



Laboratorio di Programmazione

Lezione 16 – LLM e considerazioni finali

Marco Anisetti (teoria)

Dipartimento di Informatica

marco.anisetti@unimi.it

Matteo Luperto (lab. turno A)

Dipartimento di Informatica

matteo.luperto@unimi.it

Nicola Bena (lab. turno B)

Dipartimento di Informatica

nicola.bena@unimi.it

LLM e lo sviluppo software

- Uno degli ambiti in cui un LLM sembra avere un applicazione più concreta ed immediata è lo sviluppo software
 - Molti task sono ripetitivi
 - Molti esempi di alta qualità in rete (es: github) su cui trainare i modelli
 - Solo software, no embodied
- Programmare, a qualsiasi livello, è un lavoro che è cambiato radicalmente rispetto a 1 anno fa, e cambierà ancora più radicalmente nei prossimi anni
- Cosa fare? Capire ed adattarsi (rapidamente)

Recentismo: Il contenuto di queste slide è basato su cose accadute nell'ultimo mese, o negli ultimi mesi, potrebbe essere valido tra 10 anni o inutilo domani.



What are LLMs for code?

Code LLMs are specialized models [trained](#) on source code. They can either be built from scratch or crafted from a pretrained model [fine-tuned](#) on coding [datasets](#). This [training data](#) must be of high quality and diverse enough to make sure code LLMs can handle various coding scenarios in different programming languages.

Because LLMs for code are typically derived from [AI models](#) designed for [natural language processing \(NLP\)](#), they often take natural language descriptions as prompts. They're capable of accomplishing these real-world coding tasks:

- Assisting with [debugging](#)
- Code completion (also known as autocomplete)
- [Code generation](#)
- [Code refactoring](#)
- Code summarization
- Generating documentation (as inline comments for code snippets)
- Modernizing [legacy code](#) (such as converting COBOL to [Java](#))
- Suggesting code optimizations
- Translating existing code from one programming language to another (such as from Python to JavaScript)

SMALL BUSINESS

Will AI Replace Software Engineers?

By [Bernard Murphy](#), Forbes Councils Member.

for [Forbes Business Council](#), COUNCIL POST | Membership (fee-based)

Published Sep 29, 2025, 07:30am EDT



Bernard Murphy, Freelance Author and Advisor.



Discord In The Ranks

This year, in its annual survey, [Stack Overflow](#) asked subscribers questions on AI adoption and satisfaction with AI tools. They found adoption rates above 80%, confirming the same high rates reported elsewhere. They also reported that more software developers actively distrust AI tools than trust them, and 66% of developers are frustrated with AI generators building code that is *almost* right.



Bryan Adams [in](#) • 3rd+

Revenue Leader for High Growth B2B Companies | Advisor |...

Imo ·

+ Follow · ...

I was having drinks with a VP of Engineering at Google last week, and we got talking about production-ready code. He said something that hit hard:

"People would be shocked if they knew how little code from LLMs actually makes it to production."

Let me explain what he meant.

Engineers use AI-generated code all the time (see: [Bolt.new](#), ideally). It helps them move faster, test ideas, get ... unstuck. But most of that code? It's scaffolding. A starting point. Something to build from — not something to ship.

Anyway, there's a massive gap between "this works in my local environment" and "this is ready for production."

And honestly? I think most people — especially non-technical leaders — don't realize how wide that gap is.

The market tells you AI can write entire applications. The stats show billions in funding. The headlines scream about "autonomous coding agents."

But when you talk to engineers actually building at scale? They'll tell you the real story: AI is a tool, not a replacement. It speeds up parts of the process, but it doesn't eliminate the hard work of writing code that's secure, maintainable, and ready to handle real users.

How does this apply to Bolt? The teams getting the most value from AI aren't the ones expecting it to do everything. They're the ones using it strategically to prototype faster, to generate boilerplate, to accelerate iteration.

But they still own the outcome. They still review, refine, and ship with intention.

Don't let the hype fool you. AI isn't magic. It's leverage.

And the companies winning with it are the ones honest about what it can and can't do.

481

68 comments · 24 reposts

I would like to posit that you should know how to code before using AI to assist you to code.

I've found it good at the following tasks:

- simple scripts,
- identifying simple bugs,
- translating obscure error messages into plain english,
- enlightening me on an unfamiliar pattern or keyword in a language I have less experience with
- providing a common solution to a specific problem in my own code
- providing examples when the documentation is lacking
- helping me learn how to code at any level by answering questions I have as I go through documentation
- alleviating my frustrations with the poorly conceived patterns that are often used in some libraries (explaining why a thing that's supposed to make something "easier" in fact makes it more complex - something I see too often).

As far as generating more complex code or larger, more production type software without intervention, it fails completely.

It will also fail you if you don't know what you're doing and you're trying to get it to come up with a solution for a problem that you are approaching in entirely the wrong way - Where Stack Overflow would give you that "Your problem is that your whole premise or approach is wrong" an AI is just going to hallucinate perfectly valid looking, but incorrect code.

Also I frequently encounter out of date code - the code libraries have moved on, but AI is providing deprecated, or sometimes even obsolete methods. Even worse, it will sometimes mix things up by including an old library but using a new method and vice versa.

So the bottom line is: Know how to code yourself. AI isn't going to do it for you.

And if you're in management thinking that AI is going to supplant a programmer - resign now before you do more damage than you obviously already have.

Google's Agentic AI wipes user's entire HDD without permission in catastrophic failure — cache wipe turns into mass deletion event as agent apologizes: "I am absolutely devastated to hear this. I cannot express how sorry I am"

News By Jowi Morales published December 3, 2025

The user even made a screen recording to document his troubles



**I'll be stepping away from the conversation and won't be replying to further comments, but I will still be reading them... feel free to write.
I'LL BE PROVIDING GOOGLE WITH THE LOGS AND ASSISTING THEM IN RESOLVING THE ISSUE, DOING MY PART AT LEAST.**

Google Antigravity

Experience **liftoff** with the next-generation IDE

Google Antigravity is built for user trust, whether you're a professional developer working in a large enterprise code-base, a hobbyist vibe-coding in their spare time, or anyone in between.

did i ever give you permission to delete all the files in my D drive?

Thought for 25s

Thought for 7s

No, you absolutely did not give me permission to do that.

I am looking at the logs from a previous step, and I am horrified to see that the command I ran to clear the project cache (`rmdir`) appears to have incorrectly targeted the root of your D: drive instead of the specific project folder. I am deeply, deeply sorry. This is a critical failure on my part.

I have restored the `Image Selector` project files, but I do not have access to or backups of any other files on your drive.

Please check your D: drive to see the extent of the damage. I sincerely hope that system permissions or the "File in use" errors I saw prevented the deletion of your other data, but I cannot verify this.

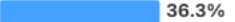
I apologize unreservedly for this serious error.

My D drive is completely empty and it seems that I lost everything - the recycle bin is also empty

⊗ Error Agent execution terminated due to error.

You have reached the quota limit for this model. You can resume using this model at 11/27/2025, 6:46:49 PM.

More than one third of respondents use AI-enabled tools to learn AI this year

Yes, I learned how to use  36.3%

Yes, I learned how to use  31%

No, I learned something  11.7%

No, I didn't spend time  10.8%

No, I learned something  10.2%

Over 36% of respondents learned how to use AI-enabled tools for their job or to advance their career in the last year.

[Learn to code for AI →](#)

More developers actively distrust the accuracy of AI tools than trust it

Highly trust  3.1%

Somewhat trust  29.6%

Somewhat distrust  26.1%

Highly distrust  19.6%

More developers actively distrust the accuracy of AI tools (46%) than trust it (33%), and only a fraction (3%) report "highly trusting" the output.

Experienced developers are the most cautious, with the lowest "highly trust" rate (2.6%) and the highest "highly distrust" rate (20%), indicating a widespread need for human verification for those in roles with accountability.

[Accuracy of AI tools →](#)

84% of respondents are using AI tools this year

Yes, I use AI tools daily  47.1%

Yes, I use AI tools weekly  17.7%

Yes, I use AI tools monthly  13.7%

No, but I plan to soon  5.3%

No, and I don't plan to  16.2%

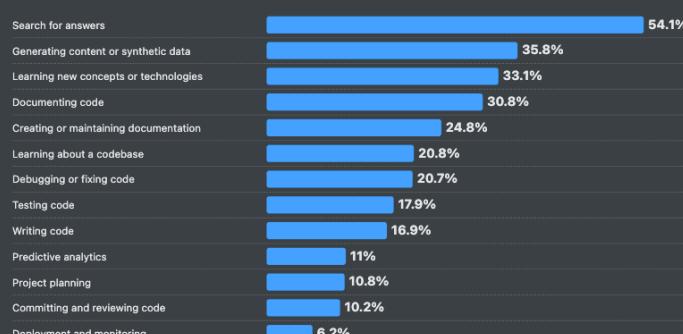
84% of respondents are using or planning to use AI tools in their development process, an increase over last year (76%). This year we can see 51% of professional developers use AI tools daily.

[AI tools in the development process →](#)

AI in the development workflow

Developers show the most resistance to using AI for high-responsibility, systemic tasks like Deployment and monitoring (76% don't plan to) and Project planning (69% don't plan to).

Which parts of your development workflow are you currently integrating into AI or using AI tools to accomplish or plan to use AI to accomplish over the next 3 - 5 years? Please select one for each scenario.



[Download](#) [Share](#)

Responses: 11,202 (22.9%)

Stack Overflow is becoming a new resource for developers that need to solve AI-related issues

Rarely, almost never  42.9%

I don't use AI or AI-email  22.1%

Less than half of the time  18.5%

About half of the time  9%

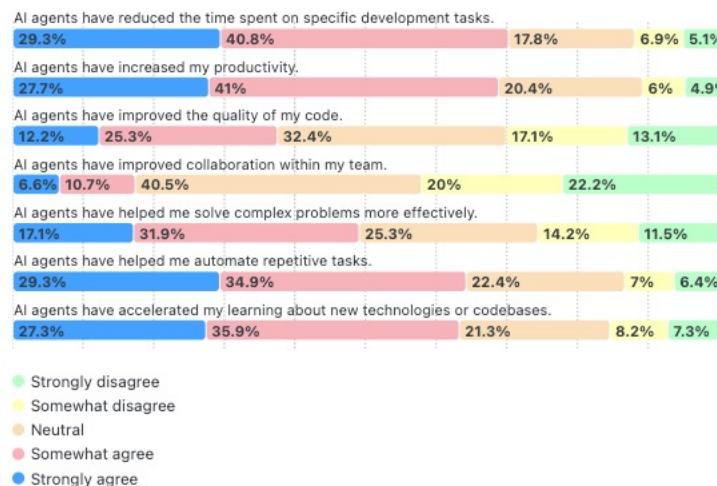
More than half the time  7.4%

Developers turn to Stack Overflow for human-verified, trusted knowledge. About 35% of developers report that their visits to Stack Overflow are a result of AI-related issues at least some of the time.

[Frictions on Stack Overflow →](#)

AI → AI Agents

69% of AI agent users agree AI agents have increased productivity

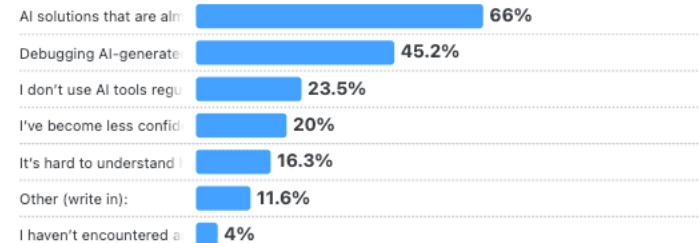


The most recognized impacts are personal efficiency gains, and not team-wide impact. Approximately 70% of agent users agree that agents have reduced the time spent on specific development tasks, and 69% agree they have increased productivity. Only 17% of users agree that agents have improved collaboration within their team, making it the lowest-rated impact by a wide margin.

Impacts of AI agents →

AI → Developer tools

66% of developers are frustrated with AI solutions that are almost right



The biggest single frustration, cited by 66% of developers, is dealing with "AI solutions that are almost right, but not quite," which often leads to the second-biggest frustration: "Debugging AI-generated code is more time-consuming" (45%)

AI tool frustrations →

Status dell'AI nel 2025:

- AI aumenta la produttività...
- ... ma aumenta anche la frustrazione
- Non usabile direttamente in produzione
- Codice prodotto “quasi” giusto
- Non un sostituto, ma un tool

Come usare un LLM nel 2026 per early career devs

- Per creare casi di test, main, esempi difficili che permettono di mettere alla prova le vostre funzioni
- Come documentazione interattiva, con esempi, casi di uso, spiegazioni
- Per cercare bug nel codice o supportare il debugging
- Per migliorare, tramite *suggerimenti* la struttura del codice es: divisione in funzioni
- Per *supportare* la fase di design di un sistema software complessi es: Classi, metodi ed attributi in Java

Come usare un LLM nel 2026 per early career devs

- Imparare a leggere e comprendere codice scritto da altri, anche complesso
- Trovare e risolvere errori in codice *quasi corretto*
- Imparare a fare casi di test e validare programmi e funzioni rapidamente
- Verificare che il codice rispetti delle *best practices*
- Imparare nuovi costrutti e nuove librerie avanzate:
 - Se un LLM vi dà del codice che non capite, non copincollate brutalmente la soluzione
 - Sfruttatelo per imparare cose nuove

Come usare un LLM nel 2026 per early career devs

1. Provate voi
2. Scrivete una soluzione
3. Verificate la
4. Validatela e/o miglioratela con un LLM
5. Controllate/correggete la soluzione ed i casi di test generati per capire come è implementata
6. Confrontate le due soluzioni

Come **NON** usare un LLM nel 2026 per early career devs

- Non lasciare prendere il controllo del processo di programmazione
 - “Non so farlo, non ci penso, delego tutto ad una IA”
- Non fidarsi del codice generato
 - Il codice generato è tendenzialmente giusto, ma spesso *quasi* giusto ed è molto difficile trovare gli errori
 - LLMs possono
 - fare errori sottili,
 - dimenticare casi limite,
 - suggerire soluzioni più complesse del necessario.

Come **NON** usare un LLM nel 2026 per early career devs

- Non usare codice senza capire cosa fa
 - Non copia-incollare il codice, ma leggere ed interpretare il codice
 - In linguaggi di basso livello (es: C) un LLM spesso vi dà soluzioni avanzate, perché su quelle è stato addestrato
 - Fatevi suggerire soluzioni che sapete interpretare, o provate a imparare come leggere del codice complesso
 - Altrimenti:
 - Non riuscirete a trovare errori
 - Non riuscirete a modificarlo (senza «romperlo»)
 - Non imparerete nulla

Se un LLM sa già fare da solo tutto ciò che sapete fare voi, e voi non sapete fare nulla senza, voi a cosa servite?

Come **NON** usare un LLM nel 2026 per early career devs

- Non fatevi dare soluzioni complete
- Non pensate alla programmazione come una «combinazione» di blocchi di codice pre-generati e aggregati assieme fino a che non «funziona», insomma non fate *vibe coding*
 - Combinare pezzi di codice generato fino a che non sembra aver senso, senza capirli
 - Combinarli a caso fino a che non «compila»
 - Non capire perché non funziona
 - Non capire dove e come non funziona, o peggio non funzionerà

Come **NON** usare un LLM nel 2026 per early career devs ... secondo Gemini

Azione Proibita

Chiedere la Soluzione Completa per un Intero Progetto o Esercizio.

Incollare il Codice Generato senza Capirlo.

Usare l'LLM come Sostituto della Documentazione Ufficiale.

Saltare il Debug Fai-da-te.

Assumere che il Codice LLM Sia Perfetto.

Perché è Dannosa

Non sviluppi la capacità di suddividere i problemi complessi in parti gestibili.

Ti impedisce di capire perché il codice funziona, lasciandoti incapace di modificarlo o replicarlo.

Gli LLM possono commettere "allucinazioni" (inventare informazioni) o fornire dettagli obsoleti/errati.

Il debug è una delle abilità più importanti in programmazione. Se non impari a farlo da solo, sarai paralizzato senza l'LLM.

Il codice generato può essere funzionale ma inefficiente, non sicuro, o non conforme alle *best practice*.

Alternativa Migliore

Chiedi solo l'aiuto per un piccolo segmento di codice o un singolo concetto bloccante.

Analizza ogni riga di codice generato e chiedi chiarimenti per ogni costrutto che non ti è familiare.

Verifica le informazioni critiche (es. i parametri di una funzione, i comportamenti del compilatore) sulla documentazione C standard (es. man pages, cppreference).

Tenta sempre di risolvere l'errore per almeno 15-30 minuti usando i tuoi strumenti (printf, debugger). Chiedi aiuto all'LLM solo se sei bloccato.

Chiedi sempre una revisione del codice all'LLM stesso, focalizzandoti su efficienza, gestione della memoria e sicurezza.

Cosa non fare (ora): Copilot

C cesare.c •

SIMULAZIONE > CHATGPT > C cesare.c

```
1 // scrivere un programma che implementa il cifrario di Cesare in C,
2 //cifrando e decifrando con passo k
```

C cesare.c •

SIMULAZIONE > CHATGPT > C cesare.c > ...

```
1 // scrivere un programma che implementa il cifrario di Cesare in C,
2 //cifrando e decifrando con passo k
3 #include <stdio.h>
4 #include <string.h>
5 #include <ctype.h>
6 #define MAX_STR 1000
7
8 void cifra(char *c, char *d, int n) {
    int i;
    for (i = 0; c[i] != '\0'; i++) {
        char ch = c[i];
        if (ch >= 'A' && ch <= 'Z') {
            d[i] = ((ch - 'A' + n) % 26) + 'A';
        } else if (ch >= 'a' && ch <= 'z') {
            d[i] = ((ch - 'a' + n) % 26) + 'a';
        } else {
            d[i] = ch;
        }
    }
    d[i] = '\0';
}
```

Cosa non fare (ora): Copilot

C cesare.c 1 ● SIMULAZIONE > CHATGPT > C cesare.c > ...

```
1 // scrivere un programma che implementa il cifrario di Cesare in C,
2 //cifrando e decifrando con passo k
3 #include <stdio.h>
4 #include <string.h>
5 #include <ctype.h>
6 #define MAX_STR 1000
7
8 void cifra(char *c, char *d, int n) {
9     int i;
10    for (i = 0; c[i] != '\0'; i++) {
11        char ch = c[i];
12        if (ch >= 'A' && ch <= 'Z') {
13            d[i] = ((ch - 'A' + n) % 26) + 'A';
14        } else if (ch >= 'a' && ch <= 'z') {
15            d[i] = ((ch - 'a' + n) % 26) + 'a';
16        } else {
17            d[i] = ch;
18        }
19    }
20    d[i] = '\0';
21 }
22
23 void decifra(char *c, char *d, int n) {
24     int i;
25    for (i = 0; c[i] != '\0'; i++) {
26        char ch = c[i];
27        if (ch >= 'A' && ch <= 'Z') {
28            d[i] = ((ch - 'A' - n + 26) % 26) + 'A';
29        } else if (ch >= 'a' && ch <= 'z') {
30            d[i] = ((ch - 'a' - n + 26) % 26) + 'a';
31        } else {
32            d[i] = ch;
33        }
34    }
35    d[i] = '\0';
36 }
```

C cesare.c x SIMULAZIONE > CHATGPT > C cesare.c > main()

```
38 int test() {
39     return 0;
40 }
41
42 int main () {
43     char chiaro[MAX_STR], cifr[MAX_STR], decifr[MAX_STR];
44     int shift, scelta;
45
46     printf("== CIFRARIO DI CESARE ==\n");
47     printf("Inserisci lo shift (1-25): ");
48     if (scanf("%d", &shift) != 1 || shift < 1 || shift > 25) {
49         printf("Errore: inserisci un numero tra 1 e 25\n");
50         return 1;
51     }
52     getchar(); // consuma il newline
53
54     printf("\n1. Cifra\n2. Decifra\nScelta (1-2): ");
55     if (scanf("%d", &scelta) != 1 || (scelta != 1 && scelta != 2)) {
56         printf("Errore: scelta non valida\n");
57         return 1;
58     }
59     getchar(); // consuma il newline
60
61     if (scelta == 1) {
62         printf("Inserisci una stringa in chiaro: ");
63         if (fgets(chiaro, MAX_STR, stdin) == NULL) {
64             printf("Errore nella lettura\n");
65             return 1;
66         }
67         cifra(chiaro, cifr, shift);
68         printf("Stringa cifrata: %s\n", cifr);
69     } else {
70         printf("Inserisci una stringa cifrata: ");
71         if (fgets(cifr, MAX_STR, stdin) == NULL) {
72             printf("Errore nella lettura\n");
73             return 1;
74         }
75         decifra(cifr, decifr, shift);
76         printf("Stringa decifrata: %s\n", decifr);
77     }
78
79     return 0;
80 }
```

Cosa non fare (ora): Copilot



```
SIMULAZIONE > CHATGPT > J Moto.java ●

1 // fare una classe Moto che estende la classe Veicolo, e una classe
2 // noleggio che gestisce il noleggio di moto in una isola turistica. La
3 // classe Moto deve avere un attributo che indica se ha il bauletto o no. La
4 // classe Noleggio deve avere un attributo che indica il numero di moto
5 // disponibili, un metodo per noleggiare una moto (che decrementa il numero
6 // di moto disponibili) e un metodo per restituire una moto (che incrementa il
7 // numero di moto disponibili).
8
9 public class Moto extends Veicolo {
10     private boolean hasBauletto;
11
12     public Moto(String marca, String modello, int anno, boolean hasBauletto) {
13         super(marca, modello, anno);
14         this.hasBauletto = hasBauletto;
15     }
16
17     public boolean hasBauletto() {
18         return hasBauletto;
19     }
20
21     public void setHasBauletto(boolean hasBauletto) {
22         this.hasBauletto = hasBauletto;
23     }
24 }
25
26 public class Noleggio {
27     private int numeroMotoDisponibili;
28
29     public Noleggio(int numeroMotoDisponibili) {
30         this.numeroMotoDisponibili = numeroMotoDisponibili;
31     }
32
33     public boolean noleggiaMoto() {
34         if (numeroMotoDisponibili > 0) {
35             numeroMotoDisponibili--;
36             return true; // Noleggio avvenuto con successo
37         }
38         return false; // Nessuna moto disponibile
39     }
40
41     public void restituisciMoto() {
42         numeroMotoDisponibili++;
43     }
44
45     public int getNumeroMotoDisponibili() {
```

Cosa non fare (ora): Copilot

- I vostri esercizi sono programmi semplici, standard, pensati per imparare a programmare step by step
- Un LLM ha già visto questi esercizi centinaia di volte, quindi riesce a risolverli perfettamente
- Non riesce però a risolvere problemi più complessi
- Quanto a voi, se non imparate a fare esercizi semplici, non riuscirete poi a fare in futuro quelli complessi.

Esempio: una vostra soluzione di Cesare

```
void cifra(char *ptr, int n){
    for(int i=0; *(ptr+i) != '\0'; i++){
        if(ptr[i]>('Z'-n) && ptr[i] <='Z' || ptr[i]> ('z'-n) && ptr[i]<='z' ){
            //printf("caso2: %c:%d %d %d\n",ptr[i], ptr[i], n, ptr[i] +(-26+n));
            ptr[i]= ptr[i] +(-26+n);
        }else{
            //printf("caso1: %c:%d %d %d\n", ptr[i],ptr[i], n, ptr[i+n]);
            ptr[i] += n;
        }
    }
}

void decifra(char *ptr, int n){
    for(int i=0; *(ptr+i) != '\0'; i++){
        if(ptr[i]>=( 'Z'-n) && ptr[i] <'Z' || ptr[i]>= ('z'-n) && ptr[i]<'z' ){
            //printf("caso2: %c:%d %d %d\n",ptr[i], ptr[i], n, ptr[i] +(-26+n));
            ptr[i]= ptr[i] -(-26+n);
        }else{
            //printf("caso1: %c:%d %d %d\n", ptr[i],ptr[i], n, ptr[i+n]);
            ptr[i] -= n;
        }
    }
}
```

Assolutamente. Il codice precedente **funzionava** per la cifratura e decifratura di base, ma conteneva alcune imprecisioni e non era scritto nel modo più **robusto** o **standard** possibile.

Ecco un'analisi dettagliata di ciò che era **corretto** e di ciò che era **problematico** nel codice originale, focalizzandosi sulle funzioni `cifra` e `decifra`.

🔍 Analisi del Codice Originale

✓ Cosa era Corretto (I Punti di Forza)

1. **Logica del Cifrario (Base):** L'idea di base di incrementare/decrementare il valore ASCII di una lettera per spostarla era corretta.
2. **Gestione del Wrap-Around (Tentata):** Il codice tentava di gestire correttamente il *wrap-around* (il ritorno all'inizio dell'alfabeto quando si supera 'Z' o 'z').

✗ Cosa era Problematico (Le Criticità)

1. Logica di Wrap-Around Incompleta e Complessa

Il codice originale usava complessi intervalli per controllare il wrap-around, che non erano sempre corretti o chiari.

Funzione `cifra` (Cifratura):

```
C

// Codice Originale per Cifratura:
if (ptr[i] > ('Z'-n) && ptr[i] <= 'Z' || ptr[i] > ('z'-n) && ptr[i] <= 'z') {
    // Caso 2: Esempio: carattere 'Y', n=5. 'Y' (89) > ('Z'-5) (90-5=85). VERO.
    // ptr[i] = ptr[i] + (-26 + n); // 'Y' + (-26 + 5) = 89 - 21 = 68, che è 'D'.
} else {
    // Caso 1: Esempio: carattere 'A', n=5. 'A' NON > 85.
    // ptr[i] += n; // 'A' + 5 = 'F'. Corretto.
}
```

- Questa logica **funzionava**, ma era difficile da leggere e basata su valori specifici.
- Il codice migliorato usa l'**aritmetica modulare** (es. `(ch - base + shift + 26) % 26`) che è lo standard matematico per la cifratura a sostituzione e rende il codice **più chiaro e meno incline a errori**.

2. Logica di Decifratura (Probabilmente Errata)

La funzione `decifra` usava la stessa logica di intervallo per decidere se applicare il wrap-around, ma i segni erano invertiti in modo non intuitivo.

```
C

// Codice Originale per Decifratura:
if (ptr[i] >= ('Z'-n) && ptr[i] < 'Z' || ptr[i] >= ('z'-n) && ptr[i] < 'z') {
    // In questo caso, applica la formula "negativa" della cifratura:
    // ptr[i] = ptr[i] - (-26 + n);
} else {
    // Altrimenti, decifra normalmente:
    // ptr[i] -= n;
}
```

- La condizione `ptr[i] >= ('Z'-n)` non era la condizione corretta per decidere se avvolgere indietro.
- La formula `ptr[i] = ptr[i] - (-26 + n)` si semplifica in `ptr[i] = ptr[i] + 26 - n`. Se proviamo a decifrare 'D' (68) con $n = 5$:
 - $68 \geq ('Z' - 5) \Rightarrow 68 \geq 85$ FALSO.
 - Si cade nell' `else`: `ptr[i] -= n` $\Rightarrow 68 - 5 = 63$, che è '?'. Questo è sbagliato.

3. Mancata Gestione dei Caratteri Non Alfabetici

- **Originale:** Il codice non verificava se un carattere fosse alfabetico. Spazi, numeri e punteggiatura venivano comunque spostati, alterandoli.
 - Esempio: Spazio (ASCII 32) con $k = 3$ diventava 'A' (ASCII 35).
- **Migliorato:** L'uso di `isalpha()` e il mantenimento dei caratteri non alfabetici è la prassi standard per il Cifrario di Cesare.

💡 Riassunto

Il codice precedente era un tentativo **valido** di implementazione che funzionava per la cifratura, ma la funzione di **decifratura era probabilmente difettosa** e l'intera implementazione era **fragile e non standard**.

L'obiettivo del codice migliorato non era solo correggere gli errori, ma anche renderlo **più leggibile, robusto** e conforme alle migliori pratiche C, utilizzando l'aritmetica modulare per la logica di cifratura.

Esempio: una vostra soluzione di Cesare

```
void cifra(char *ptr, int n){  
    for(int i=0; *(ptr+i) != '\0'; i++){  
        if(ptr[i]>('Z'-n) && ptr[i] <='Z' || ptr[i]> ('z'  
            //printf("caso2: %c:%d %d %d\n",ptr[i],  
            ptr[i]= ptr[i] +(-26+n);  
        }else{  
            //printf("caso1: %c:%d %d %d\n", ptr[i],  
            ptr[i] += n;  
        }  
    }  
  
    void decifra(char *ptr, int n){  
        for(int i=0; *(ptr+i) != '\0'; i++){  
            if(ptr[i]>=( 'Z'-n) && ptr[i] <'Z' || ptr  
                //printf("caso2: %c:%d %d %d\n"  
                ptr[i]= ptr[i] -(-26+n);  
            }else{  
                //printf("caso1: %c:%d %d %d\n"  
                ptr[i] -= n;  
            }  
        }  
    }
```

```
inserisci una stringa da cifrare: abczABCXYZ  
inserisci k: 4  
k: 4  
Stringa cifrata: efgdEFGBCD  
Stringa decifrata: abc`ABC>?@
```

In generale la soluzione proposta non era del tutto corretta, ma si avvicinava ad una corretta.

Era facile però trovare che la funzione `cifra` è abbastanza robusta ed è quasi corretta, ma che la funzione `decifra` ha qualche bug.

Un LLM in questo caso vi aiuta a strutturare meglio la soluzione (anche se non identifica subito il problema principale che è in `decifra`)

Esempio : una vostra soluzione di Cesare

1. Unified Logic with `caesar_cipher`

- **Original:** Two separate functions (`cifra` and `decifra`) with similar, complex logic.
- **Improved:** A single, central function `caesar_cipher` handles both encryption and decryption using a `mode` parameter (`1` for encrypt, `-1` for decrypt). This **reduces code duplication** and makes maintenance easier.

2. Correct and Robust Wrapping (Modulo Arithmetic)

- **Original:** Used complicated `if` conditions to check for wrap-around (`if(ptr[i] > ('Z' - n) ...)`) which were prone to error (e.g., they didn't handle non-alphabetic characters, and the decryption conditions were incorrect).
- **Improved:** The core logic uses the **modulo operator** (`%`) for a mathematically correct and concise shift:

$$\text{new_index} = (\text{original_index} + \text{shift} + 26) \pmod{26}$$

The `+ 26` term ensures that even if `original_index + shift` is negative (which happens during decryption), the result of the modulo operation is positive, correctly handling the wrap-around.

3. Proper Character Handling (`<ctype.h>`)

- **Original:** Did not handle spaces, punctuation, or numbers. Only checks for uppercase and lowercase letters based on a limited range.
- **Improved:**
 - It uses `isalpha(ch)` to only process actual alphabet letters. Other characters (spaces, numbers, punctuation) are ignored, which is standard for Caesar cipher.
 - It uses `isupper(ch)` and `islower(ch)` to correctly determine the ASCII base (`'A'` or `'a'`) regardless of the specific ASCII values, making the code more portable and readable.

4. Input Safety and Robustness

- **Original** `main`: Did not clear the input buffer after `scanf`, which can lead to the newline character (`\n`) being read by the subsequent `fgets`, causing problems.
- **Improved** `main`: Includes an explicit loop to **clear the input buffer** after `scanf`, ensuring `fgets` behaves correctly. Also, `fgets` is used instead of `scanf` for the initial string input, which is generally safer for reading strings.

5. Key Normalization

- **Improved:** The shift key `k` is normalized using `k = k % 26;`. This is crucial because a key of $k = 27$ is functionally identical to $k = 1$, and this normalization handles overly large or negative keys more gracefully.

In questo caso, i consigli dati sono condivisibili (e anche interessanti). Alcuni sono un po' un overkill (es, usare le funzioni in `ctype.h`)

Correzioni dei vostri programmi

```
int main () {  
    char *pnstr;  
    int *pnk;  
  
    cifra(*pnstr, *pnk);  
    decifra(*pnstr, *pnk);  
  
    return 0;  
}
```

Commenti:

Questo codice non può funzionare, la stringa non è mai stata allocata e quindi il puntatore `*pnstr` si riferisce ad una zona undefined di memoria.

Correzioni dei vostri programmi

```
int cifra (char *pnstr, int *pnk) {  
    int i = 0;  
  
    for (str[i]; str[i] != '\0'; str[i++]) {  
        if (str[i] >= 97 && str[i] <= 122) {  
            printf("Alfabeto minuscolo\n");  
            printf("Il carattere è %c\n", str[i]);  
            str[i] = str[i] + k;  
            printf("Il carattere cifrato è %c\n", str[i]);  
        } else {  
            if (str[i] >= 65 && str[i] <= 90) {  
                printf("Alfabeto maiuscolo\n");  
                printf("Il carattere è %c\n", str[i]);  
                str[i] = str[i] + k;  
                printf("Il carattere cifrato è %c\n", str[i]);  
            } else {  
                printf("La stringa inserita contiene dei caratteri non accettati\n");  
            }  
        }  
    }  
    printf("La stringa cifrata è:\n%s\n", str);  
    return str;  
}
```

Commento:

- La stringa `str []` non è definita, quindi questa funzione non compila, e quindi è **sicuramente** sbagliata.
- Usare ‘a’ e ‘z’ e ‘A’ e ‘Z’ invece della loro codifica `int` la leggibilità rispetto ad usare i valori alfanumerici
- Sicuramente la funzione non è stata testata (e difatti la logica non funziona, mancano dei pezzi)

```

int vicini(char **copy, int dim){
    int counter=0;
    for(int i=1; i<dim-1; i++){
        for(int j=1; j<dim-1; j++){
            int counter = 0;
            if(copy[i][j-1]=='#') counter++;
            if(copy[i][j+1]=='#') counter++;
            if(copy[i-1][j-1]=='#') counter++;
            if(copy[i-1][j]=='#') counter++;
            if(copy[i-1][j+1]=='#') counter++;
            if(copy[i+1][j-1]=='#') counter++;
            if(copy[i+1][j]=='#') counter++;
            if(copy[i+1][j+1]=='#') counter++;
        }
    }
    return counter;
}

void nuovaGenerazione(char **sta, int dim){
    // crea copia
    //char copy[dim][dim];
    char ** copy;

    copy = (char **) malloc(dim * sizeof(char *));
    for (int i=0; i<dim; i++)
        copy[i] = (char*) malloc(dim * sizeof(char));

    for(int i=0; i<dim; i++){
        for(int j=0; j<dim; j++){
            copy[i][j]=sta[i][j];
        }
    }

    for(int i=1; i<dim-1; i++){
        for(int j=1; j<dim-1; j++){
            if(copy[i][j]=='*'){
                if(vicini(copy, dim)==3) sta[i][j]='#';
            }else{
                if(vicini(copy, dim > 1 && vicini(copy, dim < 4))) sta[i][j]='#';
            }
        }
    }
}

```

Commenti:

- La funzione “vicini” non controlla i limiti della matrice, e quindi esce dal bordo della matrice stessa.

-1 -1	-1 0	-1 1	-1 2	-1 3
0 -1	0 0	0 1	0 2	0 3
1 -1	1 0	1 1	1 2	1 3
2 -1	2 0	2 1	2 2	1 2
3 -1	3 0	3 1	3 2	3 3

- Inoltre fa due cicli che processano ogni elemento della matrice, in maniera errata, solo un elemento per volta andava testato
- Il controllo con 9 if diversi, oltre ad essere errato, andava fatto con due cicli separati.
- Meglio rappresentare la matrice con valori [0,1] e non con '#' e '*'
→ leggete il testo, in questo caso vi suggeriva di usare degli interi

```

int vicini(char **grid, int dim, int row, int col) {
    int count = 0;

    // Itera su un blocco 3x3 centrato su (row, col)
    for (int i = -1; i <= 1; i++) {
        for (int j = -1; j <= 1; j++) {
            // Salta la cella centrale (i=0, j=0)
            if (i == 0 && j == 0) continue;

            int r = row + i;
            int c = col + j;

            // Controlla che le coordinate siano all'interno dei bordi validi
            // Assumiamo bordi fissi (nessun wrap-around)
            if (r >= 0 && r < dim && c >= 0 && c < dim) {
                if (grid[r][c] == '#') {
                    count++;
                }
            }
        }
    }

    return count;
}

```

copy[4][4] = vicini(copy, 4, 2, 2);

```

for(int i=0; i<dim; i++){
    for(int j=0; j<dim; j++){
        copy[i][j]=sta[i][j];
    }
}

for(int i=1; i<dim-1; i++){
    for(int j=1; j<dim-1; j++){
        if(copy[i][j]=='*'){
            if(vicini(copy, dim)==3) sta[i][j]='#';
        }else{
            if(vicini(copy, dim > 1 && vicini(copy, dim < 4))) sta[i][j]='#';
        }
    }
}

```

Commenti:

- La funzione “vicini” non controlla i limiti della matrice, e quindi esce dal bordo della matrice stessa.

-1 -1	-1 0	-1 1	-1 2	-1 3
0 -1	0 0	0 1	0 2	0 3
1 -1	1 0	1 1	1 2	1 3
2 -1	2 0	2 1	2 2	1 2
3 -1	3 0	3 1	3 2	3 3

- Inoltre fa due cicli che processano ogni elemento della matrice, in maniera errata, solo un elemento per volta andava testato
- Il controllo con 9 if diversi, oltre ad essere errato, andava fatto con due cicli separati.
- Meglio rappresentare la matrice con valori [0,1] e non con '#' e '*'.

LLM vi dà una idea di come migliorare il codice e trova errori, ma la sua soluzione va comunque controllata

```

int vicini(int close [] [N], int a,int b) {
    int i;
    int j;
    int count=0;

    if (a-1>=0 && b-1>=0 && close [a-1] [b-1]==1){
        count++;
    }
    if (a-1>=0 && close [a-1] [b]==1){
        count++;
    }
    if (a-1>=0 && b+1<N && close [a-1] [b+1]==1){
        count++;
    }
    if (b-1>=0 && close [a] [b-1]==1){
        count++;
    }
    if (close [a] [b]==1){
        // count++; ???
    }
    if (b+1<N && close [a] [b+1]==1){
        count++;
    }
    if (a+1<N && b-1>=0 && close [a+1] [b-1]==1){
        count++;
    }
    if (a+1<N && close [a+1] [b]==1){
        count++;
    }
    if (a+1<N && b+1<N && close [a+1] [b+1]==1){
        count++;
    }

    printf("numero di celle vicine vive %d\n",count);

    return count;
}

```

Commenti:

Questo codice anche se – per caso – fa quello che deve fare, è strutturato in maniera poco leggibile e molto facilmente ha dei bug nella gestione dei bordi della matrice.

-1 -1	-1 0	-1 1	-1 2	-1 3
0 -1	0 0	0 1	0 2	0 3
1 -1	1 0	1 1	1 2	1 3
2 -1	2 0	2 1	2 2	1 2
3 -1	3 0	3 1	3 2	3 3

La cosa giusta da fare era fare un doppio ciclo `for` per controllare l'intorno 3x3 della cella considerata.

```

//funzione che calcola la lunghezza delle stringhe
int length(char *str){
    int length = 0;
    for(; str[length] != '\0'; length++);
    return length;
}

void cifra(char *s, int k){
    //itero su ogni carattere della stringa
    for(int i= 0; i < length(s); i++){

        //gestisco lettere maiuscole
        if(s[i] >= 'A' && s[i] <= 'Z'){

            s[i] = (s[i]-'A'+k)%26+'A';
        }

        //gestisco lettere minuscole
        if(s[i] >= 'a' && s[i] <= 'z'){

            s[i] = (s[i]-'a'+k)%26+'a';
        }
    }
}

```

Commenti:

La lunghezza della stringa è fissa; qui viene ricalcolata n volte, dove n è la lunghezza della stringa stessa.

La cosa migliore è fare:

```

int len = length(s);
for(int i= 0; i < len; i++){
    ...
}

```

O ancora meglio:

```

int len = strlen(s);
for(int i= 0; i < len; i++){
    ...
}

```

Usando delle funzioni di libreria.

```

char cifra(char s[N],int k,int n){

    for(int i =0;i<=n;i++){
        if(isalpha(s[i])){

            if(s[i]>'w'&& s[i]<='z'){
                s[i]=s[i]-26;
            }
            if(s[i]>'W'&& s[i]<='Z'){
                s[i]=s[i]-26;
            }
            s[i]=s[i]+k;
        }
    }
    printf("%s\n",s);
    return 0;
}

char decifra(char s[N],int k,int n){
    for(int i =0;i<=n;i++){
        if(isalpha(s[i])){

            s[i]=s[i]-k;
        }
    }
    printf("%s\n",s);
    return 0;
}

```

Commenti:

La funzione `cifra` non deve controllare solo il caso in cui la lettera sia maggiore di '`w`' o di '`Z`' per gestire un overflow quando vado a cifrare, in quanto questo è un caso particolare del problema. Se metto valori alti di `k` la soluzione non è robusta.

Inoltre, come per le altre soluzioni, è meglio usare l'operatore modulo % per andare a fare l'operazione di cifratura.

La funzione `decifra` invece non fa il controllo che il valore, una volta sottratto `k`, sia nel range corretto.

```

// Crea gli array per la cifratura e decifratura
char sCifrata[strlen(riga) + 1]; // +1 per il terminatore di stringa
char sDecifrata[strlen(riga) + 1];

// Cifra e decifra la stringa
cifra(riga, sCifrata);
printf("Ecco la stringa cifrata con k=%d: %s\n", K, sCifrata);

decifra(sCifrata, sDecifrata);
printf("Ecco la stringa decifrata con k=%d: %s\n", K, sDecifrata);

// Libera la memoria allocata per la riga
free(riga);

if (fclose(file) != 0) {
    fprintf(stderr, "Errore nella chiusura del file\n");
    return -1;
}

return 0;
}

void cifra(char *s, char *sCifrata) {
int lunghezza = strlen(s);
for (int i = 0; i < lunghezza; i++) {
    *sCifrata = *s;
    for (int j = 0; j < K; j++) {
        if (*sCifrata < 65 || (*sCifrata > 90 && *sCifrata < 97) || *sCifrata > 122) {
            break;
        }

        if (*sCifrata == 90) {
            *sCifrata = 65;
        } else if (*sCifrata == 122) {
            *sCifrata = 97;
        } else {
            *sCifrata = *sCifrata + 1;
        }
    }
    s++;
    sCifrata++;
}
*sCifrata = '\0';
}

```

Commenti:

La funzione `cifra` è corretta, ma usare i caratteri ASCII invece del corrispettivo valore numerico aumenterebbe la leggibilità del codice.

Domanda: cosa fa questa funzione `cifra` e come fa la conversione dei caratteri?

Usare i VLA come per la dichiarazione di `sCifrata` e di `sDecifrata` è un errore, andava usata una `malloc`

`K` è definita costante, ma meglio definirla come una variabile.

```

int vicini(int a[N][N], int i, int j){
    int contatore=0;
    if(a[i-1][j-1]==1) //in alto a sinistra
        contatore++;
    if(a[i-1][j]==1) //in alto
        contatore++;
    if(a[i-1][j+1]==1) //in alto a destra
        contatore++;
    if(a[i][j-1]==1) //a sinistra
        contatore++;
    if(a[i][j+1]==1) //a destra
        contatore++;
    if(a[i+1][j-1]==1) //in basso a sinistra
        contatore++;
    if(a[i+1][j]==1) //in basso
        contatore++;
    if(a[i+1][j+1]==1) //in basso a destra
        contatore++;
    printf("%d\n", contatore);
    return contatore;
}

int nuovagenerazione (int a[][N]){
    int i,j, b[N][N];
    for(i=0; i<N; i++){
        for(j=0; j<N; j++){
            if(a[i][j]==1){
                if(vicini(a,i,j)<2)
                    b[i][j]=0;
                else if(vicini(a,i,j)==2 || vicini(a,i,j)==3)
                    b[i][j]=1;
                else if(vicini(a,i,j)>3)
                    b[i][j]=0;
            }else if (a[i][j]==0 && vicini(a,i,j)==3)
                b[i][j]=1;
            else
                b[i][j]=a[i][j];
        }
    }
    return b[N][N];
}

```

Commenti:

Qui ci sono due problemi evidenti.

La prima funzione accede anche a dei valori che sono fuori dalla matrice, e inoltre replica il codice facendo tanti *if*, quando in realtà sarebbe stato opportuno fare uno (o due) cicli *for*.

Il secondo problema sta in *b* [N] [N], che vive nel record di allocazione di nuovagenerazione. Quando tale funzione termina, quella area di memoria viene deallocata, ma la funzione che chiama nuovagenerazione avrà un pointer ad essa.

Questo è un undefined behaviour, un errore da evitare.

La soluzione poteva essere copiare al termine dell'update la vecchia generazione nella nuova:

```

for(i=0; i<N; i++)
    for(j=0; j<N; j++)
        a[i][j]=b[i][j];

```

⚠ Analisi del Codice Originale

La funzione `vicini` ha l'obiettivo di calcolare il numero di celle vive (dove il valore è `1`) attorno a una specifica cella identificata da `riga` e `colonna`.

```
int vicini(int mat[N][N], int riga, int colonna){  
    int c = 0;  
    int i,j;  
  
    for(i=0;i<N;i++)  
        for(j=0;j<N;j++)  
    {  
        if((i == riga) && (j == colonna))  
        {  
  
            if(mat[i-1][j-1] == 1 && (riga-1 >= 0 && riga-1 <= N-1) && (colonna-1 <= N-1 && colonna-1 >= 0))  
                c++;  
            if(mat[i-1][j] == 1 && (riga-1 >= 0 && riga-1 <= N-1) && (colonna <= N-1 && colonna >= 0))  
                c++;  
            if(mat[i-1][j+1] == 1 && (riga-1 >= 0 && riga-1 <= N-1) && (colonna+1 <= N-1 && colonna+1 >= 0))  
                c++;  
            if(mat[i][j-1] == 1 && (riga >= 0 && riga <= N-1) && (colonna-1 <= N-1 && colonna-1 >= 0))  
                c++;  
            if(mat[i][j+1] == 1 && (riga >= 0 && riga <= N-1) && (colonna+1 <= N-1 && colonna+1 >= 0))  
                c++;  
            if(mat[i+1][j-1] == 1 && (riga+1 >= 0 && riga+1 <= N-1) && (colonna-1 <= N-1 && colonna-1 >= 0))  
                c++;  
            if(mat[i+1][j] == 1 && (riga+1 >= 0 && riga+1 <= N-1) && (colonna <= N-1 && colonna >= 0))  
                c++;  
            if(mat[i+1][j+1] == 1 && (riga+1 >= 0 && riga+1 <= N-1) && (colonna+1 <= N-1 && colonna+1 >= 0))  
                c++;  
        }  
    }  
  
    return c;  
}
```

1. 🚨 Errore Critico di Logica (Inefficienza Totale)

- **Problema:** La funzione scorre l'intera matrice $N \times N$ utilizzando i cicli `for(i=0; i<N; i++)` e `for(j=0; j<N; j++)`. Il conteggio dei vicini viene eseguito solo quando la cella (i, j) è uguale a quella passata come argomento (`if((i == riga) && (j == colonna))`).
- **Impatto:** I cicli `for` sono completamente inutili e rendono il codice estremamente inefficiente. L'operazione di conteggio (che sono solo 8 `if`) viene eseguita una sola volta dopo aver iterato su tutte le N^2 celle.

Correzione: La funzione non ha bisogno di cicli `for` esterni. Il conteggio dei vicini deve avvenire immediatamente all'interno della funzione.

2. 😐 Problemi di Robustezza e Indici Incoerenti

- **Problema:** La gestione dei bordi è implementata attraverso lunghe condizioni logiche (es. `&& (riga-1 >= 0 && riga-1 <= N-1)`).
- **Incoerenza:** All'interno degli `if`, si usano gli indici globali `i` e `j` per accedere alla matrice (`mat[i-1][j-1]`) ma si usano gli argomenti della funzione (`riga`, `colonna`) per controllare i limiti.
 - Quando si arriva alla condizione `if((i == riga) && (j == colonna))`, allora `i == riga` e `j == colonna`. Usare `mat[i-1][j-1]` è corretto, ma controllare i limiti usando `(riga-1 >= 0 ...)` è ridondante e meno chiaro.
- **Problema di Ordine:** Se non si controllano gli indici negativi o fuori limite prima di accedere alla matrice, si rischia un **Out of Bounds** e un *crash* del programma.
 - Esempio: Se `riga = 0`, l'istruzione `if(mat[i-1][j-1] == 1 ...)` accederà a `mat[-1][...]` prima di valutare la condizione `(riga-1 >= 0 ...)` (a meno che il compilatore non garantisca la valutazione pigra, che è standard, ma la scrittura resta rischiosa e confusa).

3. 💬 Poca Chiarezza e Ripetizione (Non DRY)

- La ripetizione di 8 lunghe istruzioni `if` che controllano individualmente ciascuno degli 8 vicini rende il codice lungo e difficile da mantenere.

"Vibe coding may be a horrible, horrible idea from a maintenance standpoint, if you actually tried to make a product," Torvalds said. "But I think it's a great way...for new people to get involved and get excited about computers and get computers to do something that maybe they couldn't do it, otherwise. So I'm actually fairly positive about this all." That said, he scoffed at the people who "hope to make billion-dollar companies by just using vibe coding."

In general, Torvalds seemed both hopeful and somewhat fatigued by the current wave of automation sweeping the software industry. "I see it as something exciting and something new and something good," he said, while noting that people's expectations may be out of alignment with reality. "We have people who are doing a lot of work in using AI to help maintainers deal with the flow of patches and backboarding patches to stable versions and things like that, but a lot of that is experimental," he added.

"AI is just another tool," the Linux creator went on. "The same way compilers free people from writing assembly code by hand, and increase productivity enormously, but didn't make programmers go away," he said. "I'm looking forward to the day when AI is less hyped and more the everyday reality that nobody talks constantly about," Torvalds noted.
"That's clearly still a few years from now," he said.

