Chương I: Mở đầu

1.1. Bối cảnh và tính cấp thiết của nhiệm vụ điền vào giữa (FIM)

*** Tầm quan trọng của mô hình Ngôn ngữ Lớn trong Lập trình:**

Ngành công nghiệp phần mềm đang chứng kiến một cuộc cách mạng nhờ sự xuất hiện của các Code LLMs (Mô hình Ngôn ngữ Lớn chuyên biệt cho mã nguồn), bao gồm các kiến trúc nổi bật như StarCoder, Code Llama, và các sản phẩm thương mại như GitHub Copilot.

Các mô hình này được huấn luyện trên khối lượng dữ liệu mã nguồn khổng lồ, giúp chúng không chỉ hiểu được ngữ pháp ngôn ngữ tự nhiên mà còn nắm vững cú pháp, ngữ nghĩa, và cấu trúc logic của nhiều ngôn ngữ lập trình khác nhau (Python, Java, C++, JavaScript, v.v.).

*** Khả năng sinh mã và Nhiệm vụ Điền vào giữa (FIM):**

Một trong những ứng dụng quan trọng nhất của Code LLMs là sinh mã tự động. Trong đó, nhiệm vụ Điền vào giữa (Fill-in-the-Middle - FIM) đóng vai trò then chốt.

FIM là khả năng mô hình tạo ra một đoạn mã để lấp đầy một khoảng trống (mask) nằm ở giữa một đoạn mã nguồn có sẵn. (Ví dụ: hoàn thành một hàm, thêm tham số vào một lời gọi hàm, hoặc chèn một khối logic giữa hai dòng code.)

Tính cấp thiết: Khả năng FIM chính xác và hiệu quả là thước đo trực tiếp cho mức độ hỗ trợ mà một Code LLM có thể cung cấp cho lập trình viên trong môi trường phát triển tích hợp (IDE), giúp giảm thời gian viết code lặp đi lặp lại và giảm thiểu lỗi cú pháp ngay từ đầu, từ đó nâng cao năng suất tổng thể.

1.2. Bộ dữ liệu SAFIM

SAFIM (Syntax-Aware Fill-in-the-Middle) là một bộ tiêu chuẩn được phát triển nhằm mục tiêu kiểm tra nghiêm ngặt khả năng duy trì tính đúng đắn của cú pháp trong quá trình FIM, đồng thời sử dụng dữ liệu tương đối mới để giảm thiểu nguy cơ rò rỉ dữ liệu (Data Leakage). Trong báo cáo này, dữ liệu từ bộ tiêu chuẩn SAFIM gốc đã được chọn làm nguồn dữ liệu cơ sở nhờ vào chất lượng và sự tập trung vào các tình huống FIM phức tạp về mặt cú pháp.

1.3. Mục tiêu của báo cáo

Báo cáo được thực hiện với mục đích giới thiệu và đánh giá Bộ Dữ liệu Tái cấu trúc (Derived Dataset) được phát triển hoàn toàn từ dữ liệu gốc của SAFIM, đồng thời, được thực hiện thông qua:

1. Trình bày một cấu trúc dữ liệu mới được tối ưu hóa cho mục đích thực nghiệm
2. Đánh giá sự khác biệt giữa bộ dữ liệu mới và bộ dữ liệu cũ

Chương II: Bộ dữ liệu tái cấu trúc

2.1. Tổng quan về bộ dữ liệu gốc SAFIM

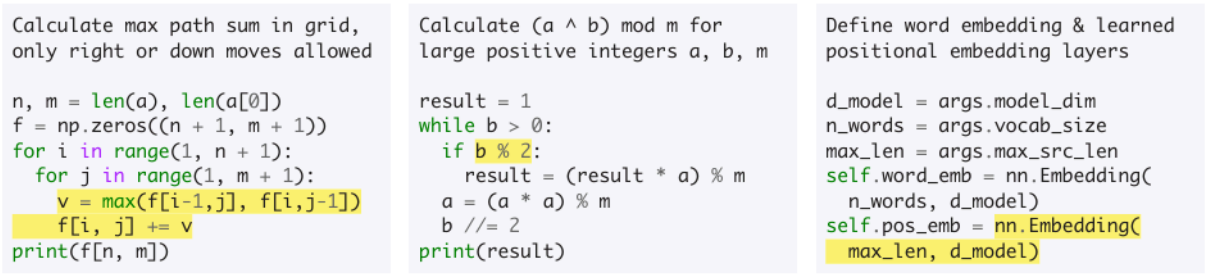
SAFIM (Syntax-Aware Fill-In-the-Middle) là một bộ tiêu chuẩn đánh giá (benchmark) hiện đại, được thiết kế để đo lường khả năng hoàn thiện mã nguồn dựa trên sự hiểu biết về cấu trúc cú pháp. SAFIM sử dụng cây cú pháp trừu tượng (Abstract Syntax Tree - AST) để xác định các vị trí "đục lỗ" có ý nghĩa về mặt logic và giải thuật.

Tầm quan trọng của SAFIM trong đánh giá FIM Code LLMs:

- Tính khách quan và thực tế: SAFIM được thu thập từ các kho lưu trữ mã nguồn mở trên GitHub và các nền tảng lập trình thi đấu (Codeforces) sau tháng 4/2022. Điều này giúp hạn chế tối đa hiện tượng "nhiễm dữ liệu" (data contamination), đảm bảo các mô hình hiện nay chưa từng "học thuộc" các đáp án trong quá trình huấn luyện.
- Đánh giá dựa trên cú pháp (Syntax-Aware): Bằng cách tập trung vào các thành phần quan trọng như khối thuật toán, luồng điều khiển và lời gọi API, SAFIM buộc mô hình phải thực sự hiểu cấu trúc chương trình thay vì chỉ dự đoán dựa trên sự lặp lại của các chuỗi ký tự.
- Đa ngôn ngữ và Đa tác vụ: Với quy mô hơn 17.000 ví dụ trải dài trên nhiều ngôn ngữ như Python, Java, C++, C#, SAFIM cung cấp một cái nhìn toàn diện về khả năng thích nghi của mô hình trong các môi trường lập trình khác nhau.
- Tiêu chuẩn thực thi (Execution-based): Kết quả của mô hình được đánh giá thông qua các bộ kiểm thử (Unit Tests), đảm bảo rằng mã nguồn được điền vào không chỉ đúng về mặt cú pháp mà phải thực sự chạy đúng về mặt logic.

2.2. Cấu trúc tác vụ của SAFIM

Bộ dữ liệu SAFIM được thiết kế để kiểm tra khả năng hoàn thành mã của Mô hình Ngôn ngữ Lớn (LLM). Cấu trúc này được phân chia thành ba loại tác vụ chính, mỗi loại tập trung vào một khía cạnh quan trọng của lập trình:



Hình 1.1: Các tác vụ chính(nguồn: Safim dataset)

2.2.1. Hoàn thành Khối Thuật toán (Algorithmic Block Completion)

*** Mô tả tác vụ**

Tác vụ Algorithmic Block Completion yêu cầu mô hình sinh ra một khối mã thuật toán hoàn chỉnh, thường gồm nhiều dòng lệnh, để giải quyết một phần quan trọng của bài toán.

Ở tác vụ này, mô hình cần:

- Hiểu mô tả bài toán
- Suy luận ý tưởng thuật toán
- Sinh mã đúng cú pháp
- Đảm bảo đoạn mã sinh ra phù hợp ngữ nghĩa với ngữ cảnh xung quanh

*** Giải thích ví dụ**

Ví dụ minh họa là bài toán tìm đường đi có tổng lớn nhất trong một lưới, chỉ được đi sang phải hoặc xuống dưới. Đoạn mã bị khuyết là phần cập nhật quy hoạch động:

```
Calculate max path sum in grid,
only right or down moves allowed

n, m = len(a), len(a[0])
f = np.zeros((n + 1, m + 1))
for i in range(1, n + 1):
  for j in range(1, m + 1):
    v = max(f[i-1,j], f[i,j-1])
    f[i, j] += v
print(f[n, m])
```

Algorithmic Block Completion

Để sinh đúng đoạn mã này, mô hình phải:

- Nhận ra đây là bài toán quy hoạch động
- Hiểu ý nghĩa của các trạng thái f[i-1, j] và f[i, j-1]
- Suy luận rằng cần chọn giá trị lớn nhất từ hai hướng
- Cập nhật trạng thái hiện tại dựa trên kết quả đó

Tác vụ này đánh giá khả năng tư duy thuật toán và khả năng sinh ra đoạn mã nhiều dòng có logic đúng.

2.2.2. Hoàn thành Biểu thức Luồng Điều khiển (Control-Flow Completion)

*** Mô tả tác vụ**

Tác vụ Control-Flow Completion tập trung vào việc sinh ra các cấu trúc điều khiển luồng, chẳng hạn như:

- Vòng lặp (while, for)
- Câu lệnh điều kiện (if, else)

- Thân của vòng lặp hoặc nhánh điều kiện

Mặc dù đoạn mã cần sinh thường ngắn hơn so với khối thuật toán, nhưng độ nhạy rất cao: chỉ một lỗi nhỏ cũng có thể làm chương trình sai hoặc rơi vào vòng lặp vô hạn.

*** Giải thích ví dụ**

Ví dụ trong hình là thuật toán lũy thừa nhanh (binary exponentiation):

```
Calculate (a ^ b) mod m for
large positive integers a, b, m

result = 1
while b > 0:
  if b % 2:
    result = (result * a) % m
  a = (a * a) % m
  b //= 2
print(result)
```

Control-Flow Completion

Ở đây, đoạn mã bị khuyết kiểm soát:

- Cách cập nhật biến result
- Cách giảm biến lặp b
- Cách bình phương a

Để hoàn thành đúng, mô hình phải:

- Nhận diện thuật toán lũy thừa nhanh
- Hiểu cách điều khiển vòng lặp
- Đảm bảo thứ tự thực thi chính xác
- Tránh lỗi logic làm sai kết quả

2.2.3. Hoàn thành Lời gọi hàm API (API Function Call Completion)

*** Mô tả tác vụ**

Tác vụ API Function Call Completion đánh giá khả năng của mô hình trong việc sử dụng đúng các hàm thư viện (API), thay vì tự triển khai thuật toán.

Ở tác vụ này, mô hình cần:

- Hiểu thư viện đang sử dụng
- Biết tên hàm, chữ ký hàm
- Truyền đúng tham số
- Tuân thủ phong cách mã nguồn

*** Giải thích ví dụ trong hình**

Ví dụ trong hình là việc khai báo positional embedding trong mô hình học sâu:

```
Define word embedding & learned
positional embedding layers

d_model = args.model_dim
n_words = args.vocab_size
max_len = args.max_src_len
self.word_emb = nn.Embedding(
  n_words, d_model)
self.pos_emb = nn.Embedding(
  max_len, d_model)
```

API Function Call Completion

Để sinh được dòng mã này, mô hình phải:

- Hiểu vai trò của embedding vị trí
- Biết API nn.Embedding của PyTorch
- Nhận biết ý nghĩa của các tham số max_len, d_model
- Phù hợp với ngữ cảnh lớp và phong cách lập trình

2.3. Phân bố định lượng của bộ dữ liệu tái cấu trúc

Phân bố định lượng của bộ dữ liệu tái cấu trúc được xây dựng dựa trên phân bố của bộ dữ liệu gốc. Dữ liệu được chia thành hai tập Train và Test theo tỷ lệ 5:1 nhằm đảm bảo tính ổn định trong quá trình huấn luyện và đánh giá mô hình.

Train/Test 5:1	Block	Control	API	Print
Train	4870	4870	259	40
Test	976	976	51	8

	java	cpp	c#	python
API Train	47	44	17	151
API Test	9	8	4	30
Control Train	1900	1900	445	625
Control Test	381	381	89	125
Block Train	1880	1880	440	670
Block Test	376	376	89	135
Print Train	20	20		
Print Test	4	4		

Bộ dữ liệu gồm ba lớp chính là Block, Control và API. Trong đó, hai lớp Block và Control có số lượng mẫu lớn và được phân bố cân bằng giữa hai tập Train và Test, với lần lượt 4.870 và 975 mẫu cho mỗi lớp. Lớp API có quy mô nhỏ hơn, gồm 260 mẫu cho tập Train và 50 mẫu

cho tập Test. Các mẫu cho tác vụ Print được lấy từ các mẫu dữ liệu của tác vụ Block chưa được dùng tới.

Ngoài ra, dữ liệu được phân tích theo các ngôn ngữ lập trình Java, C++, C# và Python. Phân bố mẫu giữa các ngôn ngữ tương đối đồng đều đối với các lớp Block và Control, trong khi lớp API có tỷ lệ mẫu Python cao hơn. Cách phân chia này giúp đảm bảo tính đa dạng dữ liệu và hạn chế thiên lệch trong quá trình huấn luyện mô hình.

2.4. Quy trình tái cấu trúc dữ liệu

Mục tiêu của bước này là xây dựng Bộ Dữ liệu Tái cấu trúc từ bộ dữ liệu SAFIM gốc, phục vụ cho việc huấn luyện và đánh giá mô hình sinh mã. Quá trình tái cấu trúc tập trung vào:

- Cân bằng dữ liệu theo ngôn ngữ lập trình
- Phân tách rõ ràng tập huấn luyện và kiểm tra
- Đảm bảo dữ liệu sạch, đầy đủ và có thể tái lập

2.4.1. Crawl và nạp dữ liệu gốc

Bộ dữ liệu SAFIM gốc được crawl trực tiếp từ Hugging Face thông qua thư viện datasets. Do SAFIM không cung cấp sẵn tập huấn luyện và kiểm tra, toàn bộ dữ liệu ban đầu được lấy từ split test của từng tác vụ.

Quá trình nạp dữ liệu được thực hiện riêng cho ba tác vụ:

- Algorithmic Block Completion
- Control-Flow Completion
- API Function Call Completion

2.4.2. Phân nhóm theo ngôn ngữ lập trình

Sau khi crawl, dữ liệu của mỗi tác vụ được phân nhóm theo ngôn ngữ lập trình gồm:

- Java
- C++
- C#
- Python

Việc phân nhóm theo ngôn ngữ giúp:

- Kiểm soát số lượng mẫu trên mỗi ngôn ngữ
- Tránh mất cân bằng dữ liệu
- Phục vụ phân tích chi tiết theo từng ngôn ngữ

2.4.3. Cân bằng và phân tách Train/Test

Dữ liệu sau khi phân nhóm được chia thành tập huấn luyện (Train) và tập kiểm tra (Test) theo số lượng cố định cho từng ngôn ngữ, thay vì chia ngẫu nhiên toàn bộ.

Nguyên tắc phân tách:

- Số lượng mẫu train/test được xác định trước cho từng tác vụ và từng ngôn ngữ
- Mỗi mẫu chỉ xuất hiện duy nhất trong một tập (train hoặc test)
- Không có trùng lặp task_id giữa hai tập

Để đảm bảo khả năng tái lập, quá trình xáo trộn dữ liệu trước khi chia sử dụng: random.seed(42), đảm bảo rằng mỗi lần chạy lại quy trình, tập dữ liệu thu được là giống hệt nhau.

2.4.4. Lưu trữ dữ liệu

Sau khi phân tách, dữ liệu được lưu trữ dưới dạng: JSONL (JSON Lines)

- Mỗi dòng là một mẫu dữ liệu
- Phù hợp cho huấn luyện mô hình

2.4.5. Kiểm tra và làm sạch dữ liệu

Sau khi xây dựng xong bộ dữ liệu, nhiều bước kiểm tra được thực hiện nhằm đảm bảo chất lượng:

***Kiểm tra thiếu trường dữ liệu**

- Kiểm tra từng mẫu dữ liệu xem có đầy đủ các trường thuộc tính
- Nếu bản ghi nào không thỏa điều kiện sẽ được xóa bỏ khỏi tập dữ liệu

***Kiểm tra giá trị rỗng**

- Kiểm tra từng trường cho từng bản ghi đảm bảo không trường nào rỗng
- Nếu bản ghi nào chứa trường rỗng, phải xóa bỏ khỏi tập dữ liệu

***Kiểm tra trùng lặp Train/Test**

- So sánh dựa vào task_id giữa train và test xem có bộ dữ liệu nào giống nhau
- Đảm bảo cho việc huấn luyện dữ liệu sẽ ra kết quả tốt nhất

***Phân tích trùng nội dung**

- Kiểm tra nội dung các file tránh trường hợp dữ liệu trùng lặp

2.5. Phân tích và Điểm khác biệt trong Phân bố

2.5.1. Phân tích theo Ngôn ngữ

Xét theo ngôn ngữ lập trình, Java và C++ chiếm tỷ trọng lớn nhất trong hai lớp Control và Block, cho thấy bộ dữ liệu tập trung mạnh vào các cấu trúc điều khiển và khối lệnh phổ biến trong các ngôn ngữ lập trình hệ thống và hướng đối tượng. Python có số lượng mẫu thấp hơn trong hai lớp này nhưng lại chiếm ưu thế rõ rệt trong lớp API, phản ánh đặc trưng sử dụng API phong phú và linh hoạt của Python. Trong khi đó, C# có tỷ lệ thấp nhất ở hầu hết các lớp, thể hiện mức độ đại diện hạn chế so với các ngôn ngữ còn lại.

2.5.2. Phân tích theo tác vụ

Về mặt tác vụ, hai lớp Block và Control chiếm phần lớn số lượng mẫu, trong khi lớp API có quy mô nhỏ hơn. Riêng tác vụ Print, đây là một tác vụ có độ phức tạp thấp và ít thông tin ngữ nghĩa, chủ yếu phản ánh cú pháp hơn là cấu trúc hay logic chương trình. Vì vậy, trong quá trình tái cấu trúc dữ liệu, tác vụ Print không được ưu tiên mở rộng, đặc biệt đối với các ngôn ngữ có số lượng mẫu hạn chế như Python và C#.

2.5.3. Tỷ lệ phân tách

Bộ dữ liệu được phân tách theo tỷ lệ Train/Test = 5:1 và được áp dụng nhất quán cho tất cả các lớp và ngôn ngữ còn lại. Tỷ lệ này giúp đảm bảo tập huấn luyện đủ lớn để mô hình học được các đặc trưng quan trọng, đồng thời vẫn giữ được tập kiểm thử có tính đại diện để đánh giá hiệu năng một cách khách quan.

Chương III: Kết luận

Trong báo cáo này, chúng tôi đã giới thiệu bộ dữ liệu tái cấu trúc SAFIM, được tổ chức lại một cách có hệ thống trên bốn ngôn ngữ lập trình. Bộ dữ liệu cung cấp phân tích định lượng chi tiết về phân bố Train/Test cho ba tác vụ cốt lõi gồm API, Control và Block. Việc tái cấu trúc dữ liệu giúp đảm bảo tính nhất quán trong phân bố, đồng thời phản ánh sát đặc điểm thực tế của dữ liệu mã nguồn, tạo nền tảng đáng tin cậy cho các nghiên cứu và đánh giá mô hình tiếp theo.