

Intelligenza artificiale per ingegneri

Transformer, LLM, RAG e agenti AI: dai fondamenti al codice

Luciano Salemmme

Copyright

© 2026 Intelligenza artificiale per ingegneri - Luciano Salemme

Tutti i diritti riservati.

Nessuna parte di questa pubblicazione può essere riprodotta, memorizzata o trasmessa in qualsiasi forma o con qualsiasi mezzo, elettronico, meccanico, fotocopia, registrazione o altro, senza il previo consenso scritto dell'autore, salvo brevi citazioni a scopo di recensione o studio.

Le informazioni contenute in questo libro sono fornite a scopo didattico e informativo. L'autore non si assume alcuna responsabilità per eventuali danni derivanti dall'uso delle tecniche, del codice o dei progetti descritti.

Contatti dell'autore:

luciano.salemme@gmail.com

Prefazione

Qualche tempo fa ho letto un articolo in cui un modello linguistico veniva definito uno *stochastic parrot*, un pappagallo stocastico. Incuriosito dall'espressione, sono andato a cercarne l'origine: viene da un articolo scientifico del 2021, *On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?*, presentato alla conferenza ACM FAccT. Il 2021 apparteneva ancora all'era precedente a ChatGPT, e allora la definizione aveva il suo perché: i modelli dell'epoca erano molto meno capaci di quelli di oggi, e la metafora coglieva un aspetto reale del loro funzionamento. Secondo questa lettura un modello linguistico non ragiona: ricombina ciò che ha già visto, e può produrre frasi plausibili e coerenti senza possedere nulla che assomigli alla comprensione umana del significato.

La metafora è sopravvissuta ai modelli che descriveva, e oggi la si ritrova, insieme alla variante “in fondo è solo un autocompletamento”, in quasi ogni discussione sull'intelligenza artificiale. Il problema è che descrive il compito, non il processo interno che serve a svolgerlo. Si prenda la frase “il bicchiere è caduto dal tavolo perché...”. Per completarla bene, anche in contesti mai incontrati durante l'addestramento, il modello deve essersi costruito una rappresentazione almeno approssimativa del bicchiere come oggetto fisico, del tavolo come superficie d'appoggio, della caduta come movimento verso il basso e della gravità come causa possibile. Senza queste approssimazioni interne, prevedere la parola giusta in una situazione nuova resterebbe poco più che un tiro di dadi fortunato.

All'estremo opposto prospera la caricatura contraria: modelli che “capiscono”, “pensano”, “creano”, menti digitali dietro un'interfaccia di chat. Entrambe le rappresentazioni contengono una parte di verità, ed entrambe, prese alla lettera, portano fuori strada. Un modello addestrato con un obiettivo semplice, prevedere il token successivo, sviluppa al suo interno strutture sorprendentemente ricche; lo stesso modello può risolvere un problema complesso e fallire su una sua variazione banale, scrivere codice funzionante e un momento dopo inventare una libreria che non esiste. In questa zona grigia, tra il pappagallo e la mente, nasce gran parte della confusione

che circonda l'intelligenza artificiale contemporanea. E la via d'uscita, come sempre in ingegneria, non è scegliere una fazione: è aprire la scatola e capire come funziona.

È facile usare questi strumenti seguendo le istruzioni: si scrive un prompt, arriva una risposta, e tutto sembra funzionare. Ma è molto più gratificante, e professionalmente più utile, capire che cosa accade dietro quella risposta. La differenza tra un prototipo brillante e un sistema affidabile sta tutta nelle domande: perché il modello ha risposto così? quanto costerà tra sei mesi? che cosa succede quando sbaglia? A queste domande non si risponde con le opinioni, ma con i meccanismi e con le misure, ed è la ragione per cui questo libro parte dalle fondamenta e non si ferma prima della produzione.

Il percorso è organizzato in quattro parti, pensate per essere lette in sequenza. Nella prima si smonta il modello fino alle fondamenta: la matematica strettamente necessaria, le reti neurali e il loro addestramento, i token e gli embedding, il meccanismo di attention eseguito numero per numero, l'architettura Transformer ricostruita pezzo per pezzo, fino alla strada che trasforma un modello appena addestrato in un assistente che dialoga. Nella seconda si costruisce: le API e i loro parametri, i prompt e il contesto trattati da ingegneri, la ricerca semantica e i vector database, i sistemi RAG che rispondono citando le fonti. Nella terza il modello smette di limitarsi a rispondere e comincia ad agire: gli agenti, il Model Context Protocol, i sistemi in cui più agenti cooperano. La quarta parte affronta ciò che separa un prototipo da un sistema in esercizio: produzione, costi, sicurezza e normativa, osservabilità e valutazione, fino a un caso di studio completo che riunisce tutti i pezzi in un sistema funzionante.

Il testo è rivolto a:

- ingegneri del software e sviluppatori che vogliono costruire sistemi basati sui modelli linguistici partendo dai fondamenti;
- tech lead e architetti che devono prendere decisioni di progetto, di costo e di rischio su questa tecnologia;
- studenti di ingegneria e informatica che cercano un percorso organico tra la teoria e i sistemi reali;
- professionisti tecnici che usano quotidianamente questi strumenti e non si accontentano più di usarli alla cieca.

Il metodo non cambia da qui all'ultima pagina: ogni concetto ostico viene accompagnato da esempi numerici svolti passo dopo passo, con tutti i conti in

vista, e da codice breve, commentato e realmente eseguibile. Nella prima parte una rete neurale di tredici parametri viene calcolata interamente a mano, e l’attention viene eseguita su una frase di tre token fino all’ultima cifra; se i numeri di un libro non si possono verificare, ho sempre pensato, non c’è motivo di fidarsi delle sue conclusioni. Dove la materia è ancora aperta, il testo distingue con cura ciò che sappiamo, ciò che osserviamo sperimentalmente e ciò che resta oggetto di ricerca. Per affrontare la lettura bastano una conoscenza di base di Python e l’algebra delle scuole superiori: tutti gli strumenti matematici necessari vengono introdotti progressivamente, senza dare nulla per scontato.

Una domanda, alla fine, resterà aperta: che cosa significhi davvero “comprendere”, e se un sistema capace di rappresentare concetti, generalizzare e risolvere problemi nuovi meriti ancora il nome di pappagallo. Questo libro non pretende di rispondere; pretende qualcosa di più concreto, cioè darti gli strumenti per affrontare la domanda con cognizione di causa. Perché prima di chiederci se una macchina sia intelligente, dobbiamo essere sicuri di aver capito come funziona.

Buona lettura e buon lavoro: il mestiere di far funzionare queste macchine è appena cominciato.

Un’ultima nota pratica. Tutto il codice del libro, organizzato per capitolo e pronto da eseguire con le istruzioni di installazione riportate nel README, è disponibile in un repository pubblico:

<https://github.com/LucianoSalemme/intelligenza-artificiale-per-ingegneri>

Se trovi un errore o hai un’idea per migliorare un esempio, segnalalo direttamente sul repository: sarà un contributo prezioso per chi leggerà dopo di te.

Luciano Salemme
Roma, 2026

Indice

I	Fondamenti: capire cosa c'è dentro	1
1	Nascita e sviluppo dell'intelligenza artificiale	2
1.1	Da Turing a Dartmouth: nascita di un'idea	3
1.2	Inverni e primavere dell'AI	4
1.3	Machine learning: dal programmare all'addestrare	6
1.4	Deep learning: la rivincita delle reti neurali	8
1.5	La rivoluzione Transformer	10
1.6	Il panorama attuale	11
1.7	Un web che cambia lettore	13
1.8	I conti dell'AI e la domanda sulla bolla	14
1.9	Cosa deve saperne un ingegnere	16
2	Fondamenti matematici	18
2.1	Vettori e matrici	18
2.1.1	Il vettore come lista di numeri	19
2.1.2	Il prodotto scalare come misura di somiglianza	19
2.1.3	Matrici e prodotto matriciale	21
2.2	Spazi ad alta dimensionalità	22
2.2.1	Distanza euclidea	23
2.2.2	Similarità coseno	23
2.2.3	Quando l'intuizione tridimensionale inganna	24
2.3	Funzioni, derivate e gradiente	24
2.3.1	La derivata come sensibilità	25
2.3.2	Derivate parziali e gradiente	25
2.3.3	La discesa del gradiente	26
2.4	Probabilità essenziale	28
2.4.1	Distribuzioni discrete e valore atteso	28
2.4.2	Probabilità condizionata	29
2.4.3	La softmax: da punteggi a probabilità	29
2.5	Entropia e cross-entropy	31
2.5.1	L'entropia come misura dell'incertezza	31
2.5.2	La cross-entropy come funzione di perdita	32
2.6	Gli strumenti: NumPy e PyTorch	34
2.6.1	Tensori, array e operazioni vettorializzate	34
2.6.2	Broadcasting	35
2.6.3	La discesa del gradiente in codice	35
2.6.4	PyTorch: tensori con il gradiente incorporato	36

3	Reti neurali	38
3.1	Il neurone artificiale	38
3.1.1	Le funzioni di attivazione	39
3.2	Dal percettrone al deep learning	41
3.3	Il forward pass, a mano	43
3.4	Funzione di perdita e backpropagation	45
3.4.1	Lo strato di uscita	46
3.4.2	Lo strato nascosto	48
3.4.3	L'aggiornamento dei pesi	49
3.5	Il ciclo di addestramento in PyTorch	50
3.6	Overfitting, regolarizzazione e iperparametri	53
3.7	Verso il linguaggio	55
4	Rappresentare il linguaggio: token ed embedding	58
4.1	Il problema: trasformare testo in numeri	58
4.2	Tokenizzazione	60
4.2.1	Parole, caratteri o vie di mezzo	61
4.2.2	Byte-Pair Encoding	61
4.2.3	I tokenizzatori dei modelli moderni	64
4.2.4	Casi patologici	65
4.3	Token e costi	66
4.4	Embedding: il significato come geometria	67
4.5	Word2vec e l'aritmetica del significato	69
4.6	Similarità coseno in pratica	71
4.7	Embedding contestuali	73
4.8	Codice: esplorare uno spazio di embedding	74
5	Il meccanismo di attention	79
5.1	Il problema del contesto	79
5.2	L'intuizione: query, chiave, valore	81
5.3	Scaled dot-product attention	83
5.4	Esempio numerico completo	85
5.4.1	Le proiezioni: query, chiavi e valori	85
5.4.2	I punteggi	86
5.4.3	Scaling e softmax	86
5.4.4	L'output	88
5.5	Multi-head attention	88
5.6	Attention causale	90

5.7	Implementazione da zero in PyTorch	92
5.8	Il costo quadratico	95
6	L'architettura Transformer	97
6.1	Vista d'insieme	97
6.2	Positional encoding	100
6.3	Il blocco Transformer	103
6.4	Impilare i blocchi	106
6.5	La testa di output	107
6.6	Generazione autoregressiva	108
6.7	Strategie di decodifica	110
6.8	Scrivere un mini-GPT completo	114
6.9	Encoder, decoder e ibridi	120
7	Dal Transformer all'LLM	122
7.1	Pretraining	122
7.1.1	I dati: raccolta e pulizia	123
7.1.2	Il costo del pretraining	124
7.1.3	Che cosa sa un modello base	124
7.2	Scaling laws	125
7.2.1	L'equilibrio tra parametri e dati	126
7.2.2	Segnali di saturazione	127
7.3	Da completatore ad assistente: instruction tuning	127
7.3.1	Il fine-tuning supervisionato	128
7.3.2	I formati di chat	128
7.4	Allineamento alle preferenze umane	129
7.4.1	Il reward model e l'RLHF	130
7.4.2	DPO: l'alternativa diretta	131
7.4.3	Che cosa implica l'allineamento	131
7.5	Fine-tuning, LoRA e quantizzazione	132
7.5.1	Quando conviene, e quando no	132
7.5.2	LoRA: adattatori a basso rango	132
7.5.3	Quantizzazione: precisione contro memoria	133
7.6	Modelli aperti e modelli chiusi	135
7.6.1	Prova pratica: un modello sul proprio computer	136
7.7	Capacità emergenti, allucinazioni e limiti	137
7.7.1	Perché il modello inventa	138
7.7.2	I limiti, e che cosa se ne fa un ingegnere	138

7.8	Modelli di reasoning	139
II	Costruire con gli LLM	141
8	Usare gli LLM via API	142
8.1	Anatomia di una chiamata	143
8.1.1	Il system prompt come contratto	143
8.1.2	La conversazione la mantieni tu	144
8.2	Il context window come risorsa	146
8.3	Parametri di generazione	148
8.4	Structured output	150
8.5	Function calling	153
8.6	Streaming e robustezza	157
8.6.1	Guasti, retry e backoff	158
8.7	Token economics	160
8.8	Multimodalità	162
9	Prompt e context engineering	165
9.1	Il prompt come interfaccia di programmazione	166
9.2	Tecniche fondamentali	167
9.2.1	Zero-shot: la sola istruzione	168
9.2.2	Few-shot: le convenzioni si mostrano	169
9.2.3	Chain-of-thought: il procedimento prima del risultato	171
9.2.4	Quale tecnica quando	172
9.3	Context engineering	173
9.4	Prompt caching	175
9.4.1	Un conto svolto	175
9.5	Prompt injection	177
9.5.1	Perché non esiste una difesa perfetta	179
9.5.2	Mitigazioni pratiche	180
9.6	Prompt come codice	181
9.7	Pattern e anti-pattern	182
10	Ricerca semantica e vector database	185
10.1	Oltre le parole chiave	185
10.2	Modelli di embedding per la ricerca	187
10.3	Similarità e dimensionalità	188
10.4	Indici approssimati	190

10.5	Vector database	192
10.6	Chunking	194
10.7	Ricerca ibrida e reranking	196
10.8	Codice: un motore di ricerca semantica	199
11	Retrieval-Augmented Generation	204
11.1	Perché RAG	204
11.2	L'architettura	207
11.3	La pipeline di ingestion	208
11.4	Retrieval di qualità	210
11.5	Il prompt di generazione	212
11.6	Valutare un RAG	214
11.7	Errori tipici e diagnosi	216
11.8	RAG avanzato	218
11.9	Codice: un RAG completo	219
III	Agenti e sistemi multi-agente	226
12	Agenti	227
12.1	Che cos'è un agente	227
12.2	Il loop agentico	229
12.3	Tool design	232
12.4	Le skill: conoscenza procedurale su richiesta	234
12.5	Memoria	237
12.6	Pianificazione	238
12.7	Guardrail	239
12.8	Quando NON serve un agente	241
12.9	Codice: un agente da zero	242
13	Il Model Context Protocol	251
13.1	Il problema dell'integrazione	251
13.2	Architettura MCP	253
13.2.1	Il ciclo di vita di una sessione	254
13.3	Le primitive	256
13.3.1	Tools	256
13.3.2	Resources	257
13.3.3	Prompts	258
13.3.4	Quando esporre cosa	259

13.3.5	Le primitive lato client	259
13.4	I canali di comunicazione	260
13.5	Costruire un server MCP	261
13.5.1	Lo strato MCP	263
13.5.2	Collaudo, senza modello	266
13.6	Sicurezza	268
13.6.1	Prompt injection via tool description	268
13.6.2	Il confused deputy	269
13.6.3	Autorizzazioni e sandboxing	270
13.7	L'ecosistema	271
14	Sistemi multi-agente	273
14.1	Quando un agente solo non basta	273
14.2	Pattern di orchestrazione	275
14.3	Comunicazione tra agenti	278
14.4	LangGraph	280
14.5	Human-in-the-loop	283
14.6	Debugging di sistemi non deterministici	285
14.7	Caso reale: un sistema enterprise in produzione	286
14.8	Codice: supervisor e specialisti	290
IV	In produzione	294
15	LLM in produzione	295
15.1	Architetture applicative	296
15.2	Latenza	298
15.2.1	Esempio svolto: il budget di latenza	300
15.3	Costi	301
15.3.1	Esempio svolto: il mix 80/20	302
15.3.2	Batch, budget e allarmi	303
15.4	Affidabilità	304
15.4.1	Conto svolto: quanto vale un secondo provider	306
15.5	Sicurezza	306
15.6	Deployment su cloud	309
15.7	Compliance	311
16	Observability e valutazione	315
16.1	Il problema	315

16.2 Tracing	317
16.3 Metriche di valutazione	320
16.4 Costruire eval set	323
16.5 Regression testing	325
16.6 Monitoraggio in produzione	327
16.7 Codice: una pipeline di eval	328
 17 Caso di studio: un assistente aziendale completo	 335
17.1 Requisiti	335
17.2 Architettura	337
17.3 La base di conoscenza	340
17.4 Gli agenti e gli strumenti	341
17.5 Valutazione e hardening	346
17.6 Deploy e monitoraggio	349
17.7 Retrospettiva	351
 Bibliografia e riferimenti	 354