come funziona un transformer. Non è un tutorial su LangChain. È una bussola per chi deve prendere decisioni architetturali e sa che sbagliare quella decisione costa tempo, denaro e credibilità.

L'arco narrativo segue una progressione che abbiamo visto ripetersi in molti progetti: si parte dal fine-tuning — la prima cosa che chiedono i clienti, spesso per le ragioni sbagliate — e si risale fino ai sistemi multi-agente e ai protocolli di standardizzazione che stanno ridisegnando come questi sistemi si connettono al mondo. Ogni capitolo è costruito sulla stessa logica: capire il problema prima della soluzione, distinguere i casi in cui uno strumento funziona da quelli in cui non funziona, portare numeri reali invece di promesse.

Troverete casi di successo e fallimenti — alcuni dei quali ci hanno fatto perdere mesi — raccontati con la stessa franchezza. Troverete architetture che funzionano in produzione e pattern che sembrano ragionevoli sulla carta ma si rompono sotto carico. Troverete opinioni, perché un libro senza opinioni basate sull'esperienza non dice nulla di utile.

Questo lavoro nasce dal contributo di un team che costruisce questi sistemi ogni giorno, si fa le domande difficili, e ha la rara capacità di tenere insieme rigore tecnico e senso pratico. È anche il frutto della collaborazione con la nostra community e con DeepLearning.AI, che ci permette di lavorare alla frontiera e riportare quello che impariamo in forma accessibile.

C'è però qualcosa che va detto esplicitamente, perché riguarda il perché questo libro esiste. HT2 ([www.ht2.it](http://www.ht2.it)) nasce nel 2016 con un'idea semplice e ambiziosa insieme: contrastare la fuga di talenti dall'Italia costruendo un ecosistema in cui merito, trasparenza e collaborazione non sono valori decorativi ma criteri operativi. In quasi dieci anni, quella scommessa ha prodotto una community viva, un network che attraversa industria, accademia e impresa, e una pratica quotidiana fondata su apertura della conoscenza e impatto sociale misurabile.

L'AI agentica non è estranea a questi valori: li mette alla prova. Un sistema che decide e agisce in

---

modo autonomo pone domande concrete su trasparenza, responsabilità e controllo che non si risolvono con una policy aziendale né con un disclaimer legale. Si risolvono nelle scelte architettoniche: come si progettano i guardrail, come si costruisce l'audit trail, a chi si dà il potere di interrompere un processo automatizzato. Questo libro cerca di essere onesto su queste domande, non di aggirarle. Perché costruire sistemi AI potenti senza costruire sistemi AI affidabili non è progresso: è semplicemente rischio spostato in avanti.

A chi legge: non cercate qui la risposta definitiva su cosa fare. Cercate gli strumenti per arrivare alla risposta giusta per il vostro contesto. Sapere qual è la domanda giusta vale più di qualsiasi risposta preconfezionata.

*Milano, 2026*

Comitato Strategico HT2 ([www.ht2.it](http://www.ht2.it))

# Contents

<b>Ringraziamenti</b>	<b>v</b>
<b>Prefazione</b>	<b>vii</b>
<b>1 Fine-Tuning: potenzialità, limiti e il momento in cui non basta</b>	<b>1</b>
1.1 Che cos'è il fine-tuning e perché tutti ne parlano . . . . .	2
1.1.1 Dal pre-addestramento al fine-tuning: la meccanica del processo . . . . .	2
1.1.2 Il fascino della personalizzazione: cosa si aspettano i clienti . . . . .	3
1.1.3 Cosa si ottiene davvero — e cosa no . . . . .	3
1.2 Le tecniche disponibili oggi: cosa scegliere e perché . . . . .	5
1.2.1 Full Fine-Tuning: perché quasi nessuno lo fa più in produzione . . . . .	5
1.2.2 LoRA e QLoRA: l'efficienza che ha cambiato il gioco . . . . .	6
1.2.3 Adapter e Prefix Tuning: modulari per design . . . . .	7
1.2.4 Instruction Tuning e RLHF: insegnare al modello a rispondere come vogliamo . . . . .	7
1.2.5 Fine-tuning su dati sintetici: generare il corpus con un altro modello . . . . .	8
1.2.6 Continued Pre-Training: quando il dominio richiede di ripartire dalle fondamenta . . . . .	9
1.2.7 Guida alla scelta: scenario ideale, costo, complessità . . . . .	10
1.3 Il vero costo: molto oltre il training . . . . .	11
1.3.1 Il costo nascosto dei dati: il vero collo di bottiglia . . . . .	11
1.3.2 Model drift: il fine-tuning non è un evento, è un processo . . . . .	12
1.3.3 Il confronto economico: fine-tuning vs. prompt engineering vs. RAG . . . . .	13
1.4 Quando ha perfettamente senso fare fine-tuning . . . . .	15
1.4.1 Adattamento di stile e tono: quando la brand voice è non negoziabile . . . . .	15
1.4.2 Task strutturati con output predicibile . . . . .	16
1.4.3 Quando la latenza è critica e il modello deve sapere, non cercare . . . . .	16
1.4.4 Ambienti air-gapped e on-premise: il fine-tuning come unica strada . . . . .	17
1.4.5 Albero decisionale: fine-tuning sì o no? . . . . .	17
1.5 I limiti strutturali: perché non scala sulla conoscenza . . . . .	18
1.5.1 Il congelamento della conoscenza al momento del training . . . . .	18
1.5.2 Overfitting e allucinazioni: il fine-tuning spesso peggiora il problema . . . . .	19
1.5.3 Il catastrophic forgetting: il costo nascosto della specializzazione . . . . .	20
1.5.4 Il paradosso della specificità e il punto di rottura . . . . .	21
1.6 Storie dal campo: successi, fallimenti e lezioni . . . . .	22
	<b>ix</b>

1.6.1	Il successo che si è trasformato: fine-tuning per assistenza clienti B2B . . . . .	22
1.6.2	Il modello specializzato per contrattualistica legale: ROI e limiti inattesi . . . . .	23
1.6.3	Il fallimento istruttivo: sei mesi persi e una lezione da 300.000 euro . . . . .	24
1.6.4	I pattern che si ripetono: una guida per non rifare gli stessi errori . . . . .	25
<b>2</b>	<b>RAG: recuperare conoscenza, generare valore</b>	<b>27</b>
2.1	L'idea fondamentale: separare chi sa da chi ragiona . . . . .	28
2.1.1	La struttura di base: cerca, arricchisci, genera . . . . .	28
2.1.2	Perché il RAG è diventato il pattern dominante . . . . .	29
2.2	Come funziona un sistema RAG: i pezzi del puzzle . . . . .	31
2.2.1	L'ingestion pipeline: come i documenti diventano materiale utilizzabile . . . . .	31
2.2.2	Il parsing: il primo punto di perdita dell'informazione . . . . .	32
2.2.3	Il chunking: tagliare senza spezzare il senso . . . . .	32
2.2.4	L'estrazione dei metadati: il contesto che il retrieval non può dedurre . . . . .	33
2.2.5	Embedding e vector store: trasformare il testo in numeri ricercabili . . . . .	34
2.2.6	La scelta del modello di embedding: quello che i benchmark non dicono . . . . .	35
2.2.7	Criteri pratici per scegliere il vector store . . . . .	36
2.2.8	Il retrieval: andare oltre la semplice similarità . . . . .	37
2.2.9	La generazione: come il contesto recuperato diventa risposta . . . . .	39
2.3	RAG avanzato: quando la pipeline lineare non basta . . . . .	41
2.3.1	Il RAG agentic: quando il retriever deve iterare . . . . .	41
2.3.2	Graph RAG: quando le relazioni contano più dei frammenti . . . . .	44
2.3.3	RAG multi-modale: documenti che non sono solo testo . . . . .	45
2.3.4	Il retrieval selettivo: il modello che decide quando cercare . . . . .	45
2.3.5	Il pattern di verifica: controllare il retrieval prima di generare . . . . .	46
2.4	Il muro tra la demo e la produzione . . . . .	47
2.4.1	Il problema della scala: da 10 a 10.000 documenti . . . . .	47
2.4.2	Il chunking che spezza il senso . . . . .	48
2.4.3	L'embedding che non capisce il vostro gergo . . . . .	50
2.4.4	Il retrieval che porta contesto irrilevante o contraddittorio . . . . .	50
2.4.5	Il modello che ignora il contesto e si inventa la risposta . . . . .	51
2.4.6	Il gap di ingegnerizzazione: tutto quello che nella demo non serviva . . . . .	52
2.5	Misurare il valore: metriche che parlano al business . . . . .	55
2.5.1	Le metriche tecniche essenziali . . . . .	55
2.5.2	Le metriche di business: quelle che il board capisce . . . . .	56
2.5.3	Come calcolare il ROI di un sistema RAG . . . . .	57
2.5.4	I framework di valutazione: come costruire una pipeline di evaluation . . . . .	58

---

2.6	RAG per settore: cosa cambia a seconda del dominio . . . . .	60
2.6.1	Legal e compliance: il vincolo della precisione assoluta . . . . .	60
2.6.2	Healthcare: il vincolo della safety . . . . .	61
2.6.3	Financial services: il vincolo della freschezza . . . . .	62
2.6.4	Manufacturing: precisione tecnica e knowledge base operativa . . . . .	63
2.6.5	Pattern trasversali e anti-pattern universali . . . . .	64
2.7	Storie dal campo: cosa abbiamo imparato . . . . .	66
2.7.1	Storia 1 — Un RAG per knowledge base aziendale: dal PoC alla produzione . . . . .	66
2.7.2	Storia 2 — Un RAG multi-sorgente per customer support B2B: la complessità che non avevamo previsto . . . . .	68
2.7.3	Storia 3 — Il RAG che non doveva essere un RAG . . . . .	69
2.7.4	La tabella decisionale: fine-tuning, RAG, o approccio ibrido? . . . . .	70
<b>3</b>	<b>Dal RAG all'Agente: quando il recupero non basta, serve l'azione</b>	<b>73</b>
3.1	I limiti del RAG puro: sa rispondere ma non sa agire . . . . .	74
3.1.1	Il pattern read-only e i suoi limiti concreti . . . . .	74
3.1.2	Dove il RAG tiene e dove cede: una mappa onesta . . . . .	77
3.1.3	Informare vs. operare: la distinzione che cambia tutto . . . . .	78
3.1.4	Il momento di rottura: quando il cliente chiede l'azione . . . . .	80
3.1.5	I vincoli che non spariscono con l'agente . . . . .	81
3.2	Cos'è un agente, in pratica . . . . .	82
3.2.1	Il ciclo fondamentale: osserva, ragiona, agisci, osserva . . . . .	82
3.2.2	Agente vs. pipeline vs. chatbot: le distinzioni che contano . . . . .	85
3.2.3	I livelli di autonomia: dallo strumento al decisore . . . . .	87
3.2.4	Come i principali provider definiscono un agente . . . . .	90
3.2.5	Le proprietà che contano davvero in produzione . . . . .	91
3.2.6	Perché la terminologia è confusa — e perché conta poco . . . . .	94
3.3	Tool calling: come l'agente interagisce con il mondo reale . . . . .	95
3.3.1	Come funziona una tool call: il flusso end-to-end . . . . .	95
3.3.2	L'arte di definire un tool: naming, descrizione, parametri . . . . .	97
3.3.3	Granularità dei tool: né troppo generici né troppo specifici . . . . .	99
3.3.4	Gestione degli errori: progettare il fallimento . . . . .	100
3.3.5	Tool hallucination: quando il modello inventa dati strutturati . . . . .	103
3.3.6	Sicurezza nel tool calling: il principio del minimo privilegio . . . . .	105
3.3.7	Sandboxing e isolamento: proteggere l'ambiente di esecuzione . . . . .	107
3.3.8	Rate limiting, tetti di costo e protezione dagli abusi . . . . .	108
3.3.9	Tre tool concreti: dal design all'implementazione . . . . .	110

3.4	Guardrail e controllo: autonomia non significa anarchia . . . . .	116
3.4.1	Il principio fondamentale: minimo potere necessario . . . . .	116
3.4.2	Guardrail sull’input: classificazione, PII, prompt injection . . . . .	117
3.4.3	Guardrail durante l’esecuzione: permessi, rate limiting, tetti operativi . . . . .	120
3.4.4	Il pattern “chiedi conferma”: quando e come senza distruggere l’usabilità . . . . .	123
3.4.5	Guardrail sull’output: faithfulness, format validation, data leakage . . . . .	127
3.4.6	Logging e audit trail: il requisito che non si può negoziare . . . . .	128
3.4.7	Costruire fiducia incrementale: il deployment progressivo dei guardrail . . . . .	133
3.5	Come si valuta un agente: le metriche che contano . . . . .	134
3.5.1	Il problema della misurazione: perché valutare un agente è diverso . . . . .	134
3.5.2	Task completion rate: la metrica regina e le sue tre facce . . . . .	135
3.5.3	Latenza e costo per task: il trade-off che determina la sostenibilità . . . . .	137
3.5.4	Tool selection accuracy: misurare se il modello sceglie bene . . . . .	139
3.5.5	Consistenza e affidabilità: la varianza che nessuno misura . . . . .	141
3.5.6	Perché i benchmark dei paper non predicono la produzione . . . . .	144
3.5.7	Costruire una eval pipeline per agenti: dataset, automazione, regressione . . . . .	145
3.5.8	Tradurre le metriche in decisioni: il dashboard operativo . . . . .	149
3.6	Architetture di agente singolo: i pattern che funzionano . . . . .	151
3.6.1	Il router agent: classifica, smista, non fa nulla da solo . . . . .	151
3.6.2	Il RAG agent: retrieval, ragionamento e azione in sequenza . . . . .	155
3.6.3	Il code agent: generare ed eseguire codice come forma d’azione . . . . .	157
3.6.4	L’agente conversazionale con stato: memoria della sessione . . . . .	160
3.6.5	L’agente reattivo-proattivo: dal task pull al task push . . . . .	163
3.6.6	Quando usare quale pattern: la guida alla scelta . . . . .	164
3.7	Casi d’uso: dall’idea al valore di business . . . . .	167
3.7.1	Customer service agent: oltre il chatbot . . . . .	167
3.7.2	Sales assistant agent: qualificazione, proposta, scheduling . . . . .	169
3.7.3	Agente operativo interno: onboarding, IT helpdesk, procurement . . . . .	171
3.7.4	Le lezioni trasversali: pattern e anti-pattern . . . . .	172
<b>4</b>	<b>Architetture Multi-Agente: dividere per conquistare</b>	<b>175</b>
4.1	Perché un singolo agente non scala . . . . .	176
4.1.1	Il problema della context window . . . . .	176
4.1.2	Specializzazione vs. generalismo: il paradosso dell’agente tuttofare . . . . .	178
4.1.3	Oltre una certa complessità, le istruzioni si contraddicono . . . . .	179
4.1.4	Il parallelo organizzativo: perché le aziende hanno dipartimenti . . . . .	181
4.1.5	Quando il singolo agente ha senso — e quando no . . . . .	183

---

4.2	I pattern architetturali: il catalogo pratico . . . . .	184
4.2.1	Supervisor — un agente orchestra, gli altri eseguono . . . . .	184
4.2.2	Handoff — gli agenti si passano il controllo . . . . .	187
4.2.3	Router — classificazione dell'intent e dispatching . . . . .	189
4.2.4	Swarm / Skills — agenti come funzioni componibili . . . . .	191
4.2.5	Gerarchico — supervisor di supervisor . . . . .	193
4.2.6	Tabella comparativa e criteri di scelta . . . . .	196
4.3	Orchestrazione: chi comanda, chi esegue, chi supervisiona . . . . .	198
4.3.1	Il ruolo dell'orchestratore: infrastruttura, non intelligenza . . . . .	198
4.3.2	I framework di orchestrazione: confronto basato su esperienza reale . . . . .	200
4.3.3	Paradigmi di orchestrazione: state machine, graph, event-driven . . . . .	203
4.3.4	Il problema delle condizioni di terminazione . . . . .	206
4.3.5	Gestione degli errori: retry, fallback, circuit breaker . . . . .	208
4.3.6	Idempotenza e checkpoint: costruire flussi che reggono gli errori . . . . .	210
4.4	Come gli agenti condividono informazione . . . . .	212
4.4.1	Pattern di condivisione: blackboard, message passing, memoria condivisa . . . . .	212
4.4.2	Memoria a livelli: working, short-term, long-term . . . . .	215
4.4.3	Il problema della consistenza: quando due agenti non sono d'accordo . . . . .	218
4.4.4	Gestione pratica della context window: iniezione selettiva e compressione . . . . .	219
4.5	Quando le cose vanno storto: fallback, loop e resilienza . . . . .	222
4.5.1	Il catalogo dei failure mode: riconoscerli prima che si manifestino . . . . .	222
4.5.2	Resilienza by design: costruire sistemi che degradano in modo controllato . . . . .	224
4.5.3	Chaos engineering applicato ai sistemi agentici . . . . .	227
4.5.4	Il drift silenzioso: rilevarlo prima che diventi un problema . . . . .	229
4.6	Misurare un sistema multi-agente in produzione . . . . .	231
4.6.1	Osservabilità distribuita: tracciare ogni decisione attraverso tutti gli agenti . . . . .	231
4.6.2	Tracciabilità delle decisioni: audit trail e compliance . . . . .	234
4.6.3	Le metriche operative e semantiche che contano davvero . . . . .	235
4.7	Human-in-the-loop: dove l'uomo deve restare nel circuito . . . . .	238
4.7.1	Il framework di autonomia: cinque livelli tra automazione e controllo umano . . . . .	238
4.7.2	Quando scatta l'escalation: i trigger che non possono essere lasciati al caso . . . . .	240
4.7.3	Progettare l'interfaccia HITL: la decisione si prende nei secondi che contano . . . . .	241
4.7.4	Il paradosso dell'automazione e come gestirlo . . . . .	243
4.7.5	Misurare e ottimizzare il bilanciamento HITL . . . . .	244
4.8	Casi reali: sistemi multi-agente in produzione . . . . .	245
4.8.1	E-commerce: il customer journey automatizzato . . . . .	245
4.8.2	Legal: due diligence automatizzata con HITL obbligatorio . . . . .	246

4.8.3	Operations: procurement e supply chain con architettura event-driven . . . . .	248
<b>5</b>	<b>Protocolli e Standardizzazione: MCP, A2A e il futuro interoperabile</b>	<b>251</b>
5.1	Il problema dell'integrazione a scala . . . . .	252
5.1.1	Da $O(N \times M)$ a $O(N+M)$ : l'analogia USB . . . . .	253
5.1.2	Il costo reale dell'integrazione custom: numeri da progetti reali . . . . .	254
5.1.3	Lo stato attuale dell'ecosistema: frammentato ma in convergenza . . . . .	255
5.1.4	Perché la standardizzazione viene dopo gli agenti — e non prima . . . . .	256
5.2	Model Context Protocol (MCP): come funziona e a cosa serve . . . . .	257
5.2.1	L'architettura Host → Client → Server: tre ruoli, tre responsabilità . . . . .	258
5.2.2	I tre pilastri: Tools, Resources, Prompts — differenze che contano . . . . .	260
5.2.3	Come funziona una connessione MCP: il ciclo di vita completo . . . . .	261
5.2.4	Trasporto: stdio vs HTTP/SSE — la scelta che impatta il deployment . . . . .	264
5.2.5	Sicurezza in MCP: il protocollo standardizza la comunicazione, non la sicurezza	265
5.2.6	L'ecosistema MCP: registry, community, e come valutare un server . . . . .	267
5.3	MCP in produzione: problemi veri e soluzioni collaudate . . . . .	268
5.3.1	Il gap tra la specifica e il deploy reale: come diventare un client difensivo . . .	268
5.3.2	Latenza dei round-trip: misurare prima di ottimizzare . . . . .	270
5.3.3	Gestione degli errori: retry, circuit breaker, fallback gerarchico . . . . .	272
5.3.4	Versioning e breaking changes: convertire i failure silenziosi in rumori . . . . .	274
5.3.5	Rate limiting e costi: il budget dei tool call . . . . .	275
5.3.6	MCP e sicurezza enterprise: dati sensibili, compliance, e tracciabilità . . . . .	276
5.3.7	Testing pre-deploy: la checklist che nessuno vuole fare e tutti rimpiangono . .	277
5.4	Agent-to-Agent (A2A): quando gli agenti parlano tra loro . . . . .	278
5.4.1	La distinzione fondamentale: MCP vs A2A — strumenti vs agenti . . . . .	278
5.4.2	Agent Card: il biglietto da visita che rende possibile la discovery . . . . .	279
5.4.3	Il ciclo di vita del task: dallo submitted al completed . . . . .	281
5.4.4	Gestione dei task asincroni: SSE, webhook, e reconnect . . . . .	283
5.4.5	Collaborazione cross-organizzazione: fiducia, responsabilità, audit . . . . .	284
5.4.6	A2A e MCP insieme: l'architettura a tre livelli . . . . .	286
5.5	Il caso verticale: e-commerce agentico e Universal Commerce Protocol . . . . .	287
5.5.1	Perché non basta MCP generico per l'e-commerce . . . . .	287
5.5.2	UCP: architettura e le quattro aree funzionali . . . . .	288
5.5.3	Il checkout come macchina a stati: zero ambiguità per l'agente . . . . .	289
5.5.4	Un flusso di acquisto agentico end-to-end . . . . .	291
5.5.5	Le implicazioni strutturali: pricing, discovery, intermediazione . . . . .	292
5.6	Come scegliere: confronto e roadmap . . . . .	293

5.6.1	Matrice di confronto su otto dimensioni . . . . .	293
5.6.2	Decision framework: cinque domande per arrivare alla scelta giusta . . . . .	294
5.6.3	Il pattern combinato MCP + A2A: l'architettura che scala . . . . .	296
5.6.4	Il panorama ad aprile 2026: chi ha scommesso su cosa . . . . .	296
5.6.5	Convergenza o frammentazione? I prossimi 12-24 mesi . . . . .	297
5.6.6	Roadmap pratica di adozione: quattro fasi per iniziare senza scommettere tutto	297
<b>6</b>	<b>Frontiere: ragionamento avanzato, memoria persistente e auto-miglioramento</b>	<b>301</b>
6.1	Ragionamento che evolve: oltre il "pensa e agisci" . . . . .	302
6.1.1	Planning gerarchico: scomporre prima di agire . . . . .	302
6.1.2	Esplorazione di alternative e ragionamento multi-step con verifica . . . . .	303
6.1.3	Il costo computazionale del ragionamento avanzato e quando vale la spesa . . .	305
6.2	Memoria: il pezzo che manca . . . . .	306
6.2.1	I quattro tipi di memoria che un agente ha bisogno . . . . .	306
6.2.2	Come si implementa la memoria oggi: vector store, compression, entity tracking	308
6.2.3	Il problema della memoria che degrada: obsolescenza, contraddizioni, rumore	309
6.2.4	Dove sta andando la ricerca: memoria come componente di prima classe . . .	310
6.3	Agenti che imparano dai propri errori . . . . .	311
6.3.1	Apprendimento dentro la sessione: critica e autocorrezione . . . . .	311
6.3.2	Apprendimento dall'ambiente: il feedback come segnale . . . . .	312
6.3.3	Il trade-off stabilità-plasticità . . . . .	313
6.4	Allineamento e controllo nei sistemi agentici . . . . .	313
6.4.1	Le tre dimensioni: trust, scope, control . . . . .	314
6.4.2	Il rischio del goal drift: quando l'agente ottimizza la cosa sbagliata . . . . .	315
6.4.3	Il quadro normativo emergente: EU AI Act e implicazioni pratiche . . . . .	316
6.5	Il futuro che è già qui . . . . .	316
6.5.1	Agenti che scrivono codice: cosa funziona davvero . . . . .	317
6.5.2	Agenti che gestiscono agenti: meta-orchestrazione e sistemi auto-organizzanti	317
6.5.3	Agenti come interfaccia: il futuro ibrido . . . . .	318
6.5.4	Un timeline realistico e onesto: 1, 3, 5 anni . . . . .	319
	<b>Appendici</b>	<b>320</b>
<b>A</b>	<b>Chi siamo, cosa costruiamo, cosa abbiamo imparato</b>	<b>321</b>
A.1	HT2: da dove nasce questo libro . . . . .	321
A.1.1	Una community . . . . .	321
A.1.2	Come lavoriamo: tre livelli di intervento . . . . .	322
A.1.3	I valori, senza retorica . . . . .	322

A.2	HT2 Academy: formare chi costruisce — learning by doing . . . . .	323
A.2.1	Il problema che risolviamo nella formazione . . . . .	323
A.2.2	Il percorso . . . . .	323
A.2.3	Perchè ne parliamo in questo libro . . . . .	323
A.3	Diet: un agente dietetico tra vincoli clinici e generazione creativa . . . . .	324
A.3.1	Il problema . . . . .	324
A.3.2	L’architettura: cinque nodi, una decisione chiave . . . . .	324
A.3.3	Il gate di validazione . . . . .	324
A.3.4	Cosa ci ha insegnato . . . . .	325
A.4	GenBot: intelligenza conversazionale multi-agente per l’enterprise . . . . .	325
A.4.1	Il problema . . . . .	325
A.4.2	L’architettura: perché un singolo agente non bastava . . . . .	325
A.4.3	La separazione tra modello e dati . . . . .	326
A.4.4	RBAC: la governance come architettura, non come feature . . . . .	326
A.4.5	Multimodalità come requisito operativo . . . . .	326
A.4.6	Cosa ci ha insegnato . . . . .	326
A.5	DeepBot: quando servono resilienza e scala . . . . .	326
A.5.1	Il problema . . . . .	326
A.5.2	La scelta fondamentale: architettura distribuita e asincrona . . . . .	327
A.5.3	I componenti . . . . .	327
A.5.4	Scalabilità orizzontale vs. verticale . . . . .	327
A.5.5	Sapere quando non decidere . . . . .	327
A.5.6	Cosa ci ha insegnato . . . . .	328
A.6	GenSAD: quando l’LLM lavora dietro le quinte . . . . .	328
A.6.1	Il problema . . . . .	328
A.6.2	L’architettura: tre layer, un principio . . . . .	328
A.6.3	L’LLM come componente di pipeline, non come interfaccia . . . . .	329
A.6.4	La dimensione temporale come asse architetturale . . . . .	329
A.6.5	Cosa ci ha insegnato . . . . .	329