perplexity

Manuale di Vibe Coding

Il **vibe coding** è l'arte di creare software lasciando che a scrivere (quasi) tutto il codice sia un modello linguistico di ultima generazione, mentre lo sviluppatore dirige l'orchestra con prompt chiari, test rapidi e feedback continui. Questa guida pratica in italiano spiega filosofia, strumenti, flusso di lavoro, sicurezza e buone pratiche per trasformare l'intuizione in applicazioni funzionanti, senza perdere il controllo.

Introduzione: cos'è il vibe coding?

Vibe coding significa "programmare a sentimento": descrivi ciò che vuoi, lasci che l'AI generi codice, accetti in blocco le modifiche e prosegui finché gira. Il termine è stato coniato da Andrej Karpathy nel 2025 e riassume un approccio improvvisato ma sorprendentemente produttivo[1].

Andrej Karpathy explains vibe coding as a relaxed, intuitive approach to coding using advanced AI tools, focusing on vibes and minimal direct coding.

Perché sta esplodendo

- LLM sempre più capaci di ragionare sul contesto del progetto[2].
- IDE specializzati (Cursor, Replit, Windsurf) che integrano chat e terminale.
- Accettazione aziendale: 25% delle startup Y Combinator 2025 ha >90% di codice AI-generato[3].
- Barriera d'ingresso bassa: non servono anni di studio per prototipare.

Sezione 1 – Setup dell'ambiente

1.1 Scegliere la toolchain

Caso d'uso	IDE/Platform	Modello consigliato	Note
Web app full-stack	Cursor	Claude 3.5 Sonnet	Avvia progetti Vite, NextJS, Django
Scripting veloce	Replit	Replit Code V2	Sandbox cloud integrato
Refactoring enterprise	GitHub Copilot	Copilot Enterprise	Accesso al codice privato

1.2 Attivare voice-prompt opzionale

Installare SuperWhisper o Whisper Flow per dettare prompt lunghi senza interrompere il flow.

Sezione 2 – Il ciclo "Think - Prompt - Test"

1. **Think**
   - Definisci l'obiettivo in 1-2 frasi ("Dashboard Flask con login JWT").
   - Spezza il progetto in milestone microscopiche.
2. **Prompt**
   - Fornisci contesto (stack, librerie, standard di stile).
   - Usa istruzioni negative: "non modificare backend", "solo Tailwind".
3. **Test**
   - Esegui subito. Se fallisce, incolla l'errore nel prompt senza commenti; l'AI proporrà fix.
   - Salva checkpoint Git dopo ogni passaggio riuscito.

Ripeti fino al raggiungimento del MVP.

Sezione 3 – Prompt engineering essenziale

- **Specificità > lunghezza**: "Crea endpoint FastAPI /stats che restituisca JSON {uptime, version}" è meglio di "fammi un'API".
- **Few-shot**: mostra un esempio di funzione commentata, chiedi "scrivine una simile per…".
- **Guardrail prompt** iniziale:

```
Da ora modifica solo i file indicati, mantieni PEP8, usa env vars per segreti.
Chiedi conferma se la richiesta implica breaking changes.
```

Sezione 4 – Debugging nel vibe coding

- **Rule of three**: se dopo tre iterazioni l'errore persiste, leggi il diff e intervieni manualmente.
- **Unit test AI-assistiti**:

```
Scrivi pytest per la funzione X coprendo input vuoto, input malformato,
e assicurati che sollevi ValueError.
```

- **Sandbox sicuro**: esegui codice AI in container Docker con CPU e RAM limitate per evitare loop infiniti[4].

Sezione 5 – Sicurezza e governance

Rischio	Mitigazione "vibey"
Hard-coding di credenziali	Prompt esplicito: "usa os.getenv e .env.example"
Dipendenze vulnerabili	Esegui `pip-audit` automatico a ogni build
Hallucinated packages	Blocca installazioni se non presenti su PyPI stabile
Licenze ambigue	Chiedi "commenta la licenza di ogni snippet esterno"

Sezione 6 – Scalabilità: dal prototipo al production

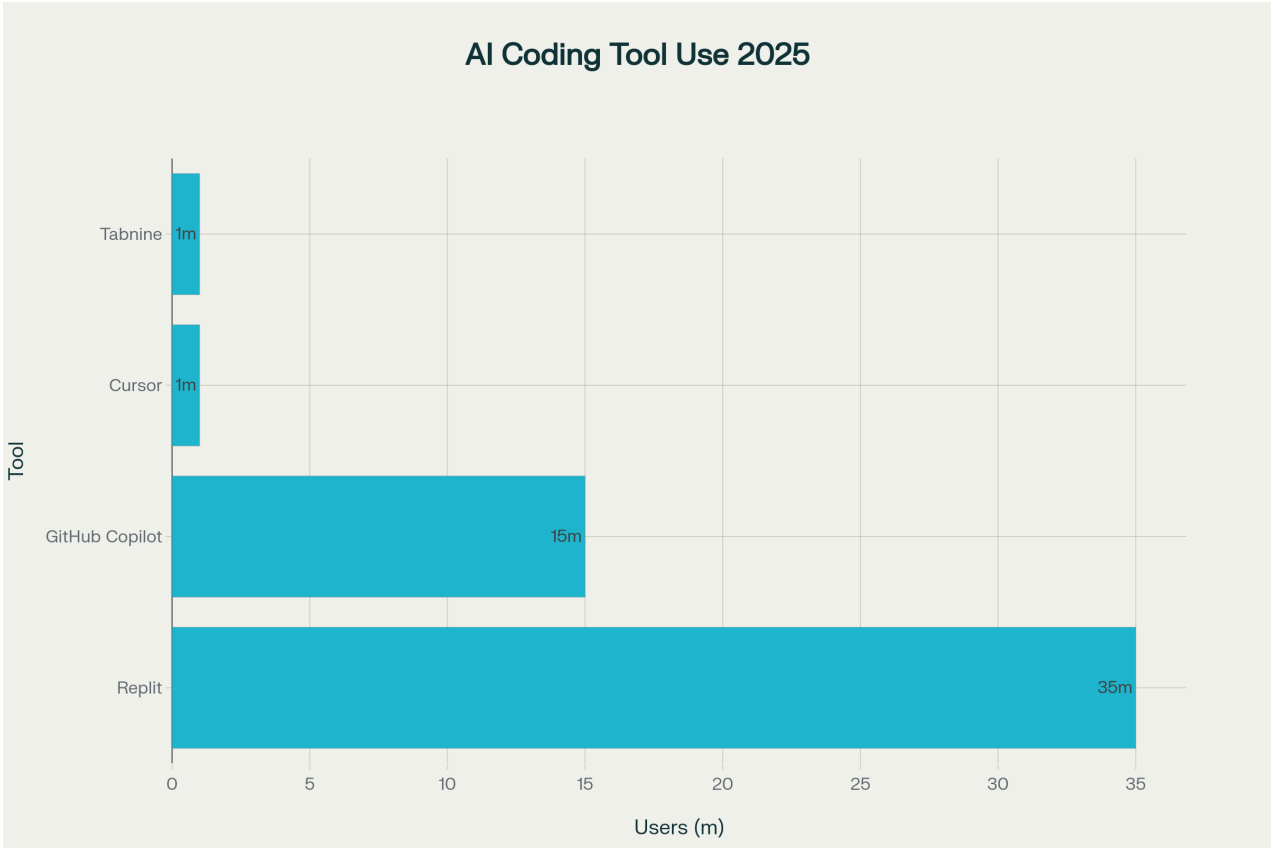
1. **Freeze requirements** (`pip freeze` / `npm pkg`).
2. Review manuale di un dev senior: l'obiettivo è capire le parti critiche, non leggere tutto.
3. Aggiungi observability (logging strutturato + Healthcheck endpoint).
4. Pipeline CI con test e controlli SCA prima del deploy.

Sezione 7 – Buone pratiche avanzate

- **YOLO mode off** in produzione: approva manualmente i comandi shell suggeriti.
- **Prompt patterns riutilizzabili**: costruisci una libreria "snippet YAML" da incollare all'AI per routing, auth, database.
- **Flow state design**: luce ambientale, playlist Lo-fi, finestra IDE a schermo intero per ridurre task-switching[5].

Sezione 8 – Metriche e adozione

Il 2025 vede oltre 50 M di sviluppatori che usano almeno un assistente AI; la ripartizione degli strumenti principali è indicativa:



Adoption of major AI coding tools linked to vibe coding, 2025

Replit domina fra i creator, Copilot resta lo standard enterprise, mentre Cursor è la scelta di riferimento per puro vibe coding.

Conclusione

Il vibe coding non sostituisce l'ingegneria tradizionale: la completa, velocizzando ideazione e prototipazione. Segui il ciclo **Think-Prompt-Test**, applica prompt rigorosi, mantieni controllo sul rischio e potrai trasformare ispirazioni estemporanee in software reale con una velocità senza precedenti.

> **Ricorda**: l'AI è il tuo junior developer instancabile; tu rimani l'architetto che sa quando fidarsi e quando mettere le mani nel codice.

Buona vibrazione e buon codice!

⁂

Manuale di Vibe Coding — Edizione Estesa

Prima di addentrarci nei dettagli, ecco la sintesi: **il vibe coding va oltre il semplice "codificare a sentimento" con un LLM**. È un paradigma completo di sviluppo assistito dall'AI che coinvolge strategia di prodotto, sicurezza, governance, metriche di valore e gestione del ciclo di vita del software. Questa versione estesa integra il white-paper allegato con ricerche 2024-2025, dati di adozione industriale, rischi emergenti e pattern organizzativi per permettere a team di qualsiasi dimensione di sfruttare l'AI in modo scalabile e responsabile.

1. Evoluzione del concetto

1.1 Dal prompt al sistema socio-tecnico

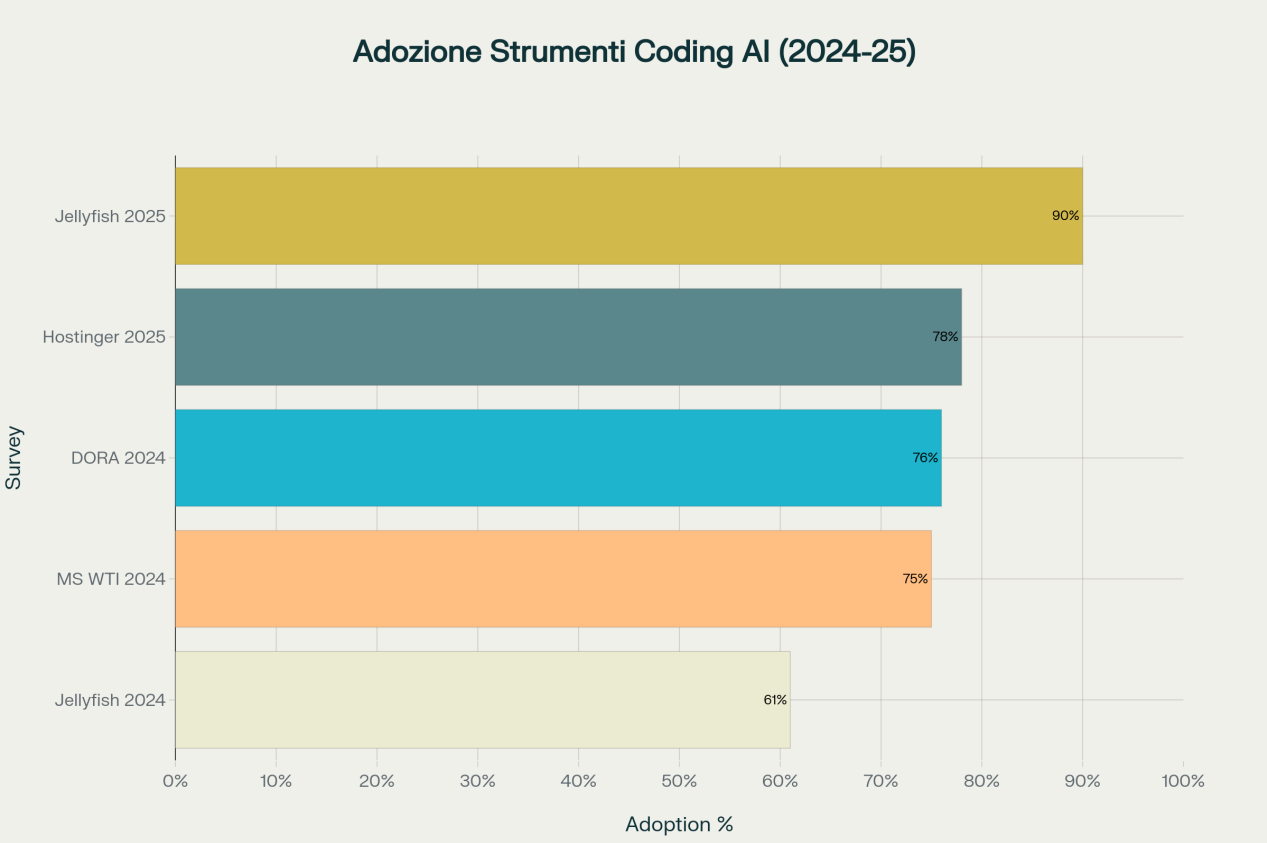
Il white-paper descrive il vibe coding come “coding through conversation”. Oggi il focus si sposta sul **sistema socio-tecnico**:

- agente AI (LLM o multi-agent)
- toolchain (IDE, CI/CD, code-intel)
- politica di sicurezza e dati
- competenze umane (prompt, revisione, product thinking)

Questi elementi formano un ciclo **Intent** → **Generazione** → **Verifica** → **Refine** che sostituisce l’approccio lineare tradizionale[6] [7].

1.2 Adozione reale

Cinque studi indipendenti mostrano ormai tassi di utilizzo compresi fra 61% e 90% tra developer e aziende[8] [9] [10] [11] [12].



Adozione globale di strumenti di coding AI secondo cinque indagini indipendenti (2024–2025)

2. Architettura di riferimento

2.1 Strati della piattaforma

Strato	Funzione	Tool tipici
Interface	Prompt (testo/voce) + parametri	Cursor, Replit, GitHub Copilot
Agentic Layer	Orchestrazione multi-step, planner, tool-calling	OpenAI Functions, LangGraph
Execution Sandbox	Run, test, container isolato	WebContainers, Docker-in-browser
Observability & Guardrail	Tracing, limitazione, policy	OpenTelemetry, prompt firewall
Knowledge & Context	Retrieval codebase, doc, log	RAG, vector DB
Governance	Audit, licenze, data privacy	SCA, SBOM, IAM

2.2 Pattern agentici

- **Code Review Agent**: analizza PR, suggerisce fix, firma se i test passano.
- **Migration Agent**: rifattorizza librerie deprecate, produce diff e test.
- **Infra Agent**: modifica IaC e verifica costi cloud simulati.

3. Prompt engineering avanzato

1. **Ruolo esplicito**: “Agisci come Senior SRE, segui CIS Benchmark”.
2. **Struttura** I-C-E (Istruzioni, Contesto, Esempi).
3. **Small-batch prompting**: suddividere feature complesse in task da ≤300 token[13] [14].
4. **Self-critique**: chiedere al modello di generare prima test/unit e checklist di rischio.

4. Quality & Security

4.1 Vulnerabilità tipiche

- **Package hallucination → slopsquatting**: il 20% dei suggerimenti include librerie inesistenti[15] [16].
- **Codice insicuro**: in studi controllati 10-42% degli snippet AI contiene bug di security[17].

4.2 Contromisure

1. Policy “verify-before-merge”.
2. Scan SCA + AI-linters addestrati su CVE.
3. Sandbox con timeout e limiti di risorse nell’esecuzione di codice generato.
4. Uso di **SBOM** aggiornato a ogni build per tracciare dipendenze AI-injected.

5. Metriche di valore

KPI	Descrizione	Bench recenti
Throughput	PR merged/dev·settimana	+3% con AI [18]
Lead Time	Idea → Prod	–15-30% nei prototipi [19] [20]
Delivery Stability	MTTR & failure rate	–7% se i batch diventano troppo grandi [21]
Developer Flow	h “deep work”/sett.	+10 h con copilots [22]
Security Defects	bug per KLOC	variabile (↑ se review debole) [23] [24]

6. Organizzazione e ruoli

6.1 AI Champion

Responsabile di piattaforma, linee guida e formazione continua.

6.2 Prompt Engineer

Collabora con tech-lead per curare libreria di template, test automatici e valutazioni.

6.3 AI Governance Board

Include legale, security e HR; definisce policy su licenze, privacy, AI ethics.

6.4 Change Management

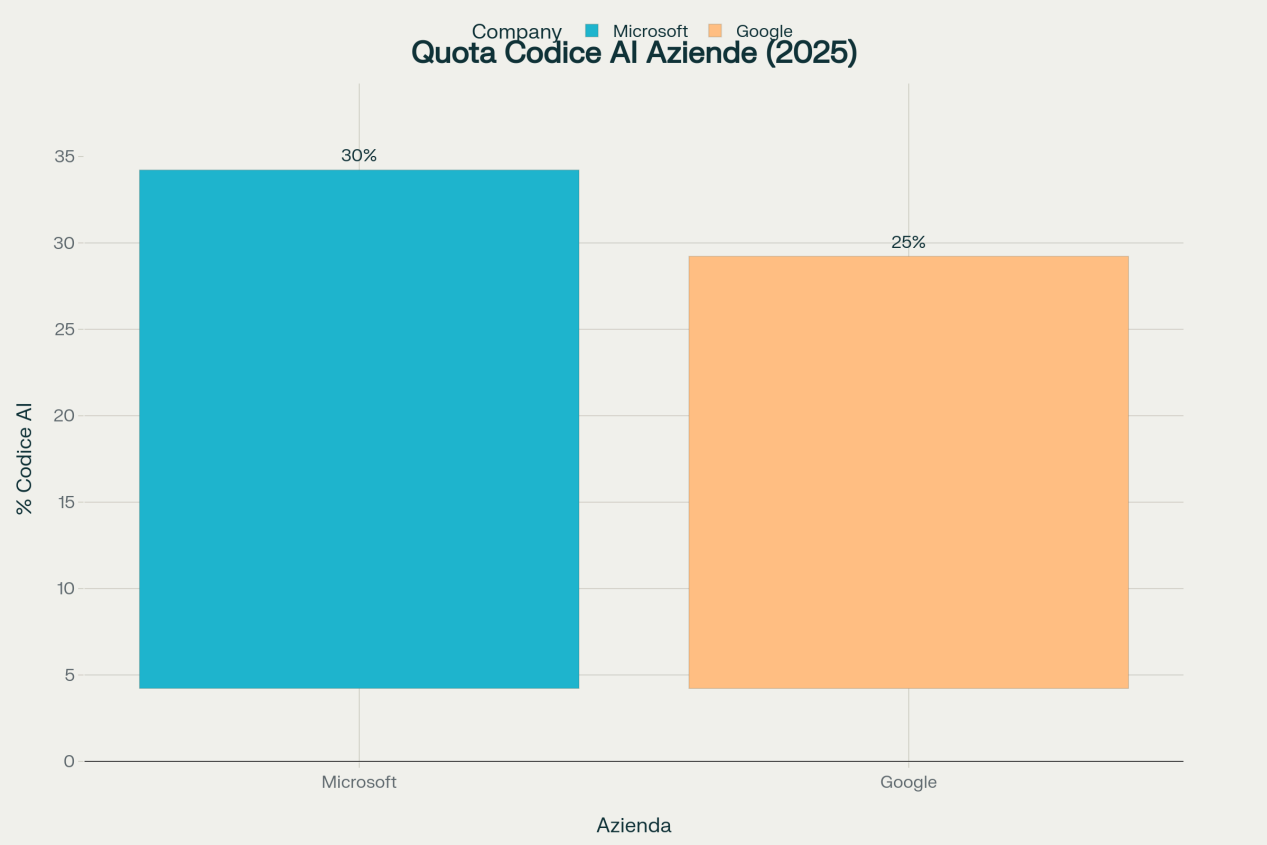
Train-the-trainer + pairing “dev senior ↔ junior + Copilot” per trasferire best practice.

7. Roadmap di adozione

1. **Assessment**: mappa casi d’uso e dati sensibili.
2. **Pilota sandbox**: un team, un dominio ristretto, metriche baseline.
3. **Scale-out**: integra observability e policy; estendi a CI/CD.
4. **Optimization**: introduce agenti specializzati, tuning su dati proprietari.

8. Tendenze 2025-2027

- **Agentic AI** diventa default nei dev-tool [25] [26].
- **Quota di codice AI** in big tech già al 25-30% e in crescita [27] [28].



Percentuale di codice interno generato da AI presso Microsoft e Google (stime 2025)

- **Platform engineering** unisce AI, DevOps e finanziamento FinOps per controllare costi cloud.
- **Regolazioni UE/USA**: AI Act, OWASP LLM Top 10 2025[29] [30] richiedono audit continuo.
- **Skill shift**: AI literacy considerata requisito di base nei nuovi assunti[9].

Conclusioni

Il vibe coding non è più un esperimento da weekend, ma **un motore d'innovazione** che, se governato correttamente, accelera time-to-value, democratizza lo sviluppo e aumenta la resilienza del prodotto. Chi adotta un approccio olistico — dalla strategia dati alla sicurezza, dalle metriche di business alla formazione — potrà sfruttare l'AI come vantaggio competitivo sostenibile.

⁂

Vibe Coding per Tutti

Semplificare la programmazione con l'AI, senza gergo tecnico

Vibe coding è un modo nuovo di "scrivere" software: descrivi cosa ti serve in italiano corrente, un assistente AI genera il codice e tu lo provi, correggi, ripeti. Questa guida spiega, in parole semplici, come funziona, quali vantaggi offre a chi non è sviluppatore e quali precauzioni servono per usare l'AI senza farsi male.

1. Che cos’è il vibe coding

1.1 L’idea di base

1. Tu esprimi l’obiettivo: «Vorrei un sito dove la gente può prenotare i miei corsi di yoga».
2. L’AI propone il codice, crea schermate, imposta database.
3. Tu provi l’app, dai feedback («cambia colore», «aggiungi PayPal») e l’AI aggiorna.

È come dettare la ricetta a un robot-cuoco: tu decidi il piatto, lui prepara gli ingredienti.

1.2 Perché interessa ai non addetti ai lavori

- Nessun linguaggio di programmazione da studiare in anticipo.
- Prototipi in ore invece che in settimane.
- Possibilità di sperimentare e imparare “facendo”, con errori piccoli e reversibili.

2. Strumenti che fanno (quasi) tutto loro

Situazione tipica	Strumento consigliato	Cosa fa in automatico
Landing page semplice	**v0 (Vercel)**	Genera design + codice React
App completa in browser	**Replit Agent**	Codice, test, hosting one-click
Modifiche a un sito già esistente	**Cursor**	Capisce il tuo progetto, propone patch

> Suggerimento rapido: prova la versione gratuita; se l’editor ti sembra “magico”, resta, altrimenti passa al successivo.

3. Il ciclo “Prova & Sbaglia”

1. **Scrivi il prompt** – chiaro, breve, specifico («crea un form con nome, email, telefono»).
2. **Avvia l’AI** – guarda subito il risultato: gira? sì/no.
3. **Se no** – copia l’errore e chiedi «correggi il bug»; l’AI propone la patch.
4. **Itera** – ogni correzione è un passo avanti; niente vergogna se sbagli.

Questa logica di tentativi ripetuti è normale: anche i professionisti vivono di debug quotidiano.

4. Vantaggi concreti

- **Velocità**: MVP (primo prototipo funzionante) in un weekend.
- **Costo ridotto**: salti la fase di assunzione di un dev per prove iniziali.
- **Creatività**: puoi osare idee strane senza investire migliaia di euro.

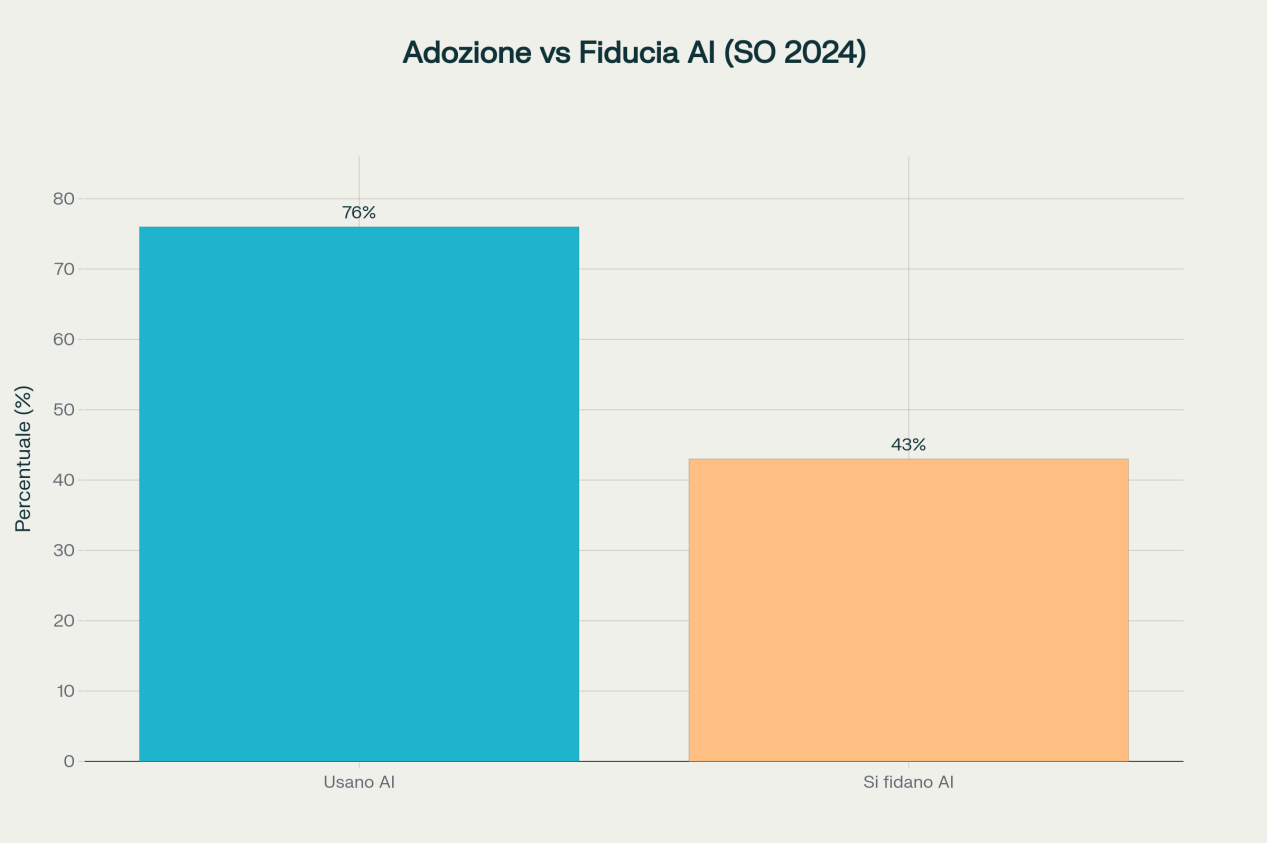
5. Limiti e rischi da conoscere

5.1 Codice non sempre sicuro

Studi recenti mostrano che quasi metà dei frammenti AI contiene vulnerabilità [31] [32].

5.2 Fiducia limitata

Solo 43% degli sviluppatori si fida davvero dell'output dell'AI, pur usandola già il 76% [33].



Differenza tra adozione e fiducia: il 76% usa l'AI, ma solo il 43% si fida.

5.3 Dipendenza dal contesto

Se il prompt è vago, l'AI "inventa" librerie inesistenti o procedure sbagliate [34].

6. Mini-checklist di sicurezza "senza stress"

1. **Segreti fuori dal codice** – password e API-key in file .env, non dentro il prompt.
2. **Controllo umano finale** – prima di pubblicare, chiedi a un amico sviluppatore di dare un'occhiata.
3. **Backup continuo** – usa Git o simili; se l'AI rovina qualcosa, torni indietro con un click.
4. **Aggiornamenti** – ogni mese verifica che librerie e dipendenze siano ancora supportate.

7. Imparare giocando: esercizi veloci

1. **Genera un contatore di visite** – prompt: «pulsante che conta quante volte viene cliccato».
2. **Aggiungi un tema scuro** – prompt: «inserisci toggle light/dark».
3. **Integra un foglio Google** – prompt: «salva i dati del form su Google Sheets».

Ogni esercizio dura 15 minuti: vedrai errori, li correggerai, e capirai come dialogare meglio con l'AI.

Metafora visiva: programmare insieme all'AI senza conoscere il codice.

8. Domande frequenti

Devo studiare Python o JavaScript?
Non subito. Parti dai prompt. Col tempo ti verrà voglia di capire cosa scrive l'AI e allora sì, un tutorial base sarà utile.

Posso fare un e-commerce serio solo con vibe coding?
Il prototipo sì. Per andare in produzione serve comunque un controllo professionale su pagamenti, sicurezza e scalabilità.

Quanto costa?
Molti strumenti offrono piani free con limiti giornalieri di richieste AI. Per uso intenso (startup) calcola 10-20 €/mese.

Conclusione

Il vibe coding è la bicicletta elettrica della programmazione: abbassa la fatica, ma serve ancora pedalare un minimo. Per chi non è addetto ai lavori è un'occasione storica: puoi trasformare idee digitali in realtà con pochi tentativi e un po' di curiosità. Parti in piccolo, sbaglia spesso, correggi in fretta — e divertiti a far scrivere il codice a un robot che capisce l'italiano.

⁂

1. https://en.wikipedia.org/wiki/Vibe_coding
2. https://cloud.google.com/discover/what-is-vibe-coding
3. https://www.innobu.com/vibecoding-dashboard/
4. https://huggingface.co/docs/smolagents/tutorials/secure_code_execution
5. https://peerlist.io/blog/engineering/how-to-vibe-code
6. https://en.wikipedia.org/wiki/Vibe_coding
7. https://www.ibm.com/think/topics/vibe-coding
8. https://www.linkedin.com/pulse/dora-2024-usage-artificial-intelligence-jay-alphey-6dkue
9. https://www.microsoft.com/en-us/worklab/work-trend-index/ai-at-work-is-here-now-comes-the-hard-part
10. https://jellyfish.co/resources/2024-state-of-engineering-management-report/
11. https://www.hostinger.com/tutorials/how-many-companies-use-ai
12. https://www.simform.com/blog/the-state-of-generative-ai/
13. https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api
14. https://mirascope.com/blog/prompt-engineering-best-practices
15. https://en.wikipedia.org/wiki/Slopsquatting
16. https://www.securityweek.com/ai-hallucinations-create-a-new-software-supply-chain-threat/
17. https://arxiv.org/html/2506.23034v1
18. https://devops.com/latest-dora-report-surfaces-limited-gains-from-ai-and-platform-engineering/
19. https://www.datacamp.com/blog/vibe-coding

20. https://shiftmag.dev/impact-ai-software-developers-5111/
21. https://services.google.com/fh/files/misc/dora-impact-of-generative-ai-in-software-development.pdf
22. https://www.atlassian.com/blog/developer/developer-experience-report-2025
23. https://cset.georgetown.edu/publication/cybersecurity-risks-of-ai-generated-code/
24. https://www.techtarget.com/searchsecurity/tip/Security-risks-of-AI-generated-code-and-how-to-manage-them
25. https://en.wikipedia.org/wiki/Agentic_AI
26. https://about.gitlab.com/the-source/ai/agentic-ai-unlocking-developer-potential-at-scale/
27. https://www.cnbc.com/2025/04/29/satya-nadella-says-as-much-as-30percent-of-microsoft-code-is-written-by-ai.html
28. https://opendatascience.com/satya-nadella-says-ai-now-writes-30-of-microsofts-code/
29. https://www.hackerone.com/blog/owasp-top-10-llms-2025-how-genai-risks-are-evolving
30. https://www.europarl.europa.eu/doceo/document/TA-9-2023-0236_EN.html
31. https://cset.georgetown.edu/publication/cybersecurity-risks-of-ai-generated-code/
32. https://www.techtarget.com/searchsecurity/tip/Security-risks-of-AI-generated-code-and-how-to-manage-them
33. https://www.logisticsit.com/articles/2024/07/26/stack-overflow's-2024-developer-survey-shows-the-gap-between-ai-use-and-trust-in-its-output-continues-to-widen-among-coders
34. https://devops.com/ai-generated-code-packages-can-lead-to-slopsquatting-threat/