Công ty Cổ phần VCCorp

BÁO CÁO TUẦN 2

Tìm hiểu Prompting Engineering for Developer

Tác giả: Trần Văn Toàn

Người hướng dẫn: Anh Ngô Văn Vĩ

Hà Nội, tháng 6 năm 2025

Tóm tắt nội dung

Báo cáo này trình bày các khái niệm, đặc điểm, cấu trúc và các phương pháp phổ biến trong Prompt Engineering, một kỹ thuật quan trọng giúp lập trình viên giao tiếp hiệu quả với mô hình ngôn ngữ lớn (LLM). Bên cạnh đó, báo cáo còn nêu rõ các ứng dụng thực tế trong phát triển phần mềm.

Mục lục

Chương 1

Giới thiệu chung

Trong thời đại AI phát triển mạnh mẽ, việc biết cách giao tiếp hiệu quả với mô hình AI là một kỹ năng quan trọng. "Prompt Engineering" không chỉ giúp AI hiểu chính xác yêu cầu của bạn mà còn giúp tiết kiệm thời gian và tối ưu hóa kết quả đầu ra. Dưới đây là các kỹ thuật quan trọng giúp anh em developer khai thác tối đa sức mạnh của AI kèm theo ví dụ cụ thể.

Chương 2

Khái niệm

Prompt Engineering for Developers là quá trình thiết kế và điều chỉnh lời nhắc (prompt) — tức là cách bạn giao tiếp với mô hình ngôn ngữ AI (LLM) — nhằm giúp AI hiểu đúng yêu cầu và tạo ra kết quả chính xác, có thể sử dụng trong phát triển phần mềm.

Prompt engineering for dev = Nghệ thuật giao tiếp thông minh với AI để nó giúp bạn viết code và phát triển phần mềm hiệu quả hơn.

Chương 3

Cấu trúc cơ bản

3.1 Instruction

Trong kỹ thuật *Prompt Engineering*, phần **Instruction** đóng vai trò là chỉ dẫn hành động chính mà người dùng muốn mô hình ngôn ngữ thực hiện. Đây là thành phần quan trọng nhất vì nó định hướng cho toàn bộ kết quả trả lời của mô hình.

Một **Instruction** hiệu quả cần đảm bảo các yếu tố sau:

- **Cụ thể**: Chỉ rõ hành động cần thực hiện (ví dụ: viết hàm, phân tích mã, tạo test case...).
- **Ngắn gọn và trực tiếp**: Dùng lối diễn đạt rõ ràng, sử dụng câu mệnh lệnh.
- **Phù hợp với ngữ cảnh lập trình**: Chỉ ra công nghệ, ngôn ngữ lập trình hoặc yêu cầu đặc thù.

> **Ví dụ:**
> Viết một hàm bằng Python để kiểm tra xem một chuỗi có phải là Palindrome hay không.

Một *Instruction* không rõ ràng, chẳng hạn như *"giúp tôi với đoạn code này"* sẽ khiến mô hình trả lời kém chính xác hoặc hiểu sai yêu cầu.

3.2 Role

Phần **Role** (vai trò) cho phép người dùng giả lập một nhân vật cụ thể cho mô hình, từ đó điều chỉnh ngữ điệu, cách diễn đạt và kiến thức chuyên môn trong phản hồi. Việc gán vai trò giúp AI "đóng vai" như một chuyên gia phù hợp với bối cảnh kỹ thuật.

Các vai trò phổ biến trong lĩnh vực phần mềm bao gồm:

- Lập trình viên backend/front-end cấp cao
- Chuyên gia kiểm thử phần mềm (QA engineer)
- Mentor (người hướng dẫn)
- Kiến trúc sư phần mềm (Software Architect)
- Chuyên gia bảo mật ứng dụng

Việc đặt **Role** đặc biệt quan trọng khi cần phản hồi có chiều sâu chuyên môn hoặc phong cách phù hợp với một đối tượng sử dụng cụ thể.

> **Ví dụ:**
> Bạn là một lập trình viên backend có 10 năm kinh nghiệm làm việc với Node.js và Express.

Khi mô hình nhận được vai trò phù hợp, nó sẽ ưu tiên đưa ra giải pháp sát với thực tế làm việc trong ngành nghề đó.

3.3 Context

Context là phần cung cấp ngữ cảnh, dữ kiện và tình huống cụ thể giúp mô hình hiểu sâu hơn về vấn đề đang được đặt ra. Trong môi trường phát triển phần mềm, bối cảnh có thể bao gồm:

- Đoạn mã nguồn đang xử lý
- Dữ liệu đầu vào/đầu ra mẫu
- Ràng buộc kỹ thuật hoặc phi chức năng (hiệu suất, bảo mật, khả năng mở rộng...)

- Môi trường công nghệ sử dụng (framework, ngôn ngữ, thư viện, công cụ)

Context đóng vai trò thiết yếu trong việc tránh hiểu nhầm và làm rõ yêu cầu.

> **Ví dụ:**
> Hàm sẽ xử lý một mảng các đối tượng sản phẩm dạng JSON, mỗi phần tử có cấu trúc `{ name: string, price: number }`.

Context càng chi tiết, mô hình càng dễ đưa ra câu trả lời chính xác và phù hợp hơn với kỳ vọng của người dùng.

3.4 Output Format

Phần **Output Format** được sử dụng để quy định định dạng đầu ra mong muốn từ mô hình ngôn ngữ. Điều này giúp đảm bảo rằng phản hồi của AI có thể được sử dụng trực tiếp hoặc dễ dàng tích hợp vào các hệ thống, tài liệu, hoặc môi trường phát triển phần mềm.

Các định dạng đầu ra phổ biến bao gồm:

- Mã nguồn được bọc trong khối Markdown (ví dụ: `\\\js` hoặc `\\\python`)
- JSON hoặc XML (dùng trong API hoặc cấu hình)
- Văn bản thuần (plain text)
- Bảng (Markdown table)
- Tài liệu mô tả (dạng tiêu đề, gạch đầu dòng, v.v.)

> **Ví dụ:**
> Trả về đoạn mã JavaScript trong khối `\\\js`, không kèm theo lời giải thích.

Một **Output Format** rõ ràng sẽ giúp mô hình tránh thêm các nội dung thừa như lời dẫn, giải thích, hoặc định dạng không đồng nhất.

Chương 4

Các phương pháp Prompting phổ biến

4.1 Các phương pháp Prompting hiệu quả cho Developer

Trong quá trình áp dụng Prompt Engineering vào lĩnh vực phát triển phần mềm, việc lựa chọn phương pháp prompting phù hợp có ảnh hưởng lớn đến độ chính xác, hiệu quả và khả năng sử dụng đầu ra từ mô hình ngôn ngữ lớn (LLM). Dưới đây là một số kỹ thuật prompting phổ biến, cùng với ví dụ và lợi ích thực tiễn khi áp dụng vào môi trường lập trình.

4.1.1 Chain-of-Thought (CoT) Prompting – Buộc AI suy luận theo từng bước

Phương pháp này yêu cầu AI trình bày quá trình suy luận theo từng bước, đặc biệt hữu ích trong các bài toán có độ phức tạp cao như tối ưu hóa thuật toán, phân tích hiệu suất, hoặc gỡ lỗi chuyên sâu.

Bad Prompt

Viết một thuật toán trong Python để tìm dãy con có tổng lớn nhất trong một mảng số nguyên.

Better Prompt

Hãy tìm dãy con có tổng lớn nhất trong một mảng số nguyên. Giải quyết theo từng bước sau:

1. Trước tiên, hãy liệt kê tất cả các phương pháp có thể áp dụng để giải quyết bài toán này (Brute Force, Dynamic Programming, Kadane's Algorithm, etc.).
2. Giải thích cách tiếp cận từng phương pháp và phân tích độ phức tạp thời gian của chúng.
3. Viết code triển khai phương pháp tối ưu nhất (Kadane's Algorithm) bằng Python.
4. Tối ưu hóa mã nguồn để xử lý mảng có kích thước lớn mà vẫn đảm bảo hiệu suất.
5. Viết unit test với các test case gồm số âm, số dương, và trường hợp rỗng.

Lợi ích: AI không chỉ đưa ra code ngay lập tức mà còn phân tích các phương pháp khác nhau, giúp người dùng hiểu rõ cách giải quyết tối ưu nhất.

4.1.2 Few-shot Prompting – Hướng dẫn AI bằng ví dụ cụ thể

Phương pháp này cung cấp cho AI một hoặc vài ví dụ mẫu để định hình cách thức xử lý và định dạng đầu ra. Rất hiệu quả khi yêu cầu AI tuân theo coding style cụ thể.

Bad Prompt

Viết code TypeScript để tạo một API GET `/users`.

Better Prompt

Dưới đây là một API GET cho danh sách sản phẩm trong ExpressJS:

Listing 4.1: TypeScript Example

```
import express from 'express';
const app = express();

app.get('/products', (req, res) => {
    res.json([{ id: 1, name: 'Laptop' }]);
});

app.listen(3000, () => console.log('Server running on port
   3000'));
```

Hãy viết một API GET `/users` với cấu trúc tương tự.

Lợi ích: AI sẽ bám sát vào coding style, giảm khả năng sinh ra code sai format hoặc không đồng nhất.

4.1.3 Role-based Prompting – Giao vai trò để định hình phong cách trả lời

Việc gán vai trò chuyên môn giúp AI điều chỉnh ngôn ngữ, độ chi tiết và cách giải thích phù hợp với trình độ hoặc mục tiêu sử dụng của người dùng.

Bad Prompt

Giải thích cách hoạt động của Redux.

Better Prompt

Bạn là một Senior Frontend Developer có kinh nghiệm với React và Redux. Hãy giải thích cách Redux hoạt động cho một lập trình viên mới học React, kèm theo ví dụ code.

Lợi ích: AI sẽ viết câu trả lời phù hợp với trình độ người đọc (ví dụ: beginner, senior, team lead), tăng hiệu quả tiếp thu và học tập.

4.1.4 Instruction Tuning – Điều chỉnh chỉ dẫn để AI trả lời đúng trọng tâm

Kỹ thuật này tập trung vào việc viết phần **Instruction** thật rõ ràng và ràng buộc các thông tin đầu vào để tránh AI trả lời sai trọng tâm hoặc thiếu chi tiết.

Bad Prompt

Viết một API để xác thực người dùng.

Better Prompt

Viết một API xác thực người dùng bằng Node.js và Express. API gồm:

- Endpoint POST `/login` nhận email và password.
- Kiểm tra xem người dùng có tồn tại trong database MongoDB hay không.
- Nếu đúng, trả về JWT token.
- Nếu sai, trả về lỗi HTTP 401.

Code cần sử dụng thư viện `jsonwebtoken` và `bcrypt` để hash password.

Lợi ích: Giúp AI hiểu đúng yêu cầu kỹ thuật và trả về code phù hợp hơn với môi trường dự kiến.

Chương 5

Kết luận

5.1 Tầm quan trọng của Prompting đối với lập trình viên

Prompting không chỉ đơn thuần là việc đặt câu hỏi cho AI, mà còn là một nghệ thuật trong việc hướng dẫn mô hình ngôn ngữ theo cách giúp người dùng nhận được kết quả chính xác, hiệu quả và phù hợp với ngữ cảnh công việc.

Đối với lập trình viên, việc tối ưu prompt mang lại nhiều lợi ích thiết thực:

- **Debug code nhanh hơn**: Giải thích lỗi, gợi ý cách sửa lỗi một cách rõ ràng và trực tiếp.
- **Viết API chuẩn format**: Sinh mã nguồn nhất quán, bám sát coding style và framework đang sử dụng.
- **Hiểu thuật toán một cách trực quan**: Diễn giải chi tiết từng bước trong thuật toán, so sánh các cách tiếp cận khác nhau.
- **Học nhanh các công nghệ mới**: Tự tạo hướng dẫn học tập cá nhân hoá theo lộ trình, kiến thức nền tảng, và ví dụ thực hành.

Khuyến nghị: Nếu bạn chưa áp dụng các kỹ thuật prompting như *Chain-of-Thought*, *Few-shot*, *Role-based Prompting*, hoặc *Instruction Tuning*, hãy thử ngay trong lần làm việc tiếp theo với AI. Chỉ cần một prompt tốt, bạn sẽ tiết kiệm thời gian và nâng cao chất lượng công việc lập trình một cách rõ rệt.