TRƯỜNG ĐẠI HỌC TRÀ VINH
TRƯỜNG KỸ THUẬT & CÔNG NGHỆ

ISO 9001:2015

PHẠM HỮU LỘC

# XÂY DỰNG CƠ CHẾ PHÁT HIỆN AI-GENERATED CODE TRONG BÀI TẬP LẬP TRÌNH SINH VIÊN

KHÓA LUẬN TỐT NGHIỆP
NGÀNH CÔNG NGHỆ THÔNG TIN

TRÀ VINH, NĂM 2025

**TRƯỜNG ĐẠI HỌC TRÀ VINH**
**TRƯỜNG KỸ THUẬT & CÔNG NGHỆ**

ISO 9001:2015

# XÂY DỰNG CƠ CHẾ PHÁT HIỆN AI-GENERATED CODE TRONG BÀI TẬP LẬP TRÌNH SINH VIÊN

## KHÓA LUẬN TỐT NGHIỆP
## NGÀNH CÔNG NGHỆ THÔNG TIN

Giảng viên hướng dẫn: **TS NGUYỄN BẢO ÂN**

Sinh viên thực hiện: **PHẠM HỮU LỘC**

Mã số sinh viên: **110121055**

Lớp: **DA21TTB**

Khoá: **2021**

**TRÀ VINH, NĂM 2025**

# LỜI MỞ ĐẦU

Trong bối cảnh công nghệ trí tuệ nhân tạo phát triển mạnh mẽ và ngày càng đóng vai trò quan trọng trong lĩnh vực lập trình, việc sử dụng các công cụ hỗ trợ tạo mã nguồn như GitHub Copilot, ChatGPT, Claude,... đã trở thành xu hướng phổ biến và mang lại hiệu quả cao cho lập trình viên. Tuy nhiên, sự phát triển này cũng đặt ra những thách thức mới, đặc biệt trong việc đảm bảo tính toàn vẹn học thuật thách thức trong việc đánh giá năng lực thực sự của sinh viên, đặc biệt là trong các môn học cơ bản như Kỹ thuật lập trình hay Cấu trúc dữ liệu. Đề tài này phát triển một cơ chế thể nhằm hỗ trợ việc phát hiện mã nguồn được tạo bởi AI thông qua việc phân tích các đặc trưng của mã nguồn mà không cần sử dụng các mô hình học sâu phức tạp.

Đề tài là sự kết hợp giữa mong muốn đóng góp vào việc giải quyết những vấn đề thực tiễn trong giáo dục và phát triển phần mềm, đồng thời là cơ hội để áp dụng những kiến thức đã học đã học vào thực tiễn. Thông qua việc tích hợp các kỹ thuật phân tích AST (Abstract Syntax Tree), độ phức tạp, độ lặp lại và các đặc điểm về cách đặt tên biến/hàm giúp nâng cao hiểu biết về kỹ thuật phân tích mã nguồn.

Tôi hi vọng rằng đề tài này sẽ không chỉ dừng lại ở giá trị học thuật mà còn có khả năng ứng dụng rộng rãi trong thực tế, góp phần hỗ trợ giảng viên, giáo viên đánh giá được năng lực của sinh viên, học sinh.

# LỜI CẢM ƠN

Trước hết, tôi xin được gửi lời cảm ơn đến toàn thể quý thầy cô, giảng viên Trường Đại học Trà Vinh, đặc biệt là các thầy cô ở Khoa Kỹ thuật và Công nghệ, Bộ môn Công nghệ thông tin, đã tạo điều kiện để tôi hoàn thành đề tài này

Để hoàn thành đề tài này, tôi xin chân thành cảm ơn các thầy cô giảng viên đã tận tình hướng dẫn, giảng dạy trong suốt quá trình học tập, nghiên cứu ở Trường Đại học Trà Vinh. Đặc biệt xin gửi lời cảm ơn chân thành tới giảng viên hướng dẫn thầy Nguyễn Bảo Ân đã tận tình, chu đáo hướng dẫn tôi thực hiện đề tài này.

Mặc dù đã có nhiều cố gắng để thực hiện đề tài một cách hoàn chỉnh nhất. Song do thời gian có hạn và còn hạn chế về kiến thức cho nên trong đồ án không thể tránh khỏi những thiếu sót. Em rất mong nhận được sự đóng góp ý kiến của thầy cô và bạn bè để tôi có thể hoàn thiện đồ án này tốt hơn.

Cuối cùng, tôi kính chúc quý thầy cô dồi dào sức khỏe, luôn thành công trong sự nghiệp giảng dạy và đạt được nhiều thành tựu đáng tự hào trong cuộc sống. Một lần nữa, tôi xin chân thành cảm ơn!

Sinh viên thực hiện

Phạm Hữu Lộc

# NHẬN XÉT

*(Của giảng viên hướng dẫn trong đồ án, khóa luận của sinh viên)*

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

**Giảng viên hướng dẫn**

*(Ký và ghi rõ họ tên)*

# NHẬN XÉT

*(Của giảng viên hướng dẫn trong đồ án, khóa luận của sinh viên)*

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

**Giảng viên hướng dẫn**

*(Ký và ghi rõ họ tên)*

UBND TỈNH TRÀ VINH
**TRƯỜNG ĐẠI HỌC TRÀ VINH**

**CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM**
**Độc lập – Tự do – Hạnh Phúc**

# BẢN NHẬN XÉT ĐỒ ÁN, KHÓA LUẬN TỐT NGHIỆP

*(Của giảng viên hướng dẫn)*

Họ và tên sinh viên: Phạm Hữu Lộc MSSV: 110121055

Ngành: Công nghệ Thông tin. Khóa: 2021

Tên đề tài: Xây dựng cơ chế phát hiện AI-generated code trong bài tập lập trình sinh viên

Họ và tên Giáo viên hướng dẫn: Nguyễn Bảo Ân

Chức danh: Giảng viên Học vị: Tiến sĩ

**NHẬN XÉT**

1. Nội dung đề tài:

.....

.....

.....

.....

.....

.....

.....

2. Ưu điểm:

.....

.....

.....

.....

3. Khuyết điểm:

.....

.....

.....

.....

4. Điểm mới đề tài:

.....

.....

.....

.....

.....

5. Giá trị thực trên đề tài:

.....

.....

.....

.....

.....

.....

.....

6. Đề nghị sửa chữa bổ sung:

.....

.....

.....

.....

.....

.....

.....

.....

7. Đánh giá:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Trà Vinh, *ngày tháng năm 20…*

**Giảng viên hướng dẫn**

*(Ký & ghi rõ họ tên)*

UBND TỈNH TRÀ VINH
**TRƯỜNG ĐẠI HỌC TRÀ VINH**

**CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM**
**Độc lập – Tự do – Hạnh Phúc**

# BẢN NHẬN XÉT ĐỒ ÁN, KHÓA LUẬN TỐT NGHIỆP

*(Của cán bộ chấm đồ án, khóa luận)*

Họ và tên người nhận xét: .....

Chức danh: ..... Học vị: .....

Chuyên ngành: .....

Cơ quan công tác: .....

Tên sinh viên: .....

Tên đề tài đồ án, khóa luận tốt nghiệp: .....

.....

.....

## I. Ý KIẾN NHẬN XÉT

1. Nội dung:

.....

.....

.....

.....

.....

.....

.....

.....

.....

2. Điểm mới các kết quả của đồ án, khóa luận:

.....

.....

.....

3. Ứng dụng thực tế:

.....

.....

.....

.....

.....

.....

.....

## II. CÁC VẤN ĐỀ CẦN LÀM RÕ

(Các câu hỏi của giáo viên phản biện)

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

## III. KẾT LUẬN

(Ghi rõ đồng ý hay không đồng ý cho bảo vệ đồ án khóa luận tốt nghiệp)

.....

.....

.....

.....

.....

*……, ngày …… tháng …… năm 20…*

**Người nhận xét**

*(Ký & ghi rõ họ tên)*

# MỤC LỤC

# DANH MỤC CÁC BẢNG, SƠ ĐỒ, HÌNH

# KÍ HIỆU CÁC CỤM TỪ VIẾT TẮT

| STT | Kí hiệu viết tắt | Nội dung viết tắt |
|---|---|---|
| 1 | AI | Artificial Intelligence |
| 2 | AST | Abstract Syntax Tree |
| 3 | API | Application Programming Interface |
| 4 | SSR | Static Site Generation |
| 5 | CSR | Client-Side Rendering |
| 6 | MDX | Markdown eXtended |
| 7 | LOC | Line Of Code |

# CHƯƠNG 1. TỔNG QUAN NGHIÊN CỨU

## 1.1. Giới thiệu đề tài

Trong bối cảnh hiện nay, các công cụ AI hỗ trợ lập trình như GitHub Copilot, ChatGPT, Claude và rất nhiều công cụ AI khác đang phát triển mạnh mẽ. Điều này đặt ra một vấn đề thực tế là ngày càng khó khăn trong việc phân biệt mã nguồn được tạo bởi AI so với mã nguồn do con người viết trong các lĩnh vực giáo dục, tuyển dụng và đánh giá mã nguồn. Đặc biệt, sự phổ biến của các công cụ AI trong việc giải bài tập lập trình của sinh viên, nhất là trong các môn học cơ bản như Kỹ thuật lập trình hay Cấu trúc dữ liệu, đã tạo ra thách thức lớn trong việc đảm bảo tính công bằng và đánh giá đúng năng lực thực tế của sinh viên, cũng như vấn đề liêm chính học thuật và phát hiện đạo văn trong các khóa học lập trình.

Đề tài này tập trung vào việc xây dựng một hệ thống phân tích và đánh giá mã nguồn. Hệ thống này sẽ cung cấp các chỉ số và thông số đặc trưng nhằm hỗ trợ người dùng phân biệt giữa mã nguồn do sinh viên viết và mã nguồn được tạo bởi các công cụ AI. Hệ thống không nhằm mục tiêu khẳng định tuyệt đối một đoạn mã là AI-generated hay không, mà đóng vai trò là công cụ tham khảo, giúp giảng viên, cán bộ chấm thi hoặc quản trị viên có thêm căn cứ khi đánh giá bài làm

## 1.2. Mục đích nghiên cứu

Mục tiêu chính của đồ án này là xây dựng một hệ thống phân tích và đưa ra các đặc trưng để phát hiện mã nguồn được tạo bởi và mã nguồn do con người viết.

Để đạt được mục tiêu này, đề tài tập trung vào các mục tiêu cụ thể sau:

- Trích xuất đặc trưng (Feature extraction): Xây dựng cơ chế trích xuất đặc trưng từ mã nguồn, bao gồm các đặc trưng dựa trên Cây cú pháp (Abstract Syntax Tree - AST), phong cách viết code (style), các chỉ số độ phức tạp (complexity metrics), patterns đặc trưng của AI, và các đặc điểm về spacing, indentation, naming conventions.

- Phân loại bằng phương pháp heuristic tối ưu (Optimized heuristic classification): Phát triển hệ thống phân loại dựa trên quy tắc (rule-based) với trọng số được tối ưu hóa từ phân tích dữ liệu thống kê. Đề tài xây dựng cơ chế đánh giá kết hợp dựa trên luật và điểm số (Heuristic Scoring) và mô hình phân loại nhị phân (Binary Classifier).
- Giao diện web: Phát triển một ứng dụng web hoàn chỉnh bằng Next.js với giao diện thân thiện. Hệ thống sẽ hiển thị kết quả dưới dạng bảng thông số và báo cáo phân tích, giúp người dùng tự đưa ra quyết định

Mục tiêu nghiên cứu của đề tài tập trung vào việc tìm hiểu sâu về các đặc trưng của mã nguồn do AI tạo ra, từ đó phát triển các phương pháp phân tích và phát hiện hiệu quả. Quá trình thực hiện đề tài giúp củng cố kiến thức về phân tích mã nguồn, hiểu rõ về cấu trúc và đặc điểm của mã do con người và AI tạo ra. Việc thu thập và xây dựng tập dữ liệu đối chiếu giúp phát triển kỹ năng xử lý dữ liệu thực tế, giúp nâng cao hiểu biết về kỹ thuật phân tích mã nguồn.

## 1.3. Đối tượng nghiên cứu

Dựa trên mục tiêu chính là xây dựng một hệ thống phân tích và phát hiện mã nguồn được tạo bởi AI so với mã nguồn do con người viết, đối tượng nghiên cứu của đề tài tập trung vào các khía cạnh chính sau:

**Mã nguồn C/C++:** Đây là đối tượng nghiên cứu chính của đề tài, bao gồm các tệp mã nguồn C/C++. Đề tài đi sâu vào nghiên cứu các đặc điểm của mã nguồn bao gồm cấu trúc, cú pháp, các chỉ số độ phức tạp và các dấu hiệu đặc trưng có thể nhận diện mã được tạo ra bởi AI. Các đặc tính này là nền tảng cho việc trích xuất đặc trưng và phân loại.

**Bộ dữ liệu mã nguồn:** Đối tượng nghiên cứu bao gồm việc thu thập và thiết kế một bộ dữ liệu đa dạng.

**Các kỹ thuật trích xuất đặc trưng:** Đề tài nghiên cứu các phương pháp để trích xuất các đặc trưng của mã nguồn. Các kỹ thuật này bao gồm phân tích cây cú

pháp trừu tượng (AST) và các kỹ thuật phân tích mã tĩnh khác. Các đặc trưng được phân loại thành các nhóm chính:

- Đặc trưng cấu trúc (Structure Metrics): Liên quan đến cấu trúc của mã như tổng số node, độ sâu của AST, yếu tố phân nhánh và các chỉ số về hàm.
- Đặc trưng phong cách (Style Metrics): Tính nhất quán trong phong cách mã hóa như tỷ lệ lỗi về khoảng trắng, tính đồng nhất trong thụt lề, phân tích mẫu đặt tên và chất lượng định dạng.
- Các đặc trưng nâng cao (Advanced Features): Các đặc trưng phức tạp hơn như phát hiện sự dư thừa, việc sử dụng template, các mẫu xử lý lỗi.

## 1.4. Phạm vi nghiên cứu

Về ngôn ngữ lập trình: Đề tài tập trung vào việc xử lý và phân tích mã nguồn được viết bằng ngôn ngữ C/C++. Hệ thống được thiết kế với khả năng mở rộng để hỗ trợ các ngôn ngữ phổ biến khác như Python, Java hoặc các mã nguồn phát triển web trong tương lai.

Về phương pháp phân tích: Để cung cấp một cái nhìn toàn diện và linh hoạt trong việc đánh giá mã nguồn, hệ thống tích hợp và triển khai bốn phương pháp phân tích chính:

- Phân tích tổng hợp: Sử dụng toàn bộ hơn 100 đặc trưng từ tất cả các phương pháp để đưa ra đánh giá
- Phân tích cấu trúc AST: Tập trung vào các đặc trưng cấu trúc dựa trên Cây cú pháp, gồm các đặc trưng total nodes, max depth, branching factor, control flow density, và function metrics.
- Phân tích phong cách con người: Dựa trên các đặc trưng về phong cách viết mã, bao gồm các đặc trưng như lỗi về các khoảng trắng, cách đặt tên, vị trí của các cặp ngoặc,…

Về phương pháp đánh giá: Hệ thống áp dụng phương pháp heuristic tối ưu hóa (optimized heuristic classification) thay vì machine learning truyền thống.

## 1.5. Phương pháp nghiên cứu

*Phương pháp nghiên cứu lý thuyết:* Quá trình tìm hiểu lý thuyết bao gồm việc nghiên cứu sâu về các phương pháp trích xuất đặc trưng từ mã nguồn thông qua Cây cú pháp trừu tượng, các kỹ thuật phân tích phong cách lập trình và độ phức tạp code, cùng với những pattern đặc trưng trong mã nguồn do AI tạo ra. Đồng thời, đề tài cũng tham khảo các nghiên cứu về phương pháp heuristic và thuật toán phân loại không giám sát. Ngoài ra, việc khảo sát các công cụ và hệ thống hiện có trong lĩnh vực code analysis như SonarQube, CodeClimate và các nghiên cứu về AI code generation từ GPT, Copilot.

*Phương pháp nghiên cứu thực nghiệm:* Sau khi có nền tảng lý thuyết vững chắc, quá trình triển khai hệ thống được thực hiện thông qua việc xây dựng một kiến trúc hoàn chỉnh gồm backend API sử dụng FastAPI và frontend tương tác bằng Next.js.

# CHƯƠNG 2. NGHIÊN CỨU LÝ THUYẾT

## 2.1. Next.js Framework

### 2.1.1 Giới thiệu tổng quan về Next.js

Next.js là một framework phát triển web mã nguồn mở được phát triển dựa trên nền tảng React. Framework này được thiết kế để mang lại trải nghiệm phát triển tốt nhất với tất cả các tính năng cần thiết được tích hợp sẵn như rendering phía máy chủ và tạo trang tĩnh cho phép chúng ta xây dựng các trang web tĩnh có tốc độ siêu nhanh. Next.js được ra đời vào năm 2016, thuộc sở hữu của Vercel. Next.js bắt đầu trở nên phổ biến vào năm 2018 và tiếp tục tăng trưởng mạnh mẽ trong cộng đồng phát triển web.

Tính đến tháng 11/2024, Next.js đã thu hút hơn 291.000 sao trên Github [1], cho thấy sự phổ biến và tin tưởng của cộng đồng developer. Framework này cung cấp một hệ sinh thái phong phú với nhiều tính năng tích hợp như file-system routing, API routes, và hỗ trợ TypeScript mặc định. Next.js cũng có khả năng tùy biến cao thông qua hệ thống cấu hình linh hoạt, cho phép developers điều chỉnh framework để phù hợp với yêu cầu cụ thể của dự án.

### 2.1.2 Kiến trúc và nguyên lý hoạt động của Next.js

Next.js được xây dựng trên nền tảng của React và Node.js, sử dụng kiến trúc hybrid cho phép kết hợp linh hoạt giữa SSR và CSR. Framework này hoạt động dựa trên nguyên lý "convention over configuration", trong đó các quy ước về cấu trúc thư mục và đặt tên file được sử dụng để tự động tạo ra routing và các cấu hình khác. Kiến trúc này cho phép developers tập trung vào việc phát triển tính năng thay vì phải lo lắng về cấu hình cơ bản.

### 2.1.3 Các tính năng chính của Next.js

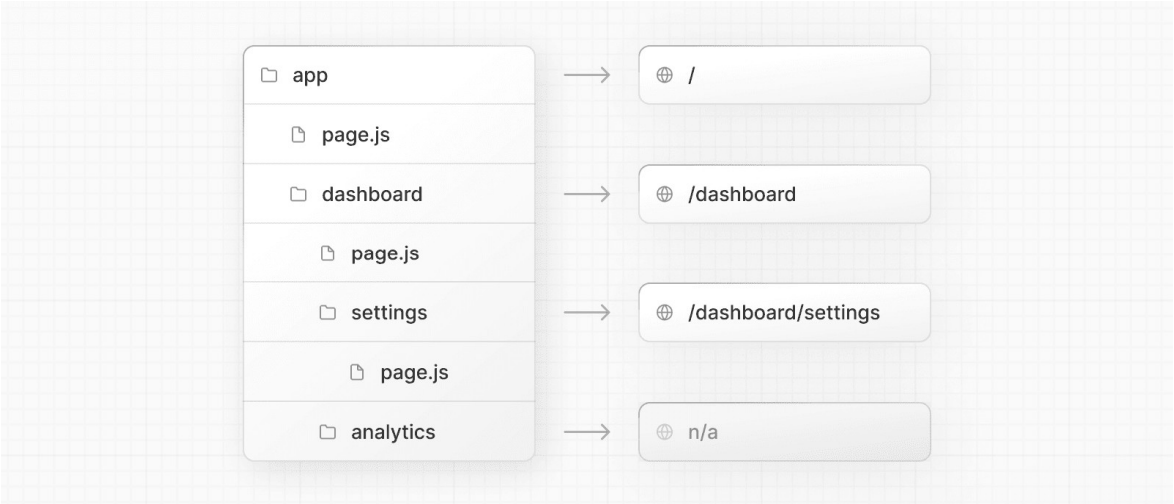
**Server-Side Rendering (SSR):** Next.js cung cấp khả năng render các trang web trên server trước khi gửi đến client. Điều này giúp cải thiện thời gian tải trang

đầu tiên và tối ưu hóa SEO. SSR đặc biệt hữu ích cho các trang web động có nội dung thường xuyên thay đổi.

**Static Site Generation (SSG):** Tính năng này cho phép tạo ra các trang HTML tĩnh tại thời điểm build. SSG giúp tăng hiệu suất và giảm tải cho server vì các trang đã được tạo sẵn và có thể được lưu cache.

**Client-Side Rendering (CSR):** Next.js vẫn hỗ trợ rendering phía client cho các tương tác động, cho phép ứng dụng hoạt động như một SPA sau lần tải đầu tiên.

**File-system Routing:** Next.js sử dụng hệ thống routing dựa trên cấu trúc thư mục, trong đó mỗi thư mục đại diện cho một router và ta có thể lòng các thư mục vào nhau. Ví dụ, nếu ta tạo một thư mục có tên là `dashboard` và thư mục `setting` được lồng bên trong. Thì Next.js sẽ tạo ra router `/dashboard` và `dashboard/settings`.




*Hình 2.1. File-system Routing.*

**Dynamic Routes:** Khi ta không biết chính xác tên của các segment và muốn tạo routes từ dữ liệu động, ta có thể sử dụng *Dynamic Routes* để tạo ra các router tại thời điểm request hoặc thời xây dựng. Ví dụ, một bài blog có route `app/blog/[slug]/page.js` trong đó `[slug]` là một *Dynamic Routes* cho các bài blog.

```
export default async function Page({
  params,
}: {
  params: Promise<{ slug: string }>
}) {
  const slug = (await params).slug
  return <div>My Post: {slug}</div>
}
```

**API Routes:** Framework cung cấp khả năng tạo API endpoints trực tiếp trong ứng dụng Next.js thông qua thư mục pages/api, cho phép xây dựng full-stack applications trong cùng một project.

### 2.1.4 Server Components và Client Components

**Server Components:** Đây là một tính năng đột phá của React và Next.js, cho phép các thành phần của trang web được render hoàn toàn phía máy chủ. Các thành phần này không được gửi đến phía người dùng, giúp giảm kích thước bundle Javascript. Server Components đặc biệt tối ưu tốt cho các phần của ứng dụng không yêu cầu tương tác của người dùng như các trang tĩnh, hoặc các thành phần chứa logic xử lý phức tạp.

Ưu điểm của Server Components là:

- Tăng tốc độ tải ở lần đầu tiên.
- Truy cập trực tiếp vào tài nguyên máy chủ như cơ sở dữ liệu.
- Bảo mật tốt hơn do logic nhạy cảm được giữ ở phía server.

**Client Components:** Ngược lại, Client Components vẫn được render và tương tác hoàn toàn ở phía người dùng. Các thành phần này sử dụng các trạng thái (state), các sự kiện, và các hooks như useEffect, useState. Chúng được đánh dấu bằng chỉ thị "use client" ở đầu file, cho phép Next.js nhận biết và xử lý phù hợp.

Ưu điểm của Client Components:

- Hỗ trợ đầy đủ các tính năng tương tác của React.

- Sử dụng các hooks và quản lý trạng thái.
- Phù hợp cho các component yêu cầu tương tác động như form, nút bấm,…

## 2.2. Abstract Syntax Tree (AST)

### 2.2.1 Khái niệm

Cây cú pháp trừu tượng (Abstract Syntax Tree) là một cấu trúc dữ liệu được sử dụng phổ biến trong ngành khoa học máy tính để biểu diễn cấu trúc của một chương trình hoặc đoạn mã nguồn dưới dạng dạng cây phân cấp. Mỗi nút (node) trong cây tương ứng với một thành phần cú pháp (syntax construct) trong mã nguồn. AST khác với cây cú pháp cụ thể (Concrete Syntax Tree) ở chỗ nó loại bỏ các chi tiết cú pháp không cần thiết như dấu ngoặc, dấu chấm phẩy, và chỉ tập trung vào nội dung và cấu trúc logic của chương trình [2].

### 2.2.2 Vai trò và ứng dụng của AST

AST được xem là bước trung gian quan trọng trong quá trình phân tích cú pháp mã nguồn, thường được sử dụng bởi các trình biên dịch (compiler) để thực hiện những tác vụ như kiểm tra ngữ nghĩa, phân tích sự phụ thuộc, tối ưu hóa chương trình, và chuyển đổi sang code máy hoặc bytecode.

AST đóng vai trò trung tâm trong các hệ thống phân tích mã nguồn tĩnh hiện đại, phát hiện mã độc, lỗ hổng bảo mật [3].

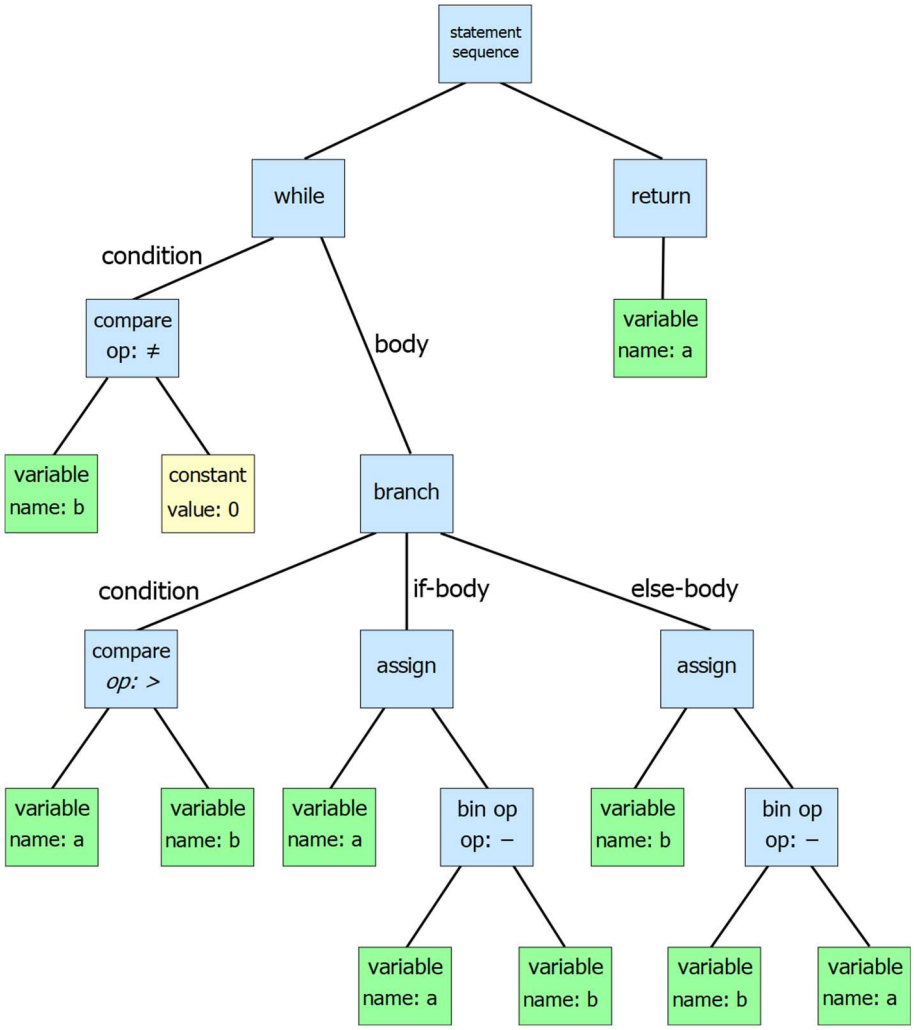
### 2.2.3 Cách xây dựng và phân tích AST

Việc xây dựng AST bắt đầu từ giai đoạn phân tích cú pháp, khi trình phân tích cú pháp chuyển đổi dãy token của chương trình thành một cây cấu trúc.

Ví dụ ta có đoạn code python sau:

```
while b ≠ 0:
    if a > b:
        a := a - b
    else:
        b := b - a
return a
```

Thì ta có cây cu pháp như sau:




*Hình 2.2. Ví dụ về một Abstract Syntax Tree.*

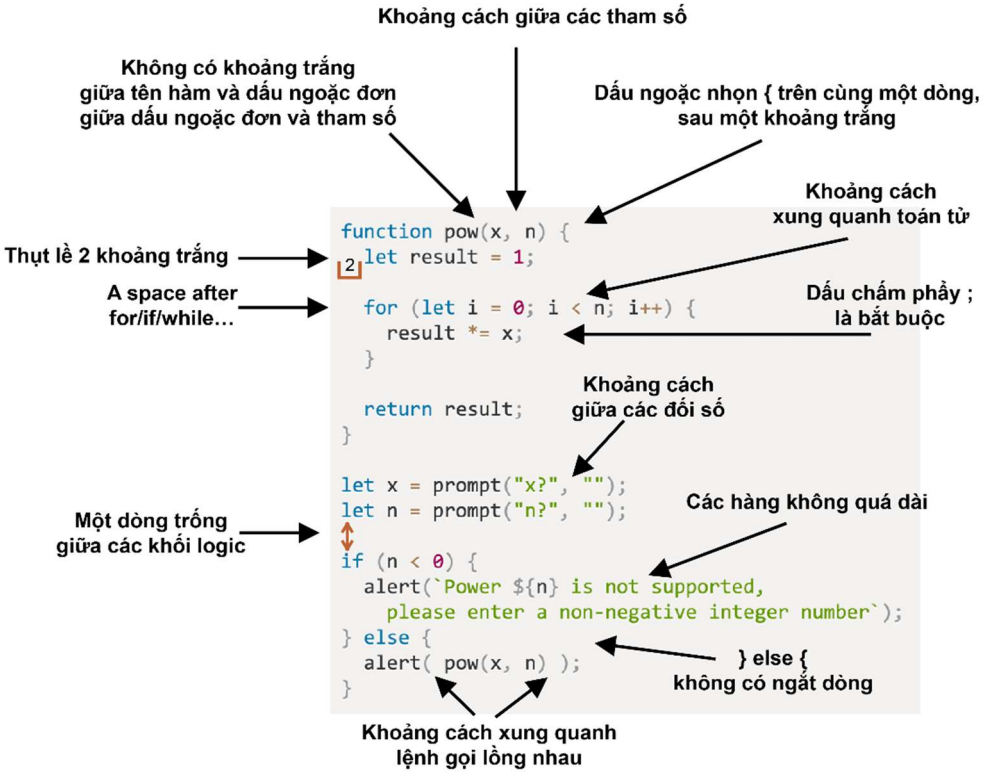
Có hai loại parser phổ biến: bottom-up và top-down, mỗi loại sẽ xây dựng AST theo hướng khác nhau.

Khi đã có AST, có thể sử dụng các thuật toán để duyệt cây, phân tích các node hoặc trích xuất thông tin về các hàm, biến, cấu trúc lệnh… Thông qua đó, có thể xác định các pattern điển hình của mã AI-generated, ví dụ như chuỗi lệnh liên tục, các hàm không có ý nghĩa rõ ràng, hoặc sử dụng các mẫu cú pháp lặp lại bất thường.

## 2.3. Code style

### 2.3.1 Khái niệm và vai trò

Phong cách viết code (code style) là tập hợp các nguyên tắc và quy tắc định dạng nhằm tạo ra mã nguồn dễ đọc, dễ hiểu và dễ bảo trì. Nó bao gồm các yếu tố như cách sử dụng khoảng trắng, thụt lề, quy tắc đặt tên biến và hàm, cách tổ chức comment, và các mẫu định dạng (pattern formatting). Mục tiêu của code style là đảm bảo tính nhất quán, giúp các lập trình viên khác dễ dàng hiểu và làm việc với code, đồng thời làm giảm thiểu lỗi phát sinh trong quá trình phát triển và bảo trì dự án.




*Hình 2.3. Ví dụ về phong cách viết code.*

### 2.3.2 Đặc điểm phong cách viết code của con người và AI

Con người thường viết code với những xu hướng cá nhân, dẫn đến không nhất quán trong cách sử dụng khoảng trắng, thụt lề hay cách đặt tên biến. Có thể dùng tên biến ngắn gọn hoặc chung chung, thậm chí có các "dấu ấn" cá nhân trong cú pháp.

Ở chiều ngược lại, mã nguồn do AI sinh ra thường có phong cách viết rất nhất quán nhờ vào việc tuân thủ nghiêm ngặt các quy ước. Code AI-generated thường sử dụng tên biến mang tính mô tả cao, rõ ràng, ít dùng tên biến ngắn hoặc vắng ý nghĩa. Việc thụt lề, spacing, và comment được tổ chức bài bản, bám sát quy tắc rõ ràng.

### 2.3.3 Phân tích các yếu tố code style bằng số liệu

Việc phân tích phong cách viết code có thể thực hiện qua việc đo các số liệu:

**Khoảng trắng và thụt lề:** mức độ sử dụng khoảng trắng, cách thụt lề thể hiện sự chuẩn mực khi viết code. Code do con người viết thường không đồng nhất, có sự chênh lệch nhỏ trong thụt lề hoặc sử khoảng trắng, như là khoảng trắng dấu phẩy, chấm phẩy hoặc sau các toán tử, trong khi code AI lại rất đều đặn và chuẩn xác.

**Quy tắt đặt tên:** việc đặt tên biến, hàm có thể đo bằng độ dài tên, mức độ mô tả (sử dụng từ khóa rõ nghĩa), kiểu đặt tên (camelCase, snake-case…). AI thường đặt tên biến theo quy chuẩn thống nhất, còn con người có thể rất đa dạng hoặc ngắn gọn không rõ nghĩa.

**Cách viết comment và cách tổ chức các comment**: cách viết comment, mức độ và vị trí comment trong code cũng là một chỉ số phân tích.

**Cách mẫu về định dạng code:** các mẫu định dạng như cách ngắt dòng, cách viết các biểu thức, cú pháp phức tạp được tổ chức như thế nào cũng giúp phân biệt giữa code của con người và AI.

## 2.4. Optimized Heuristic Classification, Heuristic Scoring và Binary Classifier

### 2.4.1 Khái niệm về heuristic classification và heuristic scoring

Heuristic classification là một phương pháp phân loại sử dụng các quy tắc hoặc quy chuẩn gần đúng (heuristics) để đánh giá và quyết định nhãn phân loại cho dữ liệu. Trong lĩnh vực phát hiện mã nguồn, heuristic classification dựa trên việc phân tích các đặc điểm, mẫu hình (patterns) và các tín hiệu biểu hiện trong đoạn code để phân biệt xem mã đó có khả năng do AI sinh ra hay không. Heuristic scoring là quá trình gán điểm số (score) dựa trên các tiêu chí heuristic để đánh giá mức độ phù hợp hoặc khả năng thuộc về một lớp nhất định [4].

Điểm số heuristic giúp tạo ra thang đo định lượng từ các đặc trưng định tính, như mức độ nhất quán của phong cách code, sự phù hợp với các pattern mã AI, hoặc các đặc điểm cấu trúc biểu diễn qua AST. Việc gán điểm này làm nền tảng để các bộ phân loại nhị phân hoạt động hiệu quả hơn, khi có thước đo để phân biệt rõ ràng giữa các lớp mã nguồn.

### 2.4.2 Vai trò của binary classifier trong hệ thống phân loại heuristic

Binary classifier trong hệ thống heuristic classification thường được sử dụng để phân loại nhị phân giữa hai nhãn: ví dụ như mã nguồn do con người viết hoặc do AI-generated code. Bộ phân loại nhị phân dựa trên các đặc điểm đã được chọn lọc và điểm số từ heuristic scoring để thực hiện quyết định cuối cùng.

Các mô hình binary classifier có thể là các thuật toán học máy như logistic regression, SVM, hoặc mạng neural nhỏ được huấn luyện trên tập dữ liệu đã gắn nhãn. Việc lựa chọn và huấn luyện binary classifier có ảnh hưởng lớn đến độ chính xác và khả năng tổng quát của hệ thống phát hiện mã AI-generated. Khi được tối ưu và kết hợp với heuristics hiệu quả, bộ phân loại này có thể đạt được hiệu suất cao trong việc phân biệt chính xác các đoạn mã từ hai nguồn khác nhau.

## 2.5. FastAPI Framework

### 2.5.1 Giới thiệu chung về FastAPI

FastAPI là một framework web hiện đại, nhanh chóng và hiệu quả, được thiết kế để xây dựng các API bằng ngôn ngữ Python (phiên bản 3.6 trở lên). FastAPI nổi bật nhờ sử dụng kiểu dữ liệu (type hints) tiêu chuẩn của Python kết hợp với thư viện Pydantic để thực hiện việc xác thực và kiểm tra dữ liệu tự động, giúp tránh lỗi ngay từ giai đoạn biên dịch thay vì khi chạy chương trình. Điều này giúp cải thiện đáng kể độ tin cậy và chất lượng của API được phát triển. Ngoài ra, FastAPI tự động tạo ra tài liệu API tương tác như Swagger UI và ReDoc, luôn đồng bộ với code nguồn, tạo điều kiện thuận lợi cho việc phát triển và sử dụng API [5].

### 2.5.2 Kiến trúc và thiết kế

FastAPI được xây dựng dựa trên Starlette – một framework web nhẹ và ASGI (Asynchronous Server Gateway Interface), cho phép hỗ trợ lập trình bất đồng bộ (asynchronous programming) một cách tự nhiên, tối ưu hiệu năng cho các ứng dụng I/O-bound. Kiến trúc của FastAPI thường áp dụng các design pattern hiện đại như dependency injection để quản lý các phần phụ thuộc như kết nối cơ sở dữ liệu, xác thực người dùng, hay gọi dịch vụ bên ngoài, giúp mã nguồn được tổ chức tốt hơn, dễ bảo trì và mở rộng.

Trong cấu trúc dự án chuẩn của FastAPI, các router được tổ chức riêng biệt thành các module, tách rời rõ ràng giữa phần xử lý routing, các schema dữ liệu dùng Pydantic, các thành phần thao tác cơ sở dữ liệu (CRUD), và logic nghiệp vụ. Điều này tạo ra một hệ thống phân lớp rõ ràng, thuận tiện cho việc quản lý và phát triển quy mô lớn [6] [7].

### 2.5.3 Các tính năng nổi bật

FastAPI mang lại nhiều tính năng nổi bật giúp tăng tốc độ phát triển và hiệu suất ứng dụng, bao gồm:

- Hỗ trợ chuẩn kiểu dữ liệu và xác thực tự động: Sử dụng Python type hints và Pydantic để kiểm soát và xác nhận dữ liệu đầu vào, đảm bảo tính đúng đắn của dữ liệu truyền qua API.
- Hiệu suất cao nhờ lập trình bất đồng bộ: Cho phép xử lý đồng thời nhiều yêu cầu API với hiệu năng vượt trội, phù hợp cho các ứng dụng số lượng lớn người dùng hoặc xử lý dữ liệu thời gian thực.
- Tài liệu API tương tác tự động: Swagger UI và ReDoc tự động được tạo ra, giúp lập trình viên dễ dàng thao tác, thử nghiệm và kiểm tra API mà không cần tạo tài liệu thủ công.
- Hệ thống Dependency Injection mạnh mẽ: Quản lý các đối tượng phục vụ nhiều route khác nhau một cách hiệu quả, giảm sự ràng buộc và tăng khả năng tái sử dụng mã.
- Bảo mật và xác thực tích hợp: Hỗ trợ các chuẩn phổ biến như OAuth2, JWT, API Key với khả năng tùy chỉnh cao.
- Xử lý phản hồi và lỗi nâng cao: Cho phép tùy chỉnh mã trạng thái HTTP, headers, cookie, và cung cấp khả năng tạo handler riêng cho các ngoại lệ.
- Hỗ trợ WebSockets và tác vụ nền: Giúp xây dựng ứng dụng thời gian thực và xử lý các tác vụ bất đồng bộ chạy phía sau mà không làm gián đoạn chính API.

# CHƯƠNG 3. THỰC HIỆN HOÁ NGHIÊN CỨU

## 3.1. Mô tả bài toán

Hệ thống được xây dựng nhằm giải quyết thách thức ngày càng lớn trong môi trường giáo dục lập trình: sự phổ biến của các công cụ sinh mã bằng AI như ChatGPT hay GitHub Copilot. Thực tế này đặt ra câu hỏi về tính minh bạch và việc đánh giá chính xác năng lực thực sự của sinh viên. Để hỗ trợ giải quyết vấn đề này, dự án tập trung phát triển một công cụ phân tích mã nguồn chuyên sâu, có khả năng làm nổi bật những đặc trưng khác biệt giữa mã do người viết và mã do AI tạo ra, đặc biệt với ngôn ngữ C/C++.

Hệ thống hoạt động bằng cách phân tích mã nguồn thông qua một quy trình đa tầng, trích xuất hơn nhiều đặc trưng khác nhau từ cấu trúc cây cú pháp trừu tượng (AST), phong cách lập trình, cho đến các chỉ số phức tạp và các mẫu thường gặp. Kết quả phân tích được tổng hợp thành một báo cáo chi tiết, bao gồm các điểm số, biểu đồ và diễn giải, giúp người dùng có cái nhìn toàn diện về bản chất của đoạn mã.

Hệ thống không nhằm mục tiêu khẳng định tuyệt đối một đoạn mã là do AI tạo ra hay không. Thay vào đó, nó đóng vai trò là một công cụ tham khảo, cung cấp thêm các căn cứ khách quan và dựa trên dữ liệu. Mục tiêu cuối cùng là trang bị cho giảng viên, cán bộ chấm thi hoặc quản trị viên một nguồn thông tin giá trị, giúp họ đưa ra những đánh giá công bằng và xác đáng hơn về bài làm của sinh viên trong bối cảnh công nghệ hiện đại.

## 3.2. Phân tích thiết kế hệ thống

### 3.2.1 Đặt tả về yêu cầu hệ thống

- **Yêu cầu chức năng**

Chức năng Phân tích mã nguồn: Hệ thống phải có khả năng tiếp nhận mã nguồn ngôn ngữ C/C++ thông qua trình soạn thảo văn bản hoặc tải tệp lên. Sau khi nhận được mã, hệ thống sẽ tiến hành quy trình trích xuất đặc trưng đa tầng để thu thập các chỉ số về cấu trúc, phong cách, độ phức tạp và các mẫu đặc trưng khác.

Chức năng Hiển thị kết quả phân tích: Kết quả phải được trình bày một cách trực quan và chi tiết trong một giao diện dashboard. Giao diện này cần hiển thị rõ ràng kết quả dự đoán, mức độ tin cậy, các biểu đồ minh họa sự phân bổ của các đặc trưng, và phần diễn giải chi tiết về các yếu tố chính dẫn đến kết luận đó.

Chức năng Nhập liệu và Tương tác: Cung cấp một trình soạn thảo mã (code editor) tích hợp sẵn với khả năng tô sáng cú pháp (syntax highlighting) cho C/C++, giúp người dùng dễ dàng dán và chỉnh sửa mã.

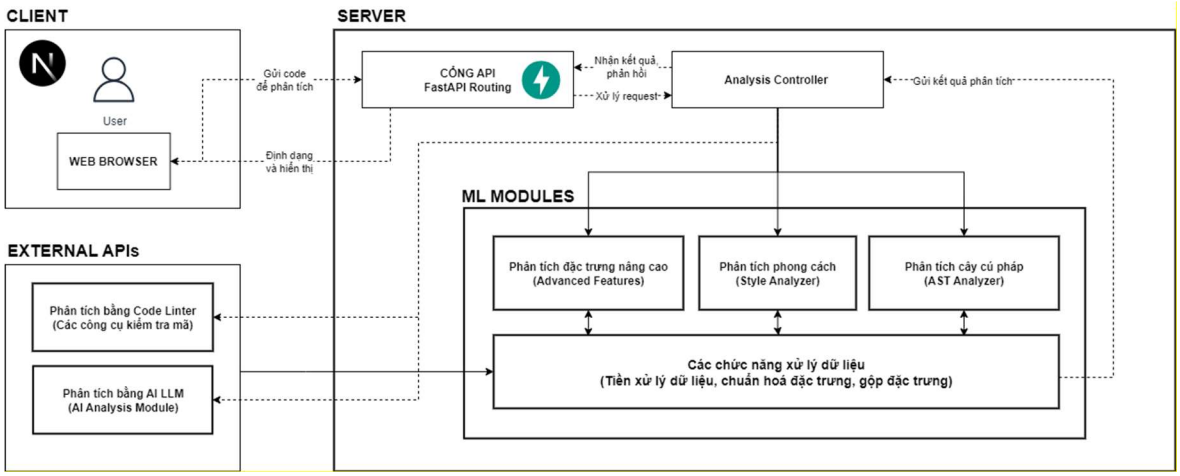
Chức năng Xuất báo cáo: Cho phép người dùng xuất toàn bộ kết quả phân tích chi tiết ra một tệp có cấu trúc (ví dụ: JSON, CSV), phục vụ cho mục đích lưu trữ, báo cáo hoặc nghiên cứu sâu hơn.

- **Yêu cầu phi chức năng**

Giao diện trực quan và dễ sử dụng: Giao diện người dùng cần được thiết kế sạch sẽ, chuyên nghiệp và dễ hiểu, đặc biệt là phần hiển thị kết quả. Các thông số kỹ thuật phức tạp phải được trình bày theo cách mà người dùng không chuyên về kỹ thuật (như giảng viên) vẫn có thể nắm bắt được.

Độ tin cậy và Tính nhất quán: Mặc dù không đưa ra kết luận tuyệt đối, các kết quả phân tích phải nhất quán và đáng tin cậy. Với cùng một đoạn mã đầu vào, hệ thống phải luôn trả về kết quả tương tự trong các lần phân tích khác nhau.

### 3.2.2 Kiến trúc hệ thống




*Ảnh 3.1. Kiến trúc hệ thống.*

Kiến trúc hệ thống bao gồm các thành phần và luồng dữ liệu như sau:

**CLIENT (Phía người dùng)**

User (Người dùng): Là đối tượng cuối cùng sử dụng hệ thống, bao gồm sinh viên, giảng viên hoặc các lập trình viên muốn kiểm tra mã nguồn để phát hiện xem code có được tạo bởi AI hay không. Người dùng tương tác với hệ thống thông qua giao diện web.

Frontend (Giao diện người dùng Next.js + Trình soạn thảo Monaco + Dashboard): Giao diện người dùng được xây dựng bằng Next.js với TypeScript, tích hợp trình soạn thảo mã Monaco có highlight systax để người dùng nhập mã nguồn cần phân tích. Dashboard hiển thị kết quả phân tích dưới dạng biểu đồ, số liệu và đánh giá chi tiết. Giao diện gửi yêu cầu phân tích đến SERVER và nhận kết quả để hiển thị cho người dùng.

**SERVER (Phía Máy Chủ)**

API Routing (Điều hướng API): Được xây dựng trên FastAPI, chịu trách nhiệm tiếp nhận các yêu cầu từ CLIENT. Hệ thống điều hướng này chuyển các yêu cầu đến Analysis Controller phù hợp và xử lý.

Analysis Controller (Bộ điều khiển phân tích): Bộ điều khiển trung tâm xử lý logic nghiệp vụ, nhận yêu cầu phân tích mã từ hệ thống điều hướng, điều phối việc gọi các module phân tích, tổng hợp kết quả từ các ML module và API bên ngoài, sau đó định dạng kết quả trả về cho giao diện dưới dạng JSON có cấu trúc.

**ML MODULES (Chứa các module phân tích)**

AST Analyzer (Bộ phân tích cây cú pháp): Module phân tích cấu trúc cây cú pháp của mã nguồn, trích xuất các đặc trưng, độ phức tạp, các mẫu của vòng lặp/điều kiện, và các số liệu liên quan đến cấu trúc mã để phát hiện các mẫu đặc trưng của mã được tạo bởi AI.

Style Analyzer (Bộ phân tích phong cách): Phân tích phong cách lập trình của con người như tính nhất quán trong thụt lề, khoảng cách, quy ước đặt tên, các mẫu về định dạng. Module này tìm kiếm các mẫu "không nhất quán" đặc trưng của mã viết bởi con người so với mà AI thường sẽ không mắc phải.

Advanced Features (Đặc trưng nâng cao): Trích xuất các đặc trưng phức tạp như độ phức tạp Halstead, chỉ số khả năng bảo trì, sự trùng lặp mã, các mẫu mã khuôn mẫu, các mẫu xử lý lỗi, và các số liệu khác để tạo ra bộ đặc trưng toàn diện (hơn 80 đặc trưng) cho việc phân loại.

Detection Models (Mô hình phát hiện): Tập hợp các mô hình học máy (heuristic, nâng cao) được huấn luyện để phân loại mã thành do AI tạo ra hoặc do con người viết dựa trên các đặc trưng từ các bộ phân tích trên. Các mô hình này cung cấp điểm tin cậy và lý do giải thích cho kết quả dự đoán.

**EXTERNAL APIs (API Bên Ngoài)**

Gemini AI API: API Google Gemini 2.0 Flash được sử dụng để thực hiện phân tích bổ sung với khả năng lập luận và xử lý ngôn ngữ tự nhiên. API này cung cấp phân tích và trả về định dạng MDX để hiện thị cho người dung.

Code Linter (Công cụ kiểm tra mã): Các công cụ kiểm tra mã bên ngoài để kiểm tra chất lượng, lỗi cú pháp. Kết quả kiểm tra được tích hợp vào quá trình phân tích để đánh giá tổng thể chất lượng.

# CHƯƠNG 4. TRIỂN KHAI VÀ KẾT QUẢ THỰC NGHIỆM

## 4.1. AST Analyzer Module

### 4.1.1 Phương pháp tiếp cận và thư viện sử dụng

AST Analyzer Module trong hệ thống sử dụng một phương pháp tiếp cận độc đáo để phân tích cấu trúc mã nguồn C/C++. Thay vì sử dụng các parser AST truyền thống phức tạp như Clang hoặc GCC, module này được thiết kế dựa trên kỹ thuật pattern matching sử dụng biểu thức chính quy (regular expressions) kết hợp với các heuristics thống kê. Lựa chọn này được đưa ra nhằm đảm bảo tính ổn định, hiệu suất cao và khả năng xử lý đa dạng các kiểu mã nguồn mà không cần phụ thuộc vào các thư viện parsing phức tạp.

Module sử dụng các thư viện Python chuẩn bao gồm `re` cho pattern matching, collections để xử lý dữ liệu thống kê, và `dataclasses` để định nghĩa cấu trúc dữ liệu. Đặc biệt, việc sử dụng regex patterns được tối ưu hóa đặc biệt cho ngôn ngữ C/C++, với các pattern phức tạp để nhận diện cấu trúc điều khiển, khai báo hàm, biến, và các đặc trưng về phong cách lập trình. Phương pháp này cho phép module hoạt động độc lập mà không cần cài đặt các compiler tool chains phức tạp.

### 4.1.2 Quy trình trích xuất đặc trưng

Quy trình của của AST Analyzer được thiết kế theo một pipeline có cấu trúc gồm nhiều giai đoạn xử lý tuần tự. Đầu tiên, module thực hiện giai đoạn tiền xử lý để làm sạch mã nguồn bằng cách loại bỏ các comment dạng `//` và `/* */`, normalize các dòng trống liên tiếp, và chuẩn bị mã nguồn cho việc phân tích. Giai đoạn này đảm bảo rằng việc pattern matching sẽ chính xác hơn và không bị nhiễu bởi các thành phần không cần thiết.

Sau giai đoạn tiền xử lý, module tiến hành trích xuất đặc trưng theo năm nhóm chính thông qua các hàm chuyên biệt. Giai đoạn đầu module phân tích cấu trúc cơ bản của mã, bao gồm ước lượng tổng số node thông qua việc đếm các statement, expression và declaration.

Tiếp theo, module phân tích các cấu trúc điều khiển như các câu lệnh điều kiện, vòng lặp for/while, và switch. Module không chỉ đếm số lượng mà còn phân tích độ sâu lồng nhau của các cấu trúc điều khiển thông qua việc theo dõi việc mở và đóng ngoặc nhọn.

### 4.1.3 Phân tích đặc trưng hàm và biến

Giai đoạn tiếp theo tập trung vào việc phân tích các đặc trưng liên quan đến hàm trong mã nguồn. Module sử dụng regex phức tạp để nhận diện khai báo hàm C/C++, bao gồm cả các kiểu dữ liệu truyền thống và hiện đại như auto. Đặc biệt, module có khả năng ước lượng độ dài của từng hàm bằng cách theo dõi việc mở và đóng ngoặc nhọn từ điểm khai báo hàm. Việc phát hiện hàm đệ quy được thực hiện thông qua heuristic đơn giản nhưng hiệu quả: kiểm tra xem tên hàm có xuất hiện nhiều lần trong body của chính nó hay không.

Module cũng thực hiện phân tích sâu về các cách đặt tên biến, hàm. Việc trích xuất tên biến được thực hiện bằng regex pattern có thể nhận diện nhiều kiểu khai báo biến C/C++, bao gồm cả pointer và khai báo hằng. Module phân loại các phương thức đặt tên như camelCase, snake_case, ký hiệu Hungary (Hungarian notation), và đặc biệt chú ý đến việc sử dụng các biến 1 ký tự trong biến vòng lặp. Điều này rất quan trọng vì AI thường có xu hướng sử dụng tên biến trực quan hơn so với con người.

### 4.1.4 Phân tích patterns và style consistency

Giai đoạn này tập trung vào việc phát hiện các mẫu code đặc trưng (code patterns) như:

- Số ma thuật (magic numbers) – các con số được viết trực tiếp trong code mà không có ý nghĩa rõ ràng hoặc không được định nghĩa bằng hằng số,
- Chuỗi ký tự trực tiếp (string literals) – các đoạn text được viết thẳng trong code,
- Câu lệnh include (include statements) – dùng để nhập thư viện hay file,
- Cách dùng macro (macro usage).

Những đặc trưng này cung cấp cái nhìn sâu hơn quan trọng về phong cách lập trình, vì code sinh bởi AI thường có cách dùng hằng số và include khác biệt so với code do con người viết.

Cuối cùng, thực hiện phân tích tính nhất quán về định dạng và phong cách viết code Module này kiểm tra:

- Sự nhất quán trong việc dùng dấu cách so với tab cho việc thụt lề
- Cách đặt ngoặc nhọn, vị trí của các cặp ngoặc nhọn
- Sự nhất quán trong việc đặt khoảng trắng quanh toán tử

Những đặc trưng này đặc biệt hữu ích để phân biệt AI code (thường rất nhất quán) với code của con người.

### 4.1.5 Chuẩn hóa

Một yếu tố quan trọng trong thiết kế của AST Analyzer là giai đoạn chuẩn hoá. Tất cả các đặc trưng đếm được chuẩn hóa theo tổng số dòng (Line of code hay LOC) hoặc theo các số liệu liên quan để đảm bảo khả năng so sánh giữa các mã nguồn số lượng dòng, kích thước khác nhau.

### 4.1.6 Các đặc trưng được trích xuất

*Bảng 4.1. Bảng các đặc trưng được trích xuất từ AST Analyzer Module.*

| Tên đặc trưng | Ý nghĩa và mục đích phân tích |
|---|---|
| **NHÓM CẤU TRÚC (Structure Features)** | |
| `total_nodes` | Tổng số node được ước lượng trong AST, phản ánh độ phức tạp cấu trúc của mã |
| `nodes_per_loc` | Mật độ node trên mỗi dòng code, cho biết mức độ đậm đặc của code |
| `max_depth` | Độ sâu tối đa của AST, thể hiện mức độ lồng nhau của các cấu trúc |
| `avg_depth` | Độ sâu trung bình của AST, đánh giá độ phức tạp tổng thể |

| | |
|---|---|
| branching_factor | Hệ số phân nhánh, tính theo tỷ lệ ngoặc nhọn mở so với tổng số dòng |
| **NHÓM LUỒNG ĐIỀU KHIỂN (Control Flow Features)** | |
| if_statements | Số lượng câu lệnh if trong code |
| if_statements_per_loc | Mật độ câu lệnh if trên mỗi dòng code |
| for_loops | Số lượng vòng lặp for |
| for_loops_per_loc | Mật độ vòng lặp for trên mỗi dòng code |
| while_loops | Số lượng vòng lặp while |
| while_loops_per_loc | Mật độ vòng lặp while trên mỗi dòng code |
| switch_statements | Số lượng câu lệnh switch |
| switch_statements_per_loc | Mật độ câu lệnh switch trên mỗi dòng code |
| nested_control_depth | Độ sâu lồng nhau tối đa của các cấu trúc điều khiển |
| **NHÓM HÀM (Function Features)** | |
| function_count | Tổng số hàm được định nghĩa trong code |
| functions_per_loc | Mật độ hàm trên mỗi dòng code |
| avg_function_length | Độ dài trung bình của các hàm (tính theo số dòng) |
| max_function_le006Egth | Độ dài của hàm dài nhất |
| recursive_functions | Số lượng hàm đệ quy được phát hiện |
| recursive_functions_ratio | Tỷ lệ hàm đệ quy so với tổng số hàm |
| **NHÓM BIẾN VÀ ĐẶT TÊN (Variable/Naming Features)** | |
| variable_count | Tổng số biến được khai báo |
| variables_per_loc | Mật độ khai báo biến trên mỗi dòng code |
| unique_variable_names | Số lượng tên biến duy nhất |
| variable_uniqueness_ratio | Tỷ lệ tên biến duy nhất so với tổng số biến |
| avg_variable_name_length | Độ dài trung bình của tên biến |

| | |
|---|---|
| camel_case_vars | Số lượng biến sử dụng quy tắt đặt tên camelCase |
| camel_case_ratio | Tỷ lệ biến camelCase so với tổng số biến |
| snake_case_vars | Số lượng biến sử dụng quy tắt đặt tên snake_case |
| snake_case_ratio | Tỷ lệ biến snake_case so với tổng số biến |
| single_char_vars | Số lượng biến có tên một ký tự (như i, j, k) |
| single_char_vars_ratio | Tỷ lệ biến một ký tự so với tổng số biến |
| hungarian_notation | Số lượng biến sử dụng Hungarian notation (strName, nCount) |
| hungarian_notation_ratio | Tỷ lệ biến Hungarian notation so với tổng số biến |
| **NHÓM PATTERNS MÃ (Code Pattern Features)** | |
| magic_numbers | Số lượng magic numbers (≥ 2 chữ số) |
| magic_numbers_per_loc | Mật độ magic numbers trên mỗi dòng code |
| string_literals | Số lượng chuỗi ký tự trong code |
| string_literals_per_loc | Mật độ chuỗi ký tựtrên mỗi dòng code |
| include_count | Số lượng câu lệnh #include |
| macro_usage | Số lượng macro definitions (#define) |
| **NHÓM PHONG CÁCH (Style Features)** | |
| indentation_consistency | Độ nhất quán trong việc sử dụng thục lề (0-1) |
| brace_style_consistency | Độ nhất quán trong style đặt ngoặc nhọn (0-1) |
| operator_spacing_consistency | Độ nhất quán trong khoảng trắng xung quanh toán tử (0-1) |

Các đặc trưng này được thiết kế để phát hiện những khác biệt tinh tế giữa mã do AI tạo ra và mã do con người viết:

- **Đặc trưng cấu trúc:** AI thường tạo ra code có cấu trúc đều đặn, độ sâu lồng nhau vừa phải

- **Đặc trưng luồng điều khiển:** AI có xu hướng sử dụng control structures một cách có pattern và tối ưu
- **Đặc trưng naming:** AI thường sử dụng tên biến descriptive, ít dùng single-char vars
- **Đặc trưng style:** AI code có có tính nhất quán cao, trong khi code người viết thường có sự không nhất quán một cách tự nhiên

Tất cả đặc trưng đều được chuẩn hoá theo LOC hoặc theo tổng đếm để đảm bảo khả năng so sánh giữa các mã nguồn có kích thước khác nhau.

## 4.2. Human Style Analyzer Module

### 4.2.1 Phương pháp tiếp cận và thư viện sử dụng

Human Style Analyzer Module được thiết kế với triết lý ngược lại so với AST Analyzer, tập trung vào việc phát hiện các đặc điểm "không hoàn hảo" và "không nhất quán" đặc trưng của mã nguồn do con người viết. Module này dựa trên nguyên lý rằng mã do AI tạo ra thường có xu hướng tuân thủ các quy tắc định dạng một cách nghiêm ngặt và nhất quán, trong khi mã do con người viết thường chứa đựng những sai sót nhỏ, tính không nhất quán trong phong cách, và các "dấu ấn" rất cá nhân trong cách tổ chức code.

Module sử dụng các thư viện Python chuẩn bao gồm `re` cho việc pattern matching nâng cao. Bộ đếm để phân tích thống kê các patterns, và so khớp cho các phép tính điểm số. Đặc biệt, module được xây dựng với hệ thống dataclass phân cấp gồm bốn lớp chính: SpacingIssues (Các lỗi về khoảng trắng), IndentationIssues (Các lỗi về thụt lề), NamingInconsistency (Các lỗi về quy tắt đặt tên), và FormattingIssues (Các lỗi về định dạng), mỗi lớp chuyên biệt hóa để phát hiện một nhóm vấn đề cụ thể. Kiến trúc này cho phép module phân tích chi tiết từng khía cạnh của phong cách lập trình một cách độc lập và sau đó tổng hợp thành điểm đánh giá tổng thể.

### 4.2.2 Quy trình phân tích chi tiết

Quy trình của Human Style Analyzer bắt đầu với việc phân chia mã nguồn thành các dòng riêng biệt và thực hiện phân tích song song trên bốn khía cạnh chính.

Giai đoạn đầu tiên tập trung vào việc phát hiện các lỗi về khoảng trắng mà con người thường mắc phải. Module kiểm tra một cách tỉ mỉ các vấn đề như thiếu khoảng trắng trước và sau các toán tử, thừa khoảng trắng không cần thiết, sai sót trong việc đặt khoảng trắng xung quanh dấu phẩy và chấm phẩy, cũng như các vấn đề về khoảng trắng phía sau (trailing spaces) và và dòng trống có chứa khoảng trắng.

Đặc biệt, module có khả năng phân biệt các ngữ cảnh khác nhau của spacing. Ví dụ, nó kiểm tra việc thiếu khoảng trắng trước các dấu ngoặc đơn mở trong các cấu trúc điều kiện như như `` `if(điều kiện)` `` thay vì `` `if␣(điều kiện)` ``, hoặc phát hiện việc đặt space không đúng cách khi gọi hàm như như `` `func( a, b )` `` thay vì `` `func(a, b)` ``. Những chi tiết này rất quan trọng vì AI code thường tuân thủ các quy tắt về khoảng trắng nghiêm ngặt, còn con người thì không.

### 4.2.3 Phân tích thụt lề và quy tắt đặt tên

Giai đoạn đầu module thực hiện phân tích sâu về các lỗi thụt lề, một trong những điểm khác biệt rõ rệt nhất giữa mã do AI và con người tạo ra. Module không chỉ phát hiện việc trộn lẫn tabs và khoảng trắng, mà còn phân tích tính nhất quán trong việc sử dụng kích thước thụt lề. Nó kiểm tra xem có sử dụng các kích thước thụt lề "bất thường" (không phải 2, 4, hoặc 8), phát hiện các dòng bị thụ vào quá mực hoặc hoặc không thụt vào một cách không hợp lý.

Module còn có khả năng phân tích patterns thụt lề thông qua việc theo dõi sự thay đổi mức độ thụt lề giữa các dòng liên tiếp. Nó tính toán mức độ thay đổi phổ biến nhất và đánh dấu những dòng không tuân theo pattern này. Điều này giúp phát hiện các trường hợp con người tự điều chỉnh việc thụt lùi theo cảm nhận cá nhân thay vì tuân theo quy tắc cứng nhắc.

Phần phân tích sự không nhất quán trong quy tắt đặt tên tập trung vào việc phát hiện sự không nhất quán trong conventions đặt tên. Module kiểm tra việc trộn lẫn camelCase và snake_case trong cùng một đoạn mã, phát hiện các patterns đặt tên kém như việc sử dụng tên biến chung chung (temp, var, foo), và đánh giá mức độ nhất quán trong các mẫu tên.

### 4.2.4 Các đặc trưng được trích xuất

*Bảng 4.2. Bảng các đặc trưng được trích xuất từ Human Style Analyzer Module.*

| Tên đặc trưng | Ý nghĩa và mục đích phân tích |
|---|---|
| **NHÓM VẤN ĐỀ KHOẢNG TRẮNG (Spacing Issues)** | |
| `missing_space_before_operator` | Số lần thiếu khoảng trắng trước toán tử (ví dụ: x+1 thay vì x + 1) |
| `missing_space_after_operator` | Số lần thiếu khoảng trắng sau toán tử (ví dụ: x+ 1 thay vì x + 1) |
| `extra_space_before_operator` | Số lần thừa khoảng trắng trước toán tử (ví dụ: x  + 1) |
| `extra_space_after_operator` | Số lần thừa khoảng trắng sau toán tử (ví dụ: x +  1) |
| `missing_space_after_comma` | Số lần thiếu khoảng trắng sau dấu phẩy (ví dụ: func(a,b)) |
| `missing_space_after_semicolon` | Số lần thiếu khoảng trắng sau dấu chấm phẩy trong vòng for |
| `extra_space_before_comma` | Số lần thừa khoảng trắng trước dấu phẩy (ví dụ: func(a , b)) |
| `extra_space_before_semicolon` | Số lần thừa khoảng trắng trước dấu chấm phẩy |
| `space_after_opening_paren` | Số lần có khoảng trắng sau ngoặc mở (ví dụ: func( a, b)) |
| `space_before_closing_paren` | Số lần có khoảng trắng trước ngoặc đóng (ví dụ: func(a, b )) |

| | |
|---|---|
| missing_space_before_opening_paren | Số lần thiếu khoảng trắng trước ngoặc mở trong control structures |
| trailing_spaces | Số dòng có khoảng trắng thừa ở cuối dòng |
| empty_lines_with_spaces | Số dòng trống nhưng có chứa khoảng trắng |
| spacing_issues_ratio | Tỷ lệ tổng các vấn đề spacing so với tổng số dòng code |
| **NHÓM VẤN ĐỀ THỤT LỀ (Indentation Issues)** | |
| mixed_tabs_spaces | Số dòng trộn lẫn tabs và spaces trong cùng một dòng |
| inconsistent_indentation | Số dòng có thụt lề không nhất quán với pattern chung |
| unusual_indent_size | Số dòng sử dụng kích thước thụt lề bất thường (không phải 2, 4, 8) |
| over_indented_lines | Số dòng bị thụt lề quá mức (thay đổi > 8 spaces) |
| under_indented_lines | Số dòng bị thụt lề không đúng mức khi giảm indentation |
| poor_continuation_alignment | Số dòng có alignment kém cho line continuation |
| indentation_issues_ratio | Tỷ lệ các vấn đề thụt lề so với tổng số dòng có indentation |
| **NHÓM BẤT NHẤT QUÁN ĐẶT TÊN (Naming Inconsistency)** | |

| | |
|---|---|
| mixed_camel_snake | Số lần trộn lẫn camelCase và snake_case trong cùng code |
| inconsistent_variable_naming | Mức độ không nhất quán trong patterns đặt tên biến |
| inconsistent_function_naming | Mức độ không nhất quán trong patterns đặt tên hàm |
| unclear_abbreviations | Số tên biến/hàm sử dụng abbreviations không rõ ràng |
| inconsistent_abbreviations | Mức độ không nhất quán trong việc sử dụng abbreviations |
| magic_numbers_without_constants | Số magic numbers chưa được chuyển thành constants |
| excessive_global_vars | Số lượng biến global được sử dụng quá mức |
| uncontrolled_global_usage | Mức độ sử dụng biến global không được kiểm soát |
| naming_inconsistency_ratio | Tỷ lệ các vấn đề naming so với tổng số identifiers |
| **NHÓM VẤN ĐỀ ĐỊNH DẠNG (Formatting Issues)** | |
| inconsistent_brace_style | Số lần thay đổi style đặt ngoặc nhọn (Allman vs K&R) |
| unnecessary_nested_braces | Số ngoặc nhọn lồng nhau không cần thiết |
| inappropriate_line_breaks | Số line breaks được đặt không phù hợp |

| | |
|---|---|
| too_long_lines | Số dòng quá dài (> 120 ký tự) |
| too_short_logical_blocks | Số logical blocks quá ngắn gọn |
| poor_comment_formatting | Số comments có formatting kém (thiếu space sau //) |
| inconsistent_comment_style | Mức độ không nhất quán trong style viết comment |
| formatting_issues_ratio | Tỷ lệ các vấn đề formatting so với tổng số dòng code |
| **ĐIỂM TỔNG THỂ** | |
| overall_human_score | Điểm tổng thể đánh giá mức độ "human-like" (0-1 scale) |

## 4.3. Advanced Features Module

### 4.3.1 Phương pháp tiếp cận và thư viện sử dụng

Advanced Features Module đóng vai trò như một module tích hợp tổng thể, kết hợp và mở rộng khả năng phân tích từ các module khác trong hệ thống. Khác với hai module trước tập trung vào các khía cạnh cụ thể, module này được thiết kế để trích xuất những đặc trưng phức tạp và đa chiều, đặc biệt là những patterns và số liệu mà chỉ có thể được phát hiện thông qua phân tích sâu về mối quan hệ giữa các thành phần khác nhau trong mã nguồn.

Module được thiết kế với khả năng tích hợp liền mạch với AST Analyzer và Human Style Analyzer, cho phép nó sử dụng kết quả từ hai module này để tạo ra một bức tranh toàn diện về đặc điểm của mã nguồn.

### 4.3.2 Quy trình tích hợp và phân tích tổng thể

Quy trình của Advanced Features Module bắt đầu với việc điều phối toàn bộ quá trình trích xuất đặc trưng. Đầu tiên, module thực hiện phân tích cơ bản để thu

thập các số liệu cơ bản như LOC, số lượng token, số hàm, và tính toán độ phức tạp chu trình thông qua việc đếm các điểm quyết định.

Tiếp theo, module tích hợp với AST Analyzer và Human Style Analyzer để thu thập đầy đủ thông tin từ các góc độ khác nhau. Ví dụ, thông tin về AST structure được sử dụng để cải thiện việc phân tích sự phức tạp, trong khi human style features được tích hợp để tạo ra đánh giá tổng thể về tính "tự nhiên" của mã nguồn.

Sau khi có đầy đủ dữ liệu cơ bản, module tiến hành bốn giai đoạn phân tích chuyên sâu song song. Giai đoạn tiếp theo sử dụng các thuật toán phân tích sequence để phát hiện dòng bị lặp, các mẫu trùng lập và tính điểm copy – dán bằng cách tìm kiếm chuỗi con dài nhất giữa các phần khác nhau của mã nguồn.

*Bảng 4.3. Bảng đặc trưng được trích xuất từ Advanced Features Module.*

| **Tên đặc trưng** | **Ý nghĩa và mục đích phân tích** |
|---|---|
| **NHÓM ĐẶC TRƯNG CƠ BẢN (Basic Features)** | |
| `loc` | Tổng số dòng code, số liệu cơ bản để đánh giá kích thước |
| `token_count` | Số lượng tokens (định danh, toán tử, phân cách) trong code |
| `cyclomatic_complexity` | Độ phức tạp cyclomatic dựa trên điểm quyết định và luồng |
| `functions` | Số lượng hàm được định nghĩa trong code |
| `comment_ratio` | Tỷ lệ dòng comment so với tổng số dòng |
| `blank_ratio` | Tỷ lệ dòng trống so với tổng số dòng |

| NHÓM REDUNDANCY (Code Redundancy Features) | |
|---|---|
| duplicate_lines | Số dòng code bị trùng lặp hoàn toàn |
| duplicate_line_ratio | Tỷ lệ dòng trùng lặp so với tổng số dòng |
| repeated_patterns | Số patterns (3-gram sequences) bị lặp lại |
| repeated_patterns_per_loc | Mật độ patterns lặp lại trên mỗi dòng code |
| copy_paste_score | Điểm copy-paste dựa trên longest common subsequences (0-1) |
| similar_function_ratio | Tỷ lệ functions có cấu trúc tương tự nhau |
| **NHÓM NAMING PATTERNS (Naming Pattern Features)** | |
| descriptive_var_ratio | Tỷ lệ biến có tên mang tính mô tả cao |
| generic_var_ratio | Tỷ lệ biến có tên chung chung (temp, var, i, j, etc.) |
| meaningful_names_score | Điểm đánh giá ý nghĩa của tên dựa trên tỉ lệ nguyên âm – phụ âm |
| verb_function_ratio | Tỷ lệ hàm có tên bắt đầu bằng động từ (get, set, calculate) |
| descriptive_function_ratio | Tỷ lệ hàm có tên mang tính mô tả (≥ 6 ký tự) |

| | |
|---|---|
| naming_consistency_score | Điểm nhất quán trong cách đặt tên (camelCase vs snake_case) |
| abbreviation_usage | Mức độ sử dụng từ viết tắng khi đặt tên |
| **NHÓM COMPLEXITY (Code Complexity Features)** | |
| halstead_complexity | Độ phức tạp Halstead dựa trên operators và operands |
| halstead_per_loc | Độ phức tạp Halstead chuẩn hóa theo LOC |
| cognitive_complexity | Độ phức tạp nhận thức dựa trên nesting và control flow |
| cognitive_per_loc | Độ phức tạp nhận thức chuẩn hóa theo LOC |
| maintainability_index | Chỉ số khả năng bảo trì (0-100 scale) |
| code_to_comment_ratio | Tỷ lệ code vs dòng comment |
| **NHÓM AI PATTERNS (AI Pattern Features)** | |
| template_usage_score | Điểm sử dụng code mẫu hoặc code tiêu chuẩn |
| boilerplate_ratio | Tỷ lệ mã mẫu (thư viện, cấu trúc đơn giản) |
| error_handling_score | Điểm xử lý lỗi có hệ thống (kiểm tra null, sử dụng return) |

| | |
|---|---|
| defensive_programming_score | Điểm code "phòng thủ" (khẳng định, xác thực - validate, kiểm chứng) |
| over_engineering_score | Điểm over-engineering (quá nhiều hàm cho công việc đơn giản) |
| **NHÓM AST FEATURES (từ AST Analyzer)** | |
| ast_* | Tất cả 25+ đặc trưng từ AST Analyzer với prefix "ast_" |
| **NHÓM HUMAN STYLE FEATURES (từ Human Style Analyzer)** | |
| spacing_* | Tất cả đặc trưng spacing issues với prefix phù hợp |
| indentation_* | Tất cả đặc trưng indentation issues với prefix phù hợp |
| naming_inconsistency_* | Tất cả đặc trưng naming inconsistency với prefix phù hợp |
| formatting_* | Tất cả đặc trưng formatting issues với prefix phù hợp |
| human_style_overall_score | Điểm tổng thể về human-style characteristics |
| overall_human_score | Điểm tổng thể đánh giá mức độ "human-like" (0-1 scale) |

## 4.4. AI Analysis Module

### 4.4.1 Phương pháp tiếp cận và tích hợp API

AI Analysis Module kết hợp sức mạnh của mô hình ngôn ngữ lớn với khả năng phân tích kỹ thuật chuyên sâu. Module này được thiết kế như một hệ thống chuyên gia có khả năng đánh giá mã nguồn từ góc độ của một lập trình viên có kinh nghiệm, không chỉ dựa trên các số liệu số học mà còn thông qua việc hiểu ngữ cảnh, các mẫu, và các nét đặc trưng tinh tế trong cách viết code.

Hệ thống tích hợp với Google Gemini 2.0 Flash API - một trong những mô hình ngôn ngữ tiên tiến nhất hiện tại, được đặc biệt tối ưu cho việc phân tích code. Việc lựa chọn Gemini 2.0 Flash không phải ngẫu nhiên mà dựa trên khả năng xử lý ngữ cảnh dài, hiểu biết sâu về lập trình.

### 4.4.2 Thiết kế prompt engineering và structured analysis

Trái tim của AI Analysis Module nằm ở phần prompt engineering được thiết kế cực kỳ cẩn thận và chi tiết. Prompt được cấu trúc thành nhiều sections rõ ràng, bắt đầu với việc định nghĩa vai trò của AI như một "AI Code Detector" chuyên nghiệp. Phần hướng dẫn cung cấp hướng dẫn cụ thể về cách phân tích, yêu cầu AI tập trung vào các mẫu, yếu tố về cấu trúc và phong cách viết code thay vì chỉ đưa ra kết luận đơn giản.

Đặc biệt quan trọng là phần tiêu chí phân tích, nơi định nghĩa tám khía cạnh chính cần được đánh giá: phong cách mã nguồn (consistency, naming conventions, formatting patterns), cấu trúc code (logic flow, organization, complexity patterns), patterns cú pháp (AI-typical vs human-typical syntax usage), comments và documentation (quality, style, documentation patterns), xử lý lỗi (error handling approaches), tuân thủ các tiêu chuẩn mã hoá, các chỉ số AI (các mẫu thường thấy trong mã do AI tạo ra) và các chỉ số của con người (sự không nhất quán tự nhiên và thói quen lập trình cá nhân).

Prompt được thiết kế để output ra định dạng MDX (Markdown eXtended) với cấu trúc được chuẩn hóa để có thể hiện thị một cách nhất quán mỗi lần chạy

### 4.4.3 Quy trình xử lý và xử lý dữ liệu trả về

Quy trình của AI Analysis bắt đầu với việc nhận request thông qua endpoint `/api/analysis/ai-analysis`, sau đó xác thực đầu vào theo các tiêu chuẩn tương tự như các module khác (chiều dài code, ngôn ngữ,…). Module tạo analysis ID và timestamp để đánh dấu, theo dõi, sau đó chuẩn bị payload gửi đến Gemini API với toàn bộ ngữ cảnh bao gồm ngôn ngữ lập trình, mã nguồn cần phân tích, và prompt hướng dẫn chi tiết.

Quá trình gọi API được gói trong hàm bất đồng bộ. Phản hồi từ API được xác thực kỹ lưỡng - kiểm tra xem có nội dung hay không, nội dung có hợp lệ hay không, và định đạng có đúng như hy vọng hay không.

*Bảng 4.4. Bảng nội dung phản hồi từ AI Analysis Module.*

| Tên thông tin | Ý nghĩa và mục đích phân tích |
|---|---|
| **THÔNG TIN CƠ BẢN** | |
| `success` | Trạng thái thành công của quá trình phân tích AI |
| `analysis_id` | ID duy nhất cho session phân tích AI |
| `timestamp` | Thời gian thực hiện phân tích |
| `analysis_type` | Loại phân tích để phân biệt với các module khác |
| `model` | Tên model AI được sử dụng (gemini-2.0-flash) |
| **NỘI DUNG MDX** | |

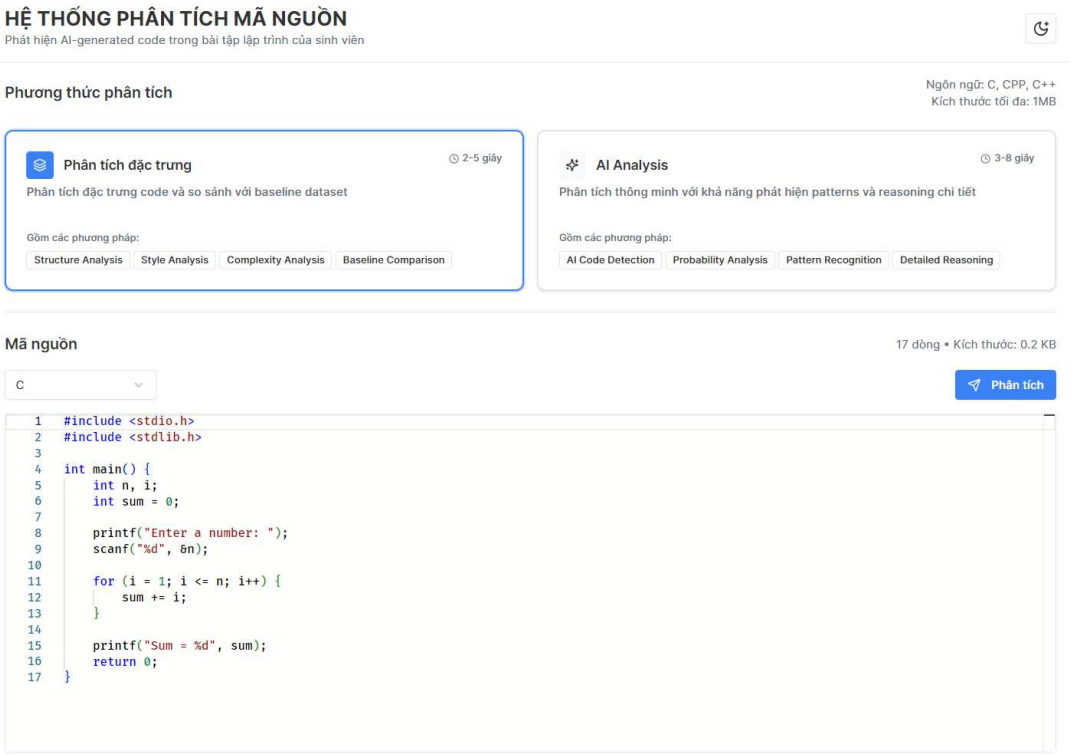
| | |
|---|---|
| `mdx_content` | Nội dung phân tích đầy đủ theo định dạng MDX |
| **CÁC SECTION TRONG MDX CONTENT** | |
| Kết quả dự đoán | Dự đoán chính (AI viết hay người viết), độ tin cậy tổng thể |
| Xác suất là AI viết | Phần trăm khả năng mã được tạo bởi AI (0-100%) |
| Xác suất là người viết | Phần trăm khả năng mã được viết bởi con người (0-100%) |
| Độ tin cậy | Mức độ tin cậy của kết quả dự đoán (0-100%) |
| **PHÂN TÍCH CHI TIẾT** | |
| Đánh giá phong cách mã nguồn | Phân tích coding style, naming conventions, formatting patterns |
| Đánh giá cấu trúc code | Phân tích logic flow, organization, complexity patterns |
| Đánh giá patterns cú pháp | So sánh AI-typical vs human-typical syntax usage |
| Đánh giá documentation | Phân tích quality và style của comments, documentation |
| **CHỈ SỐ QUAN TRỌNG** | |

| | |
|---|---|
| Patterns AI được phát hiện | Danh sách các patterns đặc trưng của AI-generated code |
| Patterns Human được phát hiện | Danh sách các patterns đặc trưng của human-written code |
| **LÝ DO VÀ GIẢI THÍCH** | |
| Lý do chính | 3 lý do chính dẫn đến kết luận với giải thích chi tiết |
| Giải thích độ tin cậy | Justification cho confidence level dựa trên evidence |
| Ghi chú bổ sung | Thông tin thêm hoặc limitations trong phân tích |

# CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 5.1. Kết quả đạt được

### 5.1.1 Giao diện chọn phương thức phân tích và trình soạn thảo mã nguồn

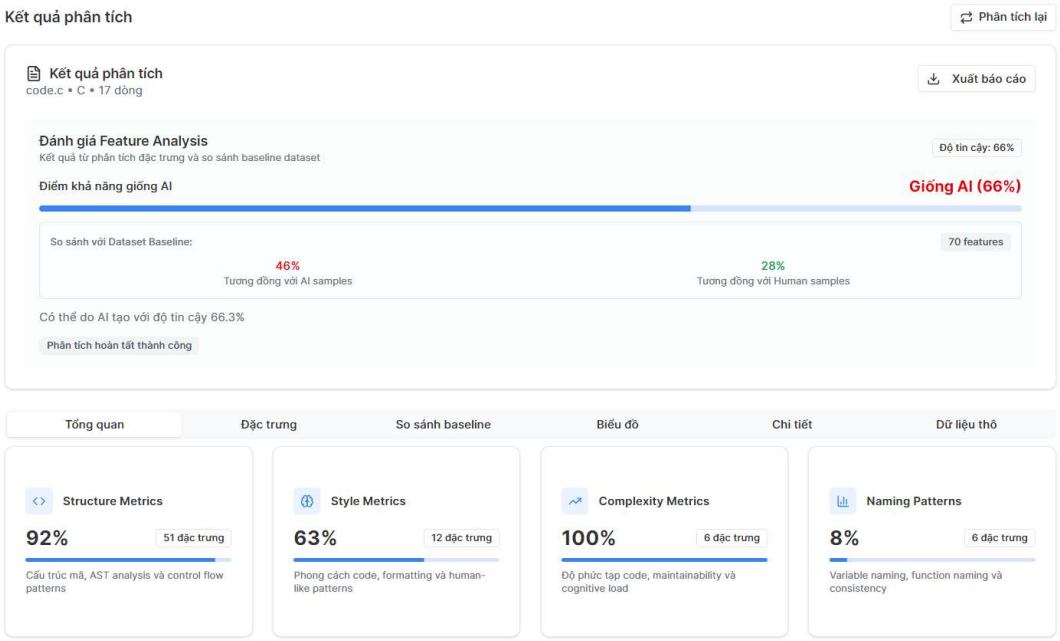



*Hình 5.1. Giao diện chọn phương thức phân tích và code editor nhập liệu.*

Giao diện chính của hệ thống được thiết kế với hai thành phần chính là bộ chọn phương thức phân tích và trình soạn thảo mã nguồn tích hợp. Phần bộ chọn phương thức được trình bày dưới dạng radio group với hai tùy chọn chính: “Phân tích toàn diện” sử dụng tất cả các module phân tích, và "AI Analysis" sử dụng mô hình Gemini.

Trình soạn thảo mã nguồn sử dụng Monaco Editor - cùng engine với Visual Studio Code, cung cấp trải nghiệm chỉnh sửa code chuyên nghiệp với syntax highlighting cho C/C++, đánh số dòng, code folding, và tự động hoàn thành code. Giao diện này tạo nền tảng thuận tiện cho việc nhập và chỉnh sửa mã nguồn cần phân tích.

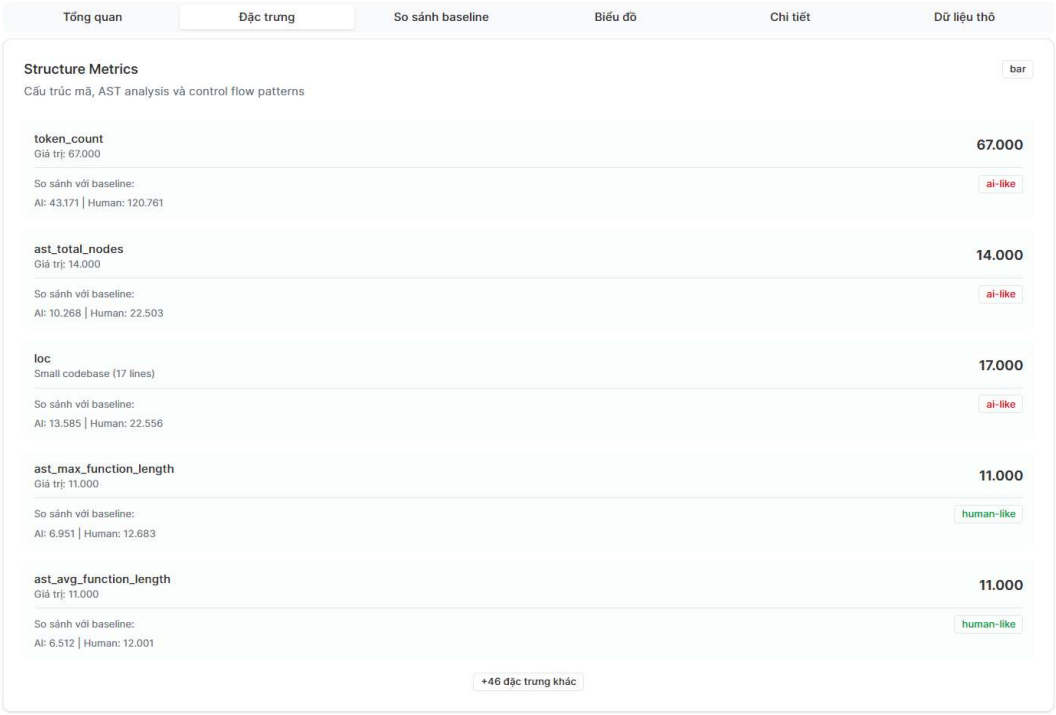
### 5.1.2 Dashboard kết quả phân tích tổng quan




*Hình 5.2. Giao diện kết quả phân tích tổng quan.*

Dashboard kết quả phân tích tổng quan trình bày thông tin một cách có cấu trúc và trực quan. Tab "Tổng quan" hiển thị các số liệu chính bao gồm điểm tổng thể về khả năng giống AI (biểu thị dưới dạng phần trăm và thanh tiến trình), độ tin cậy của kết quả phân tích, và tóm tắt đánh giá bằng ngôn ngữ tự nhiên.

### 5.1.3 Phân tích so sánh các đặc trưng cấu trúc mã nguồn

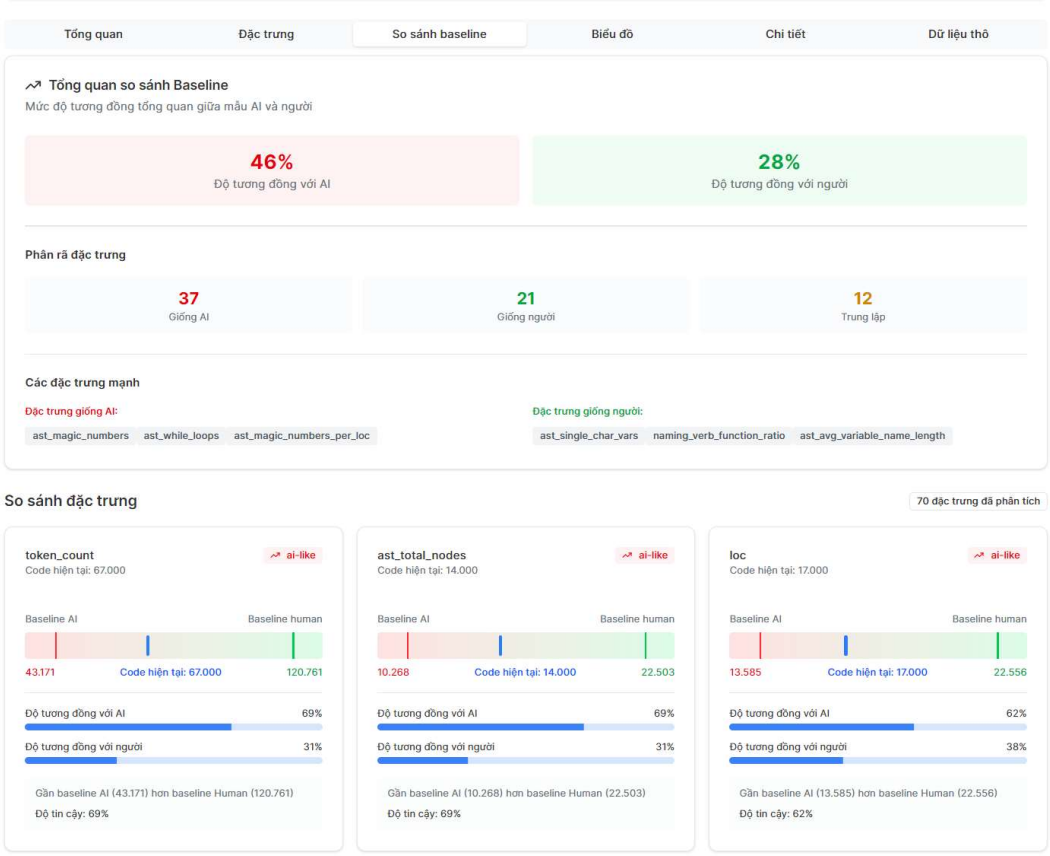



*Hình 5.3. Giao diện kết quả phân tích so sánh các đặc trưng về cấu trúc code.*

Tab "Đặc trưng" cung cấp view chi tiết về các nhóm đặc trưng theo từng nhóm: Structure Metrics, Style Metrics, Complexity Metrics, và Naming Patterns.

Mỗi đặc trưng được hiện thị với kết quả so sánh, hiển thị nhận xét(ai-like, human-like, hoặc neutral) cùng với các giá trị so sánh tương ứng cho AI và con người.

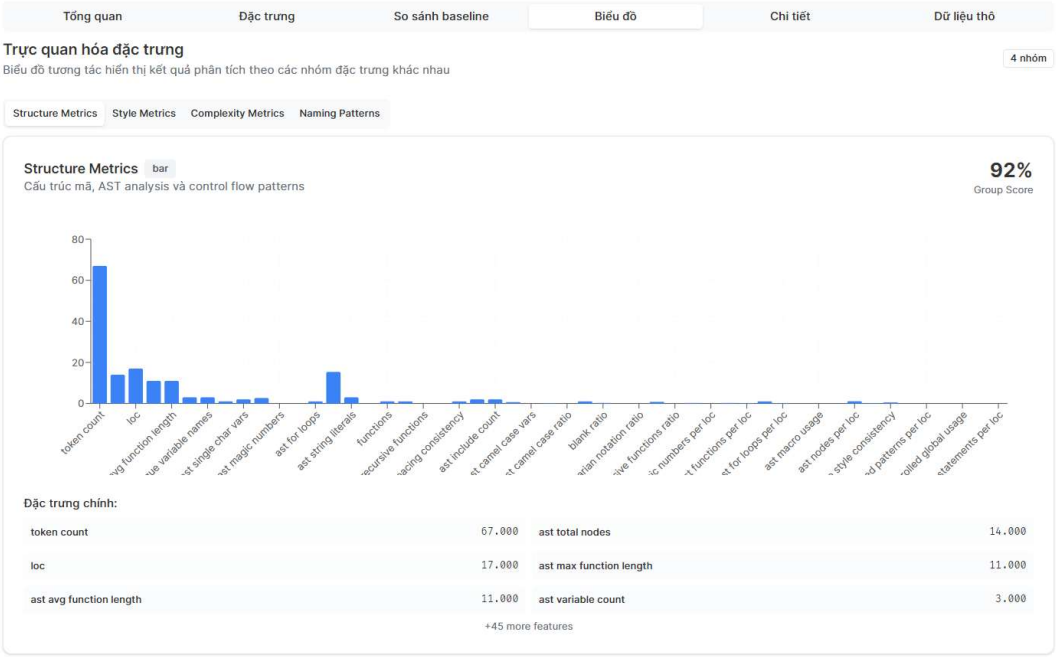
### 5.1.4 So sánh đặc trưng với baseline dataset



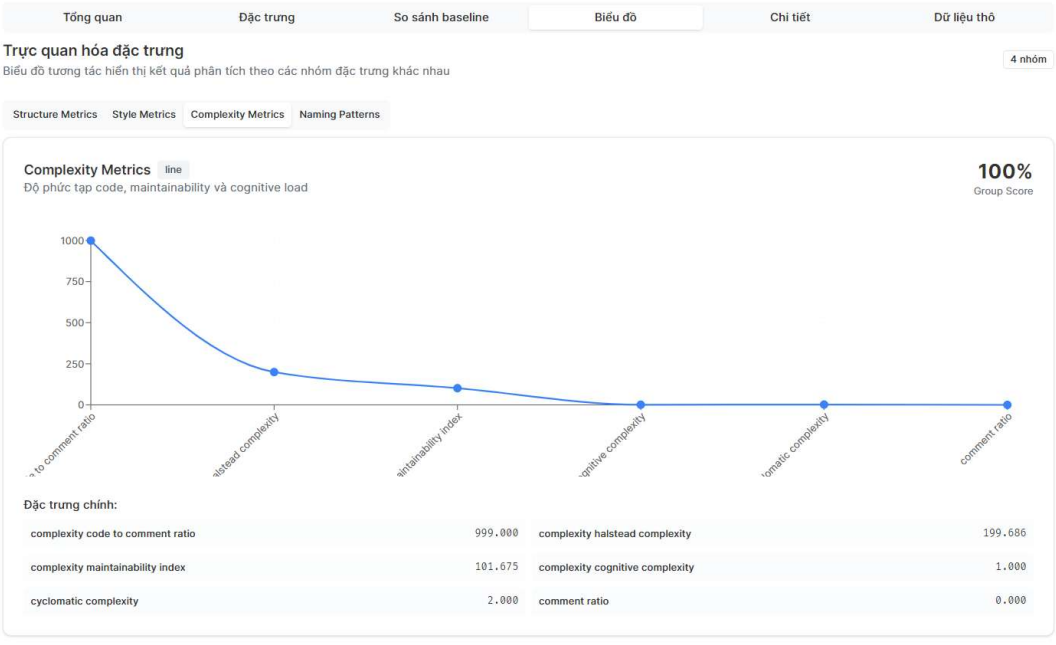

*Hình 5.4. Giao diện kết quả phân tích so sánh các đặc trưng với baseline.*

Tab "So sánh baseline" cung cấp một cái nhìn toàn diện về việc so sánh các đặc trưng của mã nguồn với tập dữ liệu cơ sở được xây dựng từ code mẫu của AI và code mẫu của người. Giao diện hiển thị so sánh với thống kê tổng quan: tổng số đặc trưng được so sánh, số lượng đặc trưng giống AI, giống người và trung lập.

### 5.1.5 Trực quan hoa và biểu đồ phân tích đặc trưng




*Hình 5.5. Giao diện biểu đồ kết quả phân tích các nhóm đặc trưng 1.*

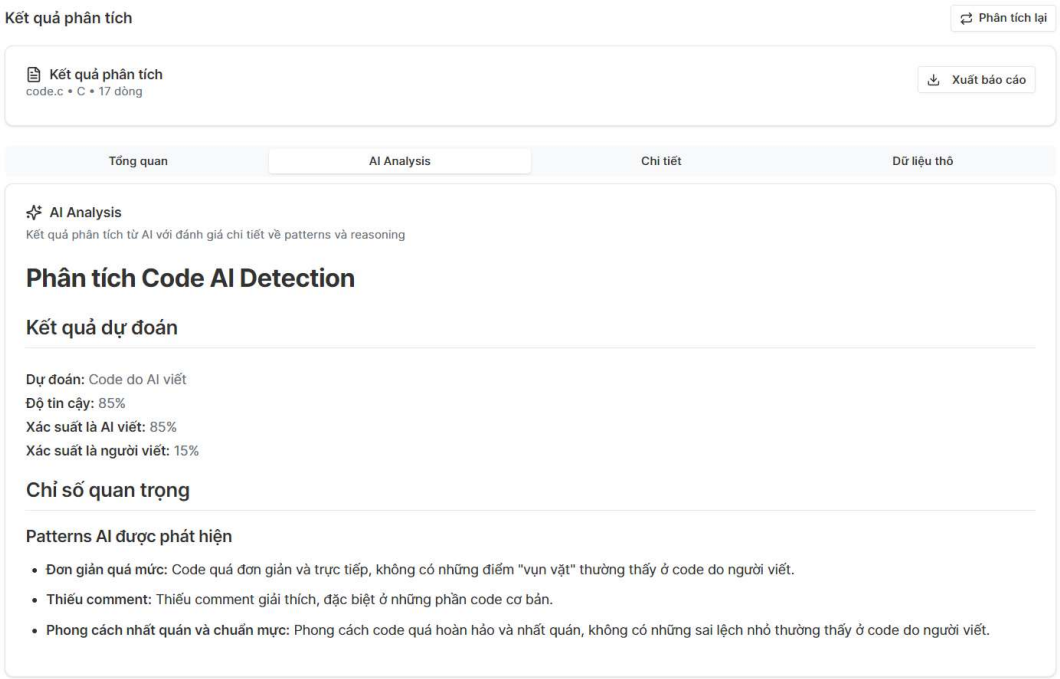



*Hình 5.6. Giao diện biểu đồ kết quả phân tích các nhóm đặc trưng 2.*

Trong tab “Biểu đồ”, hệ thống hiển thị dữ liệu phân tích dưới nhiều dạng đồ thị khác nhau để người dùng dễ quan sát.

- Biểu đồ cột dùng để so sánh các đặc trưng về cấu trúc code.
- Biểu đồ radar giúp hiển thị nhiều khía cạnh các đặc trưng cùng lúc và dễ nhận ra điểm chưa nhất quán.
- Biểu đồ đường thể hiện sự thay đổi về độ phức tạp.
- Box plot cho thấy cách phân bố các chỉ số trong việc phát hiện code AI.

### 5.1.6 Kết quả phân tích thông minh với AI




*Hình 5.7. Giao diện kết quả phân tích bằng AI.*

Trong tab "AI Analysis", hệ thống tích hợp mô hình Google Gemini 2.0 Flash để đưa ra phân tích thông minh kèm theo giải thích bằng ngôn ngữ tự nhiên. Kết quả phân tích được hiển thị rõ ràng nhờ MDX renderer, bao gồm:

- Kết quả dự đoán cùng mức độ tin cậy.
- Đánh giá chi tiết về phong cách lập trình, cấu trúc code, cách dùng cú pháp và chất lượng ghi chú (documentation).
- Nhận diện các đặc điểm thường thấy trong code AI và code của con người, kèm lý do giải thích cho từng kết luận.

Ngoài ra:

- Key indicators nêu bật các yếu tố quan trọng nhất ảnh hưởng đến kết quả phân loại.
- Confidence explanation giúp người dùng hiểu rõ vì sao hệ thống đưa ra mức độ tin cậy đó.
- Additional notes cung cấp thông tin bổ sung và nêu các giới hạn có thể có trong phân tích.

MDX renderer hỗ trợ trình bày phong phú với định dạng văn bản, tô màu cú pháp code và các yếu tố tương tác. Nhờ vậy, những phân tích AI phức tạp được trình bày thành nội dung dễ tiếp cận và dễ hiểu cho người dùng cuối.

## 5.2. Kết luận

Dự án "Xây dựng cơ chế phát hiện AI-generated code trong bài tập lập trình sinh viên" đã thành công trong việc đạt được các mục tiêu ban đầu thông qua việc phát triển một hệ thống phân tích mã nguồn toàn diện và đa chiều. Hệ thống gồm bốn mô-đun phân tích chuyên biệt:

- AST Analyzer: trích xuất hơn 25 đặc trưng về cấu trúc mã.
- Human Style Analyzer: phát hiện hơn 39 mẫu đặc trưng cho sự thiếu nhất quán thường gặp trong phong cách lập trình của con người.
- Advanced Features Module: tích hợp 32 chỉ số nâng cao để phân tích sâu.
- AI Analysis Module: tận dụng các mô hình ngôn ngữ lớn tiên tiến để đưa ra phân tích và giải thích thông minh.

Kiến trúc hệ thống được xây dựng theo mô hình microservices với backend FastAPI và frontend Next.js, chứng minh được tính khả thi, khả năng mở rộng và dễ bảo trì.

## 5.3. Hướng phát triển

**Mở rộng ngôn ngữ lập trình:** Hệ thống sẽ hỗ trợ thêm nhiều ngôn ngữ ngoài C/C++, như Python, Java, JavaScript…, thông qua framework phân tích đa ngôn ngữ. Điều này giúp dễ dàng mở rộng, nâng cao khả năng nhận diện code AI ở nhiều ngữ cảnh khác nhau.

**Cải thiện độ chính xác và hiệu năng:** Ứng dụng các kỹ thuật machine learning tiên tiến, tối ưu thuật toán và xử lý song song để tăng tốc phân tích, đảm bảo độ tin cậy và khả năng hoạt động thời gian thực.

**Tích hợp với hệ thống giáo dục:** Phát triển API để kết nối với LMS (Moodle, Canvas…), hỗ trợ phân tích trực tiếp trên IDE, nền tảng học lập trình online, và cung cấp dashboard thống kê cho nhà trường.

**Nghiên cứu và phát triển nâng cao:** Tiếp tục nghiên cứu kỹ thuật phát hiện AI tinh vi, phát triển AI có khả năng giải thích, phân tích xu hướng tiến hóa của mô hình sinh code, và hợp tác với các tổ chức học thuật để đóng góp cho cộng đồng nghiên cứu.

# TÀI LIỆU THAM KHẢO


[1] "Github," 2025. [Online]. Available: https://github.com/vercel/next.js.

[2] "Wikipedia," 9 August 2025. [Online].
Available: https://en.wikipedia.org/wiki/Abstract_syntax_tree.

[3] "Viblo," 2021. [Online]. Available: https://viblo.asia/p/phan-tich-source-code-voi-shiftleft-ng-sast-yMnKM8nN57P.

[4] "Wikipedia," 2025. [Online].
Available: https://en.wikipedia.org/wiki/Heuristic_(computer_science).

[5] "Geeksforgeeks," 23 July 2025. [Online].
Available: https://www.geeksforgeeks.org/python/fastapi-architecture/.

[6] "Uptrace," 17 October 2024. [Online].
Available: https://uptrace.dev/blog/python-fastapi.

[7] "Fastapi - Tiangolo," 2025. [Online].
Available: https://fastapi.tiangolo.com/tutorial/bigger-applications/.