ChatGPT

Prompt Engineering cho GitHub Copilot

Prompt Engineering là gì đối với GitHub Copilot?

Prompt engineering trong bối cảnh GitHub Copilot là nghệ thuật soạn thảo **đầu vào** (prompt) – thường dưới dạng comment hoặc mô tả trong code – một cách có chủ đích để hướng dẫn AI sinh mã sinh ra kết quả mong muốn [1] [2]. Nói đơn giản, **prompt** là những gì bạn "hỏi" Copilot (ví dụ: dòng comment mô tả yêu cầu) và chất lượng của gợi ý code mà Copilot đưa ra phụ thuộc rất lớn vào prompt này – *"junk in, junk out"* (đầu vào tệ thì đầu ra cũng tệ) [3] [2]. Do đó, lập trình viên cần học cách "kỹ thuật prompt" sao cho **rõ ràng, đủ ngữ cảnh và có cấu trúc**, giúp Copilot hiểu đúng yêu cầu và tạo mã phù hợp [4] [5]. Prompt engineering cho Copilot đòi hỏi tư duy như khi giao tiếp với một đồng nghiệp lập trình: cung cấp bối cảnh, hướng dẫn cụ thể và từng bước để cùng "pair-programming" với AI hiệu quả [6] [7].

Nguyên tắc và Best Practices khi viết prompt cho Copilot

Dưới đây là **những nguyên tắc quan trọng** giúp bạn viết prompt hiệu quả cho GitHub Copilot:

- **Cung cấp ngữ cảnh tổng quan trước:** Hãy bắt đầu bằng mô tả cấp cao về mục tiêu hoặc chức năng bạn định xây dựng. Điều này "đặt bối cảnh" cho Copilot hiểu bức tranh lớn trước khi đi vào chi tiết [8] [9]. Ví dụ, mở đầu file bằng comment mô tả *"Xây dựng ứng dụng to-do list với Next.js cho phép người dùng thêm, sửa, xoá việc cần làm"* trước khi chỉ dẫn cụ thể từng phần sẽ giúp Copilot định hướng đúng mục tiêu tổng quát của bạn [10].

- **Chi tiết và cụ thể trong từng yêu cầu:** Sau phần tổng quan, đưa ra hướng dẫn cụ thể về **điều bạn muốn**. Nêu rõ chức năng cần làm gì, đầu vào/đầu ra ra sao, dùng API hay thư viện nào... Càng cụ thể, Copilot càng dễ tạo mã chính xác [11] [12]. Chẳng hạn, thay vì viết chung chung *"// Get data from API"*, hãy viết *"// Lấy dữ liệu người dùng từ API jsonplaceholder.typicode.com, sau đó trả về JSON"* – prompt rõ ràng như vậy sẽ cho Copilot biết **lấy dữ liệu gì, từ đâu và xử lý như thế nào**, tăng khả năng sinh ra hàm đúng ý bạn [13] [12].

- **Đưa ví dụ minh họa trong prompt:** Nếu có thể, hãy **cho Copilot thấy ví dụ cụ thể** về kết quả mong muốn hoặc định dạng dữ liệu. Việc này giúp AI hiểu được pattern bạn cần [14] [15]. Ví dụ, bạn có thể thêm một comment ví dụ đầu ra mong đợi của hàm. Trong trường hợp tạo danh sách số nguyên tố đầu tiên, thêm dòng *"// Example: [2, 3, 5, 7, ...]"* sẽ định hướng Copilot tạo ra code đúng logic hơn (dừng khi tìm đủ 10 số nguyên tố) thay vì chỉ loop đơn giản kém hiệu quả [16] [17]. Cung cấp **ví dụ** giống như cách "dạy" nhanh cho Copilot biết bạn muốn gì, tương tự khái niệm *zero-shot, one-shot, few-shot learning* trong ML (cung cấp một vài ví dụ để AI bắt chước) [18] [19].

- **Chia nhỏ tác vụ phức tạp thành các bước đơn giản:** Tránh yêu cầu Copilot tạo ra một đoạn code quá lớn hoặc giải quyết vấn đề phức tạp chỉ bằng một prompt duy nhất. Thay vào đó, hãy **phân rã vấn đề** thành nhiều bước logic và giải quyết tuần tự [7] [20]. Copilot hoạt động hiệu quả nhất khi bạn code theo kiểu *từng bước một*: viết comment cho bước đầu tiên, để Copilot hoàn thành mã, sau

đó tiếp tục bước tiếp theo [7] [21]. Cách tiếp cận này giống như bạn viết một công thức nấu ăn – liệt kê từng bước thay vì mô tả chung chung cả món ăn [21]. Ví dụ, để viết một word puzzle, bạn có thể nhờ Copilot lần lượt: (1) Tạo lưới 10x10, (2) Viết hàm tìm từ trong lưới, (3) Sử dụng hàm đó để tạo puzzle có ít nhất 10 từ, v.v. thay vì dồn tất cả vào một prompt duy nhất [20].

- **Diễn đạt rõ ràng, tránh mơ hồ:** Sự mơ hồ sẽ khiến AI khó hiểu ý bạn. Vì vậy, **tránh những từ ngữ chung chung** hoặc đại từ không rõ nghĩa trong prompt. Ví dụ, đừng viết *"làm cái này"* hoặc *"what does this do"* khi *"cái này"* có thể hiểu theo nhiều thứ khác nhau – hãy chỉ rõ *"hàm* `createUser` *này có tác dụng gì?"* hoặc *"đoạn code X ở phản hồi trước hoạt động thế nào?"* [22]. Tương tự, nếu code bạn dùng thư viện lạ, nên nói rõ chức năng của thư viện đó, hoặc import sẵn thư viện hay nhắc tên nó trong prompt để Copilot biết nên dùng context nào [23]. Nói chung, **prompt càng rõ ràng, Copilot càng dễ cho ra đáp án đúng**.

- **Thử nghiệm và lặp lại prompt (Iterate):** Đừng nản chí nếu gợi ý đầu tiên của Copilot chưa đúng ý. Kỹ thuật prompt cũng là quá trình thử-sai và tinh chỉnh. Hãy **xóa đoạn code gợi ý chưa đạt**, bổ sung hoặc điều chỉnh prompt với chi tiết/ví dụ cụ thể hơn rồi thử lại [24] [25]. Quá trình này giúp bạn dần hiểu cần thêm bớt thông tin nào để **"nói chuyện" với Copilot hiệu quả hơn** – tương tự như huấn luyện AI qua mỗi lần tương tác [26] [27]. Ngay cả các developer advocate cũng nhấn mạnh việc *"hãy kiên trì thử nghiệm, xem prompt nào hiệu quả"* vì không ai hoàn hảo ngay từ đầu trong kỷ nguyên AI này [27] [28].

- **Mở các file liên quan để cung cấp bối cảnh:** GitHub Copilot chỉ "nhìn" được code trong file hiện tại và một số file đang mở trong editor, **không phải toàn bộ project trừ khi file mở** [29] [30]. Vì vậy, một mẹo hay là **giữ những file quan trọng, định nghĩa liên quan mở trong IDE** khi viết prompt, để Copilot có thêm ngữ cảnh. Thí dụ: nếu bạn sắp viết hàm sử dụng biến hoặc hàm định nghĩa ở file khác, hãy mở file đó song song. Tính năng *"neighboring tabs"* của Copilot sẽ tận dụng các file mở này làm ngữ cảnh bổ sung, giúp đề xuất chính xác hơn [30].

- **Tuân thủ các practice code tốt:** Copilot có xu hướng **học theo phong cách và mẫu** từ chính code của bạn [31] [32]. Do đó, hãy viết code rõ ràng, đặt tên biến/hàm có ý nghĩa cụ thể, nhất quán về style (camelCase vs snake_case, v.v.) – Copilot sẽ dựa vào đó đề xuất đoạn code phù hợp cùng convention [33] [34]. Ngược lại, nếu bạn dùng tên biến vô nghĩa hoặc coding style lộn xộn, AI có thể "bối rối" và cho ra gợi ý kém. Thực nghiệm cho thấy với hàm đặt tên rõ ràng `authenticate_user`, Copilot gợi ý được logic hợp lý, còn nếu cố ý dùng tên khó hiểu như `rndpwd`, Copilot thậm chí chỉ đưa ra comment *"// Code goes here"* do không đoán được mục đích hàm [35] [36]. Tóm lại, **code mẫu mực** không chỉ tốt cho người đọc mà còn giúp Copilot hoạt động hiệu quả hơn. Bạn vẫn cần review, test và đảm bảo chất lượng mã AI tạo ra, nhưng việc viết prompt và code "sạch" ngay từ đầu sẽ giảm thời gian chỉnh sửa [37] [38].

Checklist: Nên và Không nên làm khi viết prompt

Nên làm: Luôn cung cấp bối cảnh và ví dụ cụ thể cho yêu cầu của bạn; chia nhỏ yêu cầu phức tạp thành các bước đơn lẻ; diễn đạt rõ ràng điều bạn muốn; tận dụng comment trong code để mô tả mục đích trước khi viết hàm; tuân thủ phong cách code nhất quán và tên biến/hàm có nghĩa; kiên nhẫn thử lại với prompt chi tiết hơn nếu kết quả chưa phù hợp [39] [24].

Không nên: Tránh viết prompt mơ hồ hoặc quá chung chung (ví dụ: *"làm tính năng X"* mà không nói rõ chi tiết); không nên yêu cầu Copilot tạo cả một đoạn code lớn/phức tạp chỉ với một dòng comment duy nhất [7] [40]; đừng bỏ qua ngữ cảnh (AI không thể đoán ý định nếu bạn không cung cấp thông tin liên quan); hạn chế sử dụng từ ngữ tối nghĩa như *"this," "it,"*... trong prompt; không nên kỳ vọng câu trả lời chính xác tuyệt đối ngay lần đầu – hãy chuẩn bị tinh thần tinh chỉnh prompt nhiều lần [41] [24]. Ngoài ra, **không nên tin tưởng 100%** vào mã Copilot sinh ra nếu chưa kiểm chứng – luôn review và test lại vì bạn mới là người chịu trách nhiệm cuối cùng cho code của mình [42].

Ví dụ prompt hiệu quả với GitHub Copilot

Dưới đây là **một số ví dụ** về prompt tốt và lý do vì sao chúng hiệu quả:

Ví dụ 1: Yêu cầu cụ thể khi gọi API
Giả sử ta muốn Copilot viết hàm lấy dữ liệu người dùng từ một API. Một prompt kém sẽ là chỉ viết:

```
// Get data from API
function getData() {
  // code goes here
}
```

Với comment mơ hồ như trên, Copilot **chỉ có thể đề xuất tên hàm và khung hàm trống**, không đoán được bạn cần lấy dữ liệu gì [13] [43]. Thay vào đó, hãy viết chi tiết yêu cầu trong comment, ví dụ:

```
// Pass in user ids and retrieve user data from jsonplaceholder.typicode.com
API, then return it as a JSON object
async function getUserData(userId) {
    const response = await fetch(`https://jsonplaceholder.typicode.com/users/$
{userId}`);
    const data = await response.json();
    return data;
}
```

Prompt trên nêu rõ **mục tiêu (lấy dữ liệu user)**, nguồn API cụ thể và định dạng đầu ra, nên Copilot có thể tạo ngay phần thân hàm hoàn chỉnh sử dụng `fetch` và trả về JSON tương ứng [12]. Sự khác biệt nằm ở tính cụ thể: prompt càng chỉ rõ *cái gì, ở đâu, như thế nào*, Copilot càng dễ hiểu để sinh mã đúng mục đích.

Ví dụ 2: Đưa ví dụ đầu ra mong đợi
Bạn muốn tạo danh sách 10 số nguyên tố đầu tiên. Nếu chỉ viết:

```
# Create a list of the first 10 prime numbers
primes = []
# Copilot's suggestion...
```

Copilot có thể vẫn hoàn thành yêu cầu nhưng theo cách không tối ưu – ví dụ, nó sẽ lặp từ 2 đến 29 để tìm đủ 10 số nguyên tố, duyệt thừa số không cần thiết [44] [16]. Hãy cải thiện prompt bằng cách thêm một ví dụ kết quả mong muốn ngay trong comment:

```
# Create a list of the first 10 prime numbers
# Example: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
primes = []
for num in range(2, 30):
    if all(num % i != 0 for i in range(2, num)):
        primes.append(num)
    if len(primes) == 10:
        break
print(primes)
```

Nhờ có dòng *Example* liệt kê 10 số nguyên tố đầu tiên, Copilot "hiểu" được đích đến và bổ sung thêm **điều kiện** `break` để dừng vòng lặp khi đã thu thập đủ 10 số [45]. Kết quả là mã sinh ra **hiệu quả hơn**, chỉ duyệt đến khi tìm đủ số nguyên tố yêu cầu thay vì duyệt cố định đến 29. Việc cung cấp ví dụ đã "định hướng" cho Copilot cách giải quyết bài toán sát với mong muốn của bạn [16] [17].

Ví dụ 3: Đặt mục tiêu tổng quát và chia nhỏ bước
Khi bắt đầu với một file mới hoặc dự án mới, bạn nên "set the stage" cho Copilot bằng cách mô tả mục tiêu tổng quan, sau đó chia thành các bước. Chẳng hạn, bạn định viết một **ứng dụng Markdown Editor**. Trước hết, viết ở đầu file một comment mô tả mục tiêu chính và các tính năng chính:

```
/*
Create a basic markdown editor in Next.js with the following features:
- Use React hooks
- State for markdown text with default "type markdown here"
- Text area input for the markdown
- Live preview of the markdown as user types
- Support basic markdown syntax (headers, bold, italics)
- Use react-markdown npm package
*/
```

Với prompt dạng khối mô tả chi tiết như trên, Copilot nhận được bức tranh toàn cảnh về **những gì cần xây dựng** và các yêu cầu cụ thể cho từng phần [46]. Sau đó, bạn có thể tiếp tục làm việc theo từng bước: viết comment `// Tạo component Editor...` rồi để Copilot sinh code cho component đó, kế tiếp `// Thêm chức năng live preview...` và cứ thế hoàn thiện từng phần. Cách tiếp cận *"mục tiêu lớn – nhiều bước nhỏ"* này giúp Copilot luôn có **ngữ cảnh rõ ràng** cho mỗi đoạn code nó sinh ra, giảm thiểu khả năng đi lệch hướng [47] [21]. Thực tế cho thấy với ví dụ Markdown Editor, chỉ chưa đầy 30 giây Copilot đã tạo được một phiên bản đơn giản nhưng chạy được của editor theo đúng spec liệt kê [48] (dù có chỗ chưa hoàn hảo, ví dụ text mặc định hơi khác yêu cầu, nhưng việc đó dễ dàng chỉnh sửa sau) [49]. Điều này minh chứng rằng prompt chi tiết và có cấu trúc sẽ giúp Copilot hiểu và thực thi ý tưởng của bạn nhanh chóng hơn nhiều so với prompt sơ sài.

So sánh viết prompt cho GitHub Copilot vs. ChatGPT

Dù đều là AI ngôn ngữ sinh sinh, **cách thức viết prompt cho GitHub Copilot và ChatGPT có một số khác biệt quan trọng**. Bảng dưới đây tóm tắt những điểm khác nhau chính:

Khía cạnh	Prompt cho **GitHub Copilot** (trong IDE)	Prompt cho **ChatGPT** (đối thoại chatbot)
Cách tương tác	Tích hợp trong IDE, gợi ý mã *theo ngữ cảnh code* khi đang viết code. Developer thường viết prompt dưới dạng **comment hoặc tên hàm, dòng code dở dang** trực tiếp trong file để gợi ý hoàn thành [50]. Copilot hoạt động kiểu *autocomplete*, đóng vai trò "đồng lập trình viên" bên cạnh bạn.	Giao tiếp qua hộp thoại bằng ngôn ngữ tự nhiên. Người dùng nhập yêu cầu hoặc câu hỏi như nói chuyện, có thể bao gồm mô tả, code hoặc thông tin bất kỳ. ChatGPT trả lời dưới dạng đoạn văn hoặc mã trong khung chat – không trực tiếp trong mã nguồn.
Ngữ cảnh sử dụng	Hiểu ngữ cảnh từ **code hiện tại và các file đang mở** trong project. Copilot xem xét ngôn ngữ lập trình (theo đuôi file), code xung quanh và comment bạn vừa viết để đưa ra gợi ý [51]. Mọi lần gõ đều diễn ra trong ngữ cảnh dự án cụ thể của bạn.	Hiểu ngữ cảnh từ **lịch sử hội thoại** mà bạn cung cấp. Mỗi lời nhắn trước đó đều là ngữ liệu cho ChatGPT, đặc biệt lời nhắn cuối được ưu tiên cao [52]. Nếu bạn muốn ChatGPT hiểu code, bạn phải **chép đoạn code đó vào chat** hoặc mô tả rõ – nó không tự đọc được code từ môi trường bên ngoài.
Hình thức prompt	Prompt thường ngắn gọn, đi thẳng vào yêu cầu code (ví dụ: comment "// sort an array of users by age" ngay trên code). Có thể tuần tự nhiều comment liên tiếp để hoàn thành dần tính năng. Copilot ưu tiên **prompt ngắn, cụ thể** và phản hồi bằng đoạn code tương ứng (thường là vài dòng đến một hàm) [53].	Prompt có thể dài và diễn đạt tự do bằng ngôn ngữ tự nhiên, thậm chí nhiều câu. Bạn có thể yêu cầu giải thích, viết code kèm giải thích, hoặc đặt câu hỏi mở. ChatGPT chấp nhận prompt phức tạp và phản hồi có thể rất dài, bao gồm cả giải thích chi tiết bên cạnh mã (nếu được yêu cầu).
Cách nhận kết quả	Copilot **tự động hoàn thành** mã ngay bên trong file khi bạn dừng gõ hoặc nhấn tổ hợp phím. Kết quả thường là code thuần (không kèm giải thích) và bạn có thể chấp nhận (Tab) hoặc bỏ qua. Copilot có thể đề xuất nhiều phương án code khi bạn **ấn tổ hợp yêu cầu gợi ý khác**.	ChatGPT **trả lời dưới dạng tin nhắn**. Nếu bạn yêu cầu code, nó sẽ đưa mã trong định dạng markdown (được format) kèm giải thích (trừ khi bạn bảo không cần). Bạn có thể copy code từ câu trả lời. Để có giải pháp khác, bạn phải yêu cầu lại hoặc nhờ nó sửa đổi.

Khía cạnh	Prompt cho **GitHub Copilot** (trong IDE)	Prompt cho **ChatGPT** (đối thoại chatbot)
Điều chỉnh/ Phản hồi	Để tinh chỉnh kết quả, bạn thường **sửa hoặc bổ sung prompt (comment/code) trong file** và xem Copilot đề xuất lại. Mỗi lần sửa là một lần prompt mới, không có "nhớ" lâu dài ngoài nội dung file hiện tại [24]. (Ghi chú: Copilot Chat trong IDE thì có nhớ lịch sử chat trong phiên làm việc, nhưng Copilot hoàn thành code thông thường thì không nhớ ngoài file/code context).	Bạn có thể **hỏi tiếp** hoặc yêu cầu chỉnh sửa ngay trong hội thoại. ChatGPT duy trì ngữ cảnh qua nhiều lượt trao đổi, nên bạn có thể nói "Hãy sửa lại hàm trên cho trường hợp X" mà nó hiểu được hàm trước đó bạn đưa [54]. Tính tương tác của ChatGPT mang tính hội thoại liên tục, giống như thảo luận với một trợ lý, giúp đào sâu hoặc điều chỉnh đáp án dễ dàng.
Lưu ý đặc thù	Copilot nhạy với **chất lượng code dự án**: code càng rõ ràng, có comment, đặt tên tốt, Copilot gợi ý càng chuẩn [33] [36]. Ngoài ra, Copilot tận dụng *toàn bộ context dự án* (bao gồm ngôn ngữ, thư viện import, các file mở) để hiểu yêu cầu [51], vì vậy nhất quán trong mã và cung cấp đủ thông tin ngữ cảnh là chìa khóa.	ChatGPT thì linh hoạt hơn về ngữ cảnh không lập trình – bạn có thể hỏi bất cứ chủ đề gì. Nó không bị giới hạn bởi codebase hiện tại, nhưng cũng vì thế **không tự biết về project của bạn** trừ khi bạn mô tả. ChatGPT giỏi tạo nội dung diễn giải, ngôn từ lưu loát [55], phù hợp giải thích hoặc viết tài liệu. Tuy nhiên, về code, ChatGPT có thể không tuân thủ 100% style codebase của bạn trừ khi bạn yêu cầu và cung cấp ví dụ.

(Trong bảng trên, so sánh chủ yếu dựa trên trải nghiệm với Copilot code completion *thông thường. Lưu ý GitHub hiện có Copilot Chat (trong IDE) kết hợp ưu điểm của cả hai – cho phép chat với AI trong khi vẫn hiểu ngữ cảnh code dự án. Prompt cho Copilot Chat nhìn chung giống phong cách ChatGPT nhưng vẫn tận dụng được context project như Copilot.)*

Tài liệu tham khảo

- **Prompt engineering for AI: what is prompt engineering and how to get good results from AI engines** – *Michelle Duke (GitHub) trên Dev.to* [56] [2]
- **A Beginner's Guide to Prompt Engineering with GitHub Copilot** – *Rizèl Scarlett (GitHub) trên Dev.to* [57] [58]
- **How to write better prompts for GitHub Copilot** – *GitHub Blog* [59] [60]
- **Prompt Engineering Guide (GitHub Copilot Docs)** – *Tài liệu chính thức* [61] [62]

---

[1] [2] [3] [4] [7] [39] [40] [41] [42] [51] [52] [54] [55] [56] Prompt engineering for AI: what is prompt engineering and how to get good results from AI engines - DEV Community
https://dev.to/github/prompt-engineering-for-ai-what-is-prompt-engineering-and-how-to-get-good-results-from-ai-engines-5ch6

5 8 10 11 12 13 14 15 16 17 18 19 24 26 29 31 32 33 34 37 38 43 44 45 47 57 58 A Beginner's Guide to Prompt Engineering with GitHub Copilot - DEV Community
https://dev.to/github/a-beginners-guide-to-prompt-engineering-with-github-copilot-3ibp

6 9 21 25 27 28 30 35 36 46 48 49 50 53 59 60 How to write better prompts for GitHub Copilot - The GitHub Blog
https://github.blog/developer-skills/github/how-to-write-better-prompts-for-github-copilot/

20 22 23 61 62 Prompt engineering for Copilot Chat - GitHub Docs
https://docs.github.com/en/copilot/concepts/prompt-engineering