

# CODE REPRESENTATION LEARNING AT SCALE

*Dejiao Zhang & Wasi Ahmad | Ming Tan & Hantian Ding*

*Ramesh Nallapati & Dan Roth & Xiaofei Ma & Bing Xiang*

*AWS AI Labs*

## ABSTRACT

Các nghiên cứu gần đây đã chỉ ra rằng các mô hình ngôn ngữ mã trên quy mô lớn chứng minh hiệu suất tăng đáng kể đối với các downstream task, ví dụ như là sinh mã. Tuy nhiên, hầu hết các công trình hiện liên quan đến biểu diễn mã ở quy mô hàng trăm triệu tham số sử dụng tập dữ liệu tiền huấn luyện rất hạn chế. Trong báo cáo này, chúng tôi nâng cấp việc học biểu diễn mã bằng một lượng lớn dữ liệu mã thông qua gồm hai giai đoạn. Giai đoạn một, chúng tôi huấn luyện các bộ mã thông qua sự kết hợp tận dụng của randomness in masking language modeling (RiMLM) và khía cạnh cấu trúc của ngôn ngữ lập trình. Sau đó, chúng tôi nâng cao các biểu diễn thông qua contrastive learning (học tương phản) với hard negative và hard positive được xây dựng theo học không giám sát. Chúng tôi xây dựng một mô hình bộ mã hóa có sẵn hoạt động tốt hơn các mô hình hiện có trong nhiều downstream task với large margins. Để hiểu rõ các yếu tố góp phần vào sự thành công trong việc biểu diễn mã, chúng tôi sẽ loại bỏ các chi tiết và chia sẻ sự phát hiện của chúng tôi về (i) sơ đồ khử nhiễu mức mã thông báo tùy chỉnh và hiệu quả cho mã nguồn; (ii) tầm quan trọng của hard negative và hard positive; (iii) bimodal contrastive learning trong việc thúc đẩy hiệu suất tìm kiếm ngữ nghĩa đa ngôn ngữ; (iv) cách các pretraining quyết định các hiệu suất của downstream task theo kích thước mô hình.

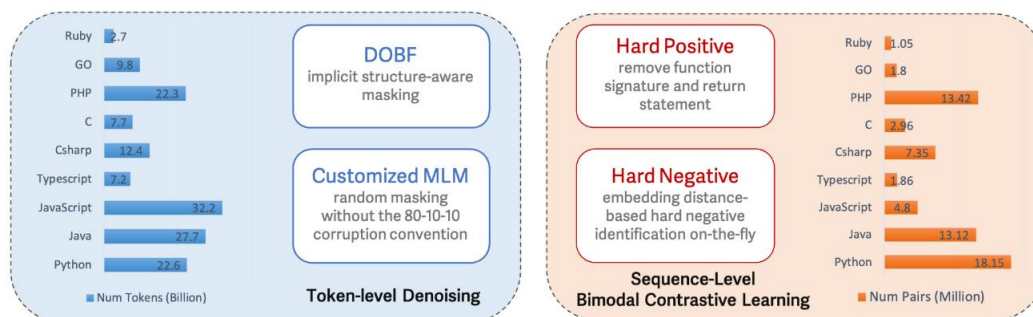
## 1. GIỚI THIỆU

Large language models (LLMs) được huấn luyện trước trên một lượng lớn mã nguồn đã thay đổi đáng kể lĩnh vực tạo mã (Chen et al., 2021; Chowdhery et al., 2022; Li et al., 2023, và những người khác). Ví dụ, việc phát hành gần đây một tập dữ liệu 6TB (Kocetkov et al., 2022) bao gồm mã nguồn có giấy phép cho phép đã đóng vai trò then chốt trong việc thúc đẩy sự phát triển của các mô hình ngôn ngữ mã trong thời điểm hiện tại. Tuy nhiên, những tập dữ liệu lớn này chưa được tận dụng hết để phát triển các Programming Language (PL) embedding models nhiều mục đích. Đến nay, hầu hết các mô hình PL (Feng et al., 2020a; Guo et al., 2021; 2022, và những người khác) có không quá 125 triệu tham số và chủ yếu được huấn luyện trên một vài triệu ví dụ huấn luyện, ví dụ như CodeSearchNet (Husain et al., 2019).

Mặc dù không thể phủ nhận tầm quan trọng của dữ liệu quy mô lớn, nhưng cần phải thừa nhận vai trò quan trọng của các mục tiêu huấn luyện trước. Cách tiếp cận phổ biến hiện nay để huấn luyện trước một bộ mã hóa Transformer hai chiều nhằm học các biểu diễn là thông qua việc tối ưu hóa mục tiêu Mask Language Model (MLM), như được đề xuất bởi Devlin và cộng sự (2019b). Quy trình masking trong mục tiêu MLM tiêu chuẩn tuân theo quy tắc 80-10-10. Tuy nhiên, chúng tôi nhận thấy rằng quy trình che như vậy dẫn đến sự phát triển của các mô hình nhúng mã không tối ưu. Vì các đoạn mã chứa cả các câu lệnh natural language (NL) (tức là docstring, comment) và mã thuần túy, việc thay thế các token bị che bằng một token ngẫu nhiên

theo quy tắc 80-10-10 có thể dẫn đến việc thay thế một token NL bằng một token PL, và ngược lại (xem thống kê ở Phụ lục A.3). Chúng tôi suy đoán rằng sự đồng xuất hiện của PL và NL cùng với bản chất cú pháp của mã nguồn làm cho nó dễ bị phá vỡ cả ngữ nghĩa và cấu trúc của mã bị che, dẫn đến việc học ngôn ngữ mô hình không tối ưu.

Trong khi MLM đã được huấn luyện mang lại các biểu diễn ngữ cảnh của token, hầu hết các downstream sau này chủ yếu hoạt động ở mức độ chuỗi. Khi mục tiêu là tăng cường khả năng phân biệt biểu diễn để ứng dụng ngay lập tức trong các nhiệm vụ ở mức độ chuỗi, Contrastive Learning (CL) trở thành phương pháp được lựa chọn. Các công trình hiện tại đã sử dụng CL đơn (sử dụng các cặp Code-Code) (Guo et al., 2022; Jain et al., 2021) hoặc CL đôi (sử dụng các cặp Text-Code) (Li et al., 2022) cho việc học biểu diễn. Trong CL đơn, một lựa chọn phổ biến là sử dụng bổ sung dropout (Gao et al., 2021) để xây dựng các cặp mã tích cực. Tuy nhiên, chúng tôi nhận thấy rằng bổ sung dropout gặp khó khăn trong việc hỗ trợ quá trình huấn luyện dài, điều này cũng được báo cáo bởi Zhou et al. (2022). Ngược lại, CL đôi trở thành một lựa chọn hấp dẫn, chủ yếu là do sự sẵn có của các cặp tự nhiên. Các nghiên cứu trước đây sử dụng các hàm và docstring tương ứng của chúng để thiết lập các cặp huấn luyện đa mô. Tuy nhiên, các thí nghiệm sơ bộ của chúng tôi chỉ ra rằng sự trùng lặp đáng kể giữa docstring và chữ ký hàm làm đơn giản hóa quá trình học đôi lập (xem thống kê ở Phụ lục A.6).



Ảnh 1: Tổng quan về các thành phần chính của CODESAGE cho học biểu diễn mã.

Để đạt được mục tiêu này, chúng tôi giới thiệu CODESAGE, một mô hình biểu diễn mã hóa hai chiều cho mã nguồn. Chúng tôi huấn luyện trước CODESAGE bằng cách sử dụng một lược đồ huấn luyện hai giai đoạn với một lượng lớn dữ liệu huấn luyện được tùy chỉnh (Kocetkov et al., 2022). Chúng tôi mô tả các thành phần chính của CODESAGE trong Hình 1. Đầu tiên, chúng tôi huấn luyện các bộ mã hóa hai chiều thông qua sự kết hợp của hai mục tiêu bổ sung cho nhau: Identifier Deobfuscation (DOBF) và MLM mà không theo quy tắc 80-10-10. Tương tự như một lập trình viên con người, việc tìm các tên có ý nghĩa cho các định danh bị làm xáo trộn yêu cầu mô hình phải có sự hiểu biết sâu sắc về ngữ nghĩa và cấu trúc của mã. Đồng thời, như một mục tiêu chung hơn, MLM bao phủ các khía cạnh khác ngoài các định danh của mã – điều này quan trọng để làm phong phú các tín hiệu huấn luyện, đặc biệt đối với các ví dụ dữ liệu có tên định danh không mang tính thông tin. Trong giai đoạn thứ hai, chúng tôi tận dụng các cặp (văn bản, mã) bằng bimodal contrastive learning (CL). Khác với các phương pháp hiện có chủ yếu dựa vào các cặp văn bản và mã tự nhiên, chúng tôi đề xuất một chiến lược để giảm khả năng mô hình học các lỗi tắt. Phương pháp của chúng tôi bao gồm chỉ sử dụng phần thân hàm trong khi bỏ qua chữ ký và các câu lệnh trả về. Chúng tôi cũng khai thác CL dựa trên các hard negative trong không

gian nhúng. Chúng tôi cho thấy rằng chiến lược xây dựng các cặp hard positive và hard negative như vậy đơn giản nhưng rất cần thiết cho việc bimodal contrastive learning hiệu quả.

Chúng tôi huấn luyện ba mô hình biểu diễn mã hóa hai chiều, cụ thể là CODESAGE-SMALL (130M), CODESAGE-BASE (356M) và CODESAGE-LARGE (1.3B). Chúng tôi đánh giá hiệu quả của phương pháp của mình trên nhiều nhiệm vụ phân biệt khác nhau, trong đó CODESAGE vượt trội hơn đáng kể so với các mô hình tiên tiến trước đó có kích thước tương tự trong hầu hết các nhiệm vụ. Để hiểu rõ các yếu tố góp phần vào việc học biểu diễn mã thành công, chúng tôi phân tích tỉ mỉ các thành phần chính của khung làm việc của mình và trình bày những phát hiện của chúng tôi cho các nghiên cứu trong tương lai.

## 2. CÁC NGHIÊN CỨU LIÊN QUAN.

**Embedding for Programming Languages:** gần đây đã chứng kiến sự gia tăng đáng kể trong việc học các biểu diễn đa mục đích để hỗ trợ nhiều nhiệm vụ tiếp theo trong ngôn ngữ lập trình. Feng et al. (2020a); Kanade et al. (2020); Li et al. (2023) lấy cảm hứng từ thành công trong xử lý văn bản và tối ưu hóa mục tiêu Masking Language Modeling (MLM) trên dữ liệu mã hóa thành dạng tuyến tính. Tương tự như văn bản, họ cũng tối ưu hóa với mục tiêu phát hiện token được thay thế (Clark et al., 2020) hoặc mục tiêu dự đoán câu tiếp theo (Devlin et al., 2019b) cho mã nguồn. Một dòng nghiên cứu khác tận dụng mặt cấu trúc của mã để cung cấp tín hiệu huấn luyện bổ sung. Trong số đó, Guo et al. (2021) tận dụng luồng dữ liệu để mã hóa mối quan hệ "nơi-giá-trị-xuất-phát" giữa các biến. Wang et al. (2021a); Jiang et al. (2021) chèn cấu trúc cú pháp từ abstract syntax tree (AST) thông qua các mục tiêu phụ trợ biến thể. Một công trình mới đây hơn (Guo et al., 2022) làm phẳng cấu trúc AST trực tiếp thành một chuỗi và mã hóa thông tin cú pháp qua các mục tiêu mô hình ngôn ngữ. Wang et al. (2021b); Lachaux et al. (2021) huấn luyện một mô hình ngôn ngữ từ chuỗi đến chuỗi để tái tạo mã gốc từ một mã bị làm rối các tên lớp, hàm và biến bằng các token đặc biệt. Giải mã mà không chỉ rõ nguồn dữ liệu và AST ngầm định mã hóa mà không liên quan đến các mục tiêu phụ trợ hoặc đầu vào phức tạp với sự phân cấp sâu, vì mô hình cần hiểu rõ sự phụ thuộc giữa các biến cũng như cấu trúc mã để dự đoán chính xác tên cho các định danh.

**Contrastive Learning:** Kể từ những thành công ban đầu của mạng Siamese (Hadsell et al., 2006), học đối lập đã được rộng rãi áp dụng trong học biểu diễn bằng mạng nơ-ron sâu. Song et al. (2016) mở rộng mất mát ba cụm nguyên bản bằng cách so sánh mỗi ví dụ tích cực với tất cả các âm tích cực trong cùng một lô, điều này đã cải thiện đáng kể hiệu quả học và được phổ biến hơn nữa bởi SimCLR (Chen et al., 2020). Tuy nhiên, khác với lĩnh vực tính toán nơi mà các dương tích cực hiệu quả có thể được đạt được bằng các biến đổi ngẫu nhiên của hình ảnh trong không gian đầu vào, việc tăng cường dữ liệu hiệu quả đã lâu là một thách thức trong NLP do tính rời rạc của đầu vào. Thách thức này được khẳng định thêm trong công trình của Gao et al. (2021) chỉ ra rằng dropout (Srivastava et al., 2014) như là biến cố dữ liệu tối thiểu thường hiệu quả hơn so với những biến cố được đạt được bằng cách hoạt động trong không gian đầu vào rời rạc, ví dụ như xóa và thay thế từ.

Một cách tiếp cận khác là các phương pháp đã được đề xuất để tận dụng các cặp tự nhiên làm dương tích cực. Zhou et al. (2022) xem xét các lời nói liên tiếp từ dữ liệu hội thoại như là dương tích cực, trong khi Neelakantan et al. (2022) xem xét các văn bản lảng giềng được khai thác từ internet. Một công trình rất gần đây (Wang et al., 2022) tận dụng các cặp câu hỏi và câu trả lời hoặc nhận xét từ StackExchange và Reddit. Trên cùng một hướng đi với ngôn ngữ lập trình, Guo et al. (2022); Wang et al. (2021a); Neelakantan et al. (2022) tận dụng các cặp (văn bản, mã) với văn bản được khai thác từ các docstring. Chúng tôi đi một bước xa hơn bằng việc tập trung vào xây dựng tích cực khó và âm tích cực khó, đây là thành phần chính cho học biểu diễn và cho phép chúng tôi đạt được các mô hình nhúng sẵn sàng sử dụng.

## METHOD

### 3. MASK LANGUAGE MODELING VÀ DEOBFUSCATION PRE-TRAINING

Cho một câu đầu vào với  $N$  tokens, ví dụ như  $x = [x_1, x_2, \dots, x_N]$ . Mask Language Model (MLM) sẽ cố gắng tối ưu hàm Loss sau:

$$L(x) = - \sum_{i \in M} \log P(x_i | x^M)$$

Ở đây  $M$  đại diện cho mask áp dụng trên chuỗi ban đầu. Phương trình trên cơ bản là dự báo các từ bị che (mask) với điều kiện câu đầu vào bị che.

Ví dụ như ta có câu đầu vào như sau: “Con mèo trèo lên cây”. Ta có thể mask từ “trèo” và từ “cây” với token [MASK] như sau: “Con mèo [MASK] lên [MASK]”. Sau đó chúng ta sẽ encoder các từ trong câu đó sang vector  $d$  chiều. Giả sử chúng ta ra được một tập các vector biểu diễn câu trên như sau  $D = [d_1, d_2, d_3, d_4, d_5]$  ( $d_3$  và  $d_5$  chính là biểu diễn vector của token [MASK]). Khi đó hàm Loss sẽ được tính như sau:

$$L(x) = -\log P(d_3 | d_1 d_2 d_4) - -\log P(d_5 | d_1 d_2 d_4)$$

**Giải mã (deobfuscation):** Chúng ta đầu tiên xem xét việc giải mã định danh (DOBF), trong đó tiền huấn luyện mô hình để dự đoán tên các định danh đã bị che giấu. Tương tự như các lập trình viên con người, để giải mã mã nguồn (dự đoán các định danh), mô hình cần phải hiểu cả ngữ nghĩa và cấu trúc của mã nguồn. Cũng cần lưu ý rằng các token ngôn ngữ tự nhiên (NL), tức là các docstring và comment, được loại trừ khỏi việc che giấu mã nguồn. Khi mô hình được huấn luyện để dự đoán tên định danh, nó có thể hưởng lợi từ việc nhìn vào và liên kết với các token NL trong comment hoặc docstring vì chúng thường mang ý nghĩa phong phú về mã nguồn. Kết quả là, mô hình được khuyến khích học các biểu diễn chung giữa ngôn ngữ lập trình và ngôn ngữ tự nhiên tốt hơn, điều này được thể hiện qua hiệu suất tìm kiếm NL2Code tốt hơn mà DOBF đạt được so với chiến lược che giấu ngẫu nhiên.

Giả sử chúng ta có một đoạn mã Python cơ bản đã bị xáo trộn như sau:



```
1 def func1(a, b):  
2     c = a + b  
3     return c
```

Trong ví dụ này, tên của hàm và các biến đã bị làm xáo trộn (obfuscated). Để giải mã các tên này, mô hình phải dự đoán tên định danh phù hợp.


### **Giải mã định danh (DOBF)**

**Mã nguồn gốc:**



```
1 def add_two_numbers(number1, number2):  
2     sum = number1 + number2  
3     return sum
```

**Mã đã bị làm xáo trộn:**




```
1 def func1(a, b):
2     c = a + b
3     return c
```

Mô hình sẽ được huấn luyện để hiểu rằng func1 thực ra nên là add\_numbers, a nên là num1, b nên là num2, và c nên là result. Để làm điều này, mô hình cần phải hiểu cả ngữ nghĩa và cấu trúc của đoạn mã.

### Token ngôn ngữ tự nhiên (NL):

Docstring và comment:



```
1 def func1(a, b):
2     """
3     Hàm này cộng hai số và trả về kết quả.
4     """
5     # Cộng hai tham số đầu vào
6     c = a + b
7     return c
8
```

Trong trường hợp này, docstring ("Hàm này cộng hai số và trả về kết quả.") và comment ("Cộng hai tham số đầu vào") không bị làm xáo trộn. Mô hình có thể sử dụng thông tin từ docstring và comment để dự đoán rằng func1 nên là add\_numbers, a là num1, b là num2, và c là result.

## Kết quả:

Khi mô hình được huấn luyện để giải mã các tên định danh như trên, nó sẽ học cách liên kết thông tin giữa ngôn ngữ lập trình và ngôn ngữ tự nhiên. Điều này dẫn đến việc cải thiện khả năng tìm kiếm mã từ mô tả ngôn ngữ tự nhiên (NL2Code).

**Che giấu Token Ngẫu nhiên (Random Masking):** Để cải thiện khả năng biểu diễn của mô hình bằng cách học từ nhiều hơn chỉ các định danh. Ví dụ, trong Python, khoảng 30% token của mã nguồn liên quan đến định danh, do đó, việc mã hóa thông tin từ 70% token còn lại có thể mang lại biểu diễn tốt hơn. Không phải lập trình viên nào cũng tuân theo quy ước đặt tên, ví dụ như sử dụng tên biến vô nghĩa như v1, v2, v3. Việc dự đoán những token này là không cần thiết và cung cấp tín hiệu huấn luyện rất hạn chế.

Chúng tôi không tuân theo quy ước 80-10-10 trong MLM tiêu chuẩn. Trong MLM tiêu chuẩn, Devlin và cộng sự đề xuất quy ước che giấu 80-10-10 (80% token bị che giấu, 10% token bị thay thế ngẫu nhiên, 10% token giữ nguyên). Chúng tôi không tuân theo vì mã nguồn bao gồm các token ngôn ngữ tự nhiên (NL) và mã nguồn (các định danh, từ khóa, toán tử). Việc thay thế ngẫu nhiên các token có thể làm tổn hại cả cấu trúc và ý nghĩa của mã nguồn, dẫn đến sự suy giảm trong việc học biểu diễn. Chúng tôi đã thử nghiệm và thấy rằng tỉ lệ che giấu ngẫu nhiên là 15% sẽ là con số tối ưu.

## 4. BIMODAL CONTRASTIVE LEARNING WITH HARD NEGATIVE VÀ HARD POSITIVE

Giả sử  $x_i, x_{i+}$  kí hiệu cho một cặp positive input, cặp này có thể được tạo ra bởi data augmentation từ anchor ban đầu (nhưng không đảm bảo việc các phần tử được tạo ra từ anchor là một cặp positive input),  $h_i, h_{i+}$  là representations của  $x_i, x_{i+}$  qua layer cuối cùng của encoder.

Đặt  $B = \{h_1, h_{1+}, h_2, h_{2+}, h_3, h_{3+}, \dots, h_N, h_{N+}\}$  kí hiệu cho một tập  $h_i, h_{i+}$  được chọn ngẫu nhiên thành một batch có N cặp. Chúng ta phải tối ưu hóa hàm Loss sau:

$$L(h_i, h_{i+}) = -\log \frac{\exp\left(\frac{\text{sim}(h_i, h_{i+})}{\tau}\right)}{\exp\left(\frac{\text{sim}(h_i, h_{i+})}{\tau}\right) + \sum_{k \in B \setminus \{i, i+\}} \gamma_i^k \exp\left(\frac{\text{sim}(h_i, h_k)}{\tau}\right)} - \log \frac{\exp\left(\frac{\text{sim}(h_{i+}, h_i)}{\tau}\right)}{\exp\left(\frac{\text{sim}(h_{i+}, h_i)}{\tau}\right) + \sum_{k \in B \setminus \{i, i+\}} \gamma_{i+}^k \exp\left(\frac{\text{sim}(h_{i+}, h_k)}{\tau}\right)}$$

Trong đó:

$\tau$  là temperature hyper-parameter chúng tôi đặt bằng 0.05 trong bài báo này.



$\text{sim}(h_i, h_{i+})$  là cosine similarity giữa  $h_i, h_{i+}$ .

$\gamma_i^k$  được định nghĩa như sau:

$$\gamma_i^k = \frac{\exp\left(\frac{\text{sim}(h_i, h_k)}{\tau}\right)}{\exp\left(\frac{\text{sim}(h_i, h_k)}{\tau}\right) + \sum_{j \in B \setminus \{i, i+, k\}} \gamma_i^k \exp\left(\frac{\text{sim}(h_i, h_j)}{\tau}\right)}$$

$\gamma_i^k$  đại diện cho tầm quan trọng tương đối của biểu diễn  $h_k$  đối với anchor  $h_i$ , trong số tất cả các negatives (ví dụ âm tính) trong batch hiện tại, ngoại trừ cặp đầu vào tích cực đã cho.

Cụ thể,  $\gamma_i^k$  là một giá trị đo lường mức độ khó khăn của một ví dụ âm tính  $h_k$  khi so sánh với anchor  $h_i$ .

## 5. CODESAGE-SMALL, CODESAGE-BASE, CODESAGE-LARGE

Chúng tôi huấn luyện ba mô hình biểu diễn mã nguồn hai chiều, gồm CODESAGE-SMALL (130 triệu tham số), CODESAGE-BASE (356 triệu tham số) và CODESAGE-LARGE (1.3 tỷ tham số). Chúng tôi đánh giá hiệu quả của phương pháp của mình trên một loạt các nhiệm vụ phân biệt, trong đó CODESAGE vượt trội hẳn so với các mô hình tiên tiến trước đó có kích thước tương tự trên hầu hết các nhiệm vụ. Để hiểu được những yếu tố góp phần vào việc học biểu diễn mã nguồn thành công, chúng tôi phân tích tỉ mỉ các thành phần chính của khung công việc của chúng tôi và trình bày những kết quả của mình để định hướng cho các nỗ lực nghiên cứu trong tương lai.

### Điểm mạnh của các mô hình:

#### CODESAGE-SMALL (130M):

- Đạt được cải tiến đáng kể so với các mô hình cơ sở trên các nhiệm vụ Code2Code search và NL2Code search.
- Cải thiện tương đối từ 18.54% đến 51.1% so với UnixCoder trên NL2Code search.
- Cải thiện tương đối 12.86% trên CosQA và 8.4% trên AdvTest so với OpenAI-Embedding-Ada-002.
- Hiệu suất tốt trên các nhiệm vụ phức tạp như dự đoán phức tạp và lỗi thời gian chạy.

#### CODESAGE-BASE (356M):



- Với kích thước mô hình lớn hơn, CODESAGE-BASE cải thiện từ 20.56% đến 40.91% so với các mô hình cơ sở trên các nhiệm vụ Code2Code search và NL2Code search.
- Đạt được hiệu suất vượt trội so với OpenAI-Embedding-Ada-002, đặc biệt là trên các nhiệm vụ như CosQA và AdvTest.

### **CODESAGE-LARGE (1.3B):**

- Mô hình lớn nhất trong loạt CODESAGE, cải thiện tương đối từ 40.91% trở lên so với các mô hình cơ sở trên các nhiệm vụ Code2Code search và NL2Code search.
- Đạt được hiệu suất cao trên các nhiệm vụ phức tạp như AdvTest, trong đó yêu cầu mô hình phải hiểu và xử lý mã nguồn được làm mờ.

### **Điểm yếu của các mô hình:**

#### **CODESAGE-SMALL:**

- Dưới hiệu suất so với mô hình OpenAI-Embedding-Ada-002 trên nhiệm vụ CSN.
- Dưới hiệu suất so với UnixCoder và OpenAI-Embedding-Ada-002 trên phát hiện lỗi mã nguồn.

#### **CODESAGE-BASE VÀ CODESAGE-LARGE:**

- Cũng có thể dưới hiệu suất so với UnixCoder và OpenAI-Embedding-Ada-002 trên nhiệm vụ phát hiện lỗi mã nguồn.

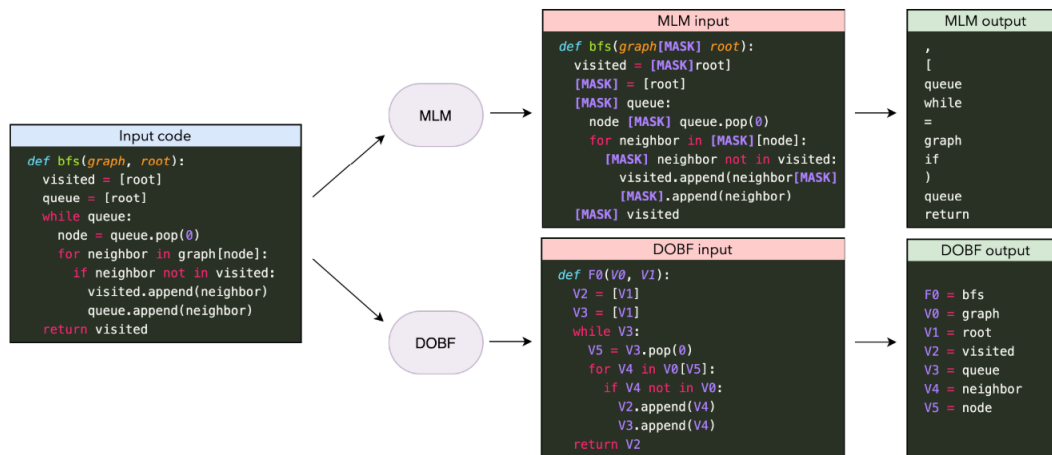
### **Tóm tắt:**

**Điểm mạnh:** Các mô hình CODESAGE có hiệu suất vượt trội trên các nhiệm vụ Code2Code search và NL2Code search so với các mô hình cơ sở và mô hình tiên tiến trước đó.

**Điểm yếu:** Mặc dù có hiệu suất cao trên nhiều nhiệm vụ, CODESAGE vẫn có thể gặp khó khăn trong việc phân tích lỗi và thực hiện phát hiện các lỗi trong mã nguồn so với các mô hình khác.

Điều này cho thấy CODESAGE đặc biệt mạnh mẽ trong việc biểu diễn mã nguồn và giải quyết các nhiệm vụ phức tạp, nhưng vẫn còn tiềm ẩn một số thách thức trong các nhiệm vụ đòi hỏi phát hiện lỗi và xử lý mã nguồn không hoàn hảo.

## 6. SỰ KHÁC BIỆT GIỮA MLM VÀ DOBS



Hình ảnh minh họa về MLM và DOBF. Trên nhiệm vụ MLM, các mã nguồn được ngẫu nhiên che đi. Đối với mã nguồn, các phần như cú pháp (ví dụ như dấu phẩy, dấu ngoặc) thường dễ đoán và ít hữu ích cho mô hình học. Thay vào đó, DOBF che tên hàm và biến, huấn luyện mô hình giải mã lại mã nguồn ban đầu. Khi một biến bị che, tất cả các lần xuất hiện của nó đều được thay thế bằng một ký hiệu che duy nhất để mô hình không sao chép tên. Mục tiêu của DOBF khó hơn nhưng cung cấp tín hiệu học tập tốt hơn.

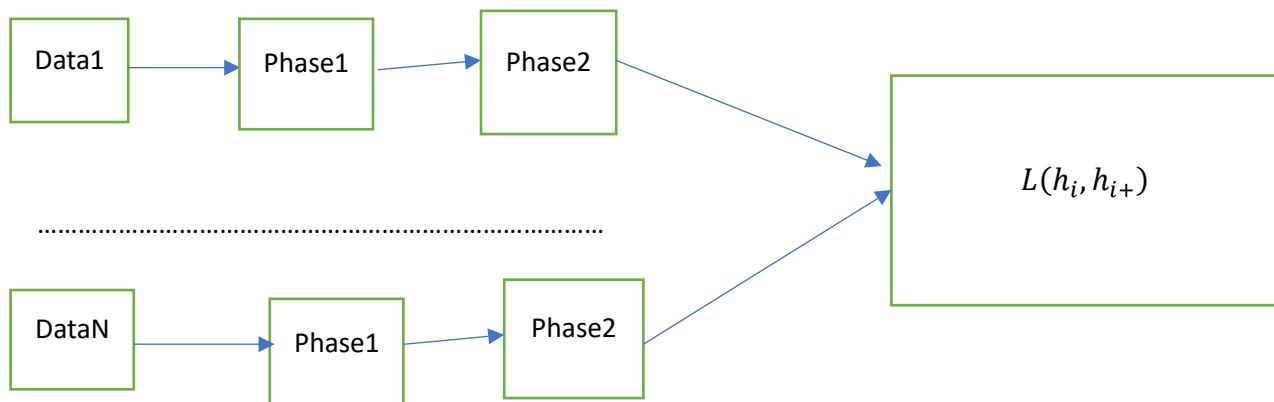
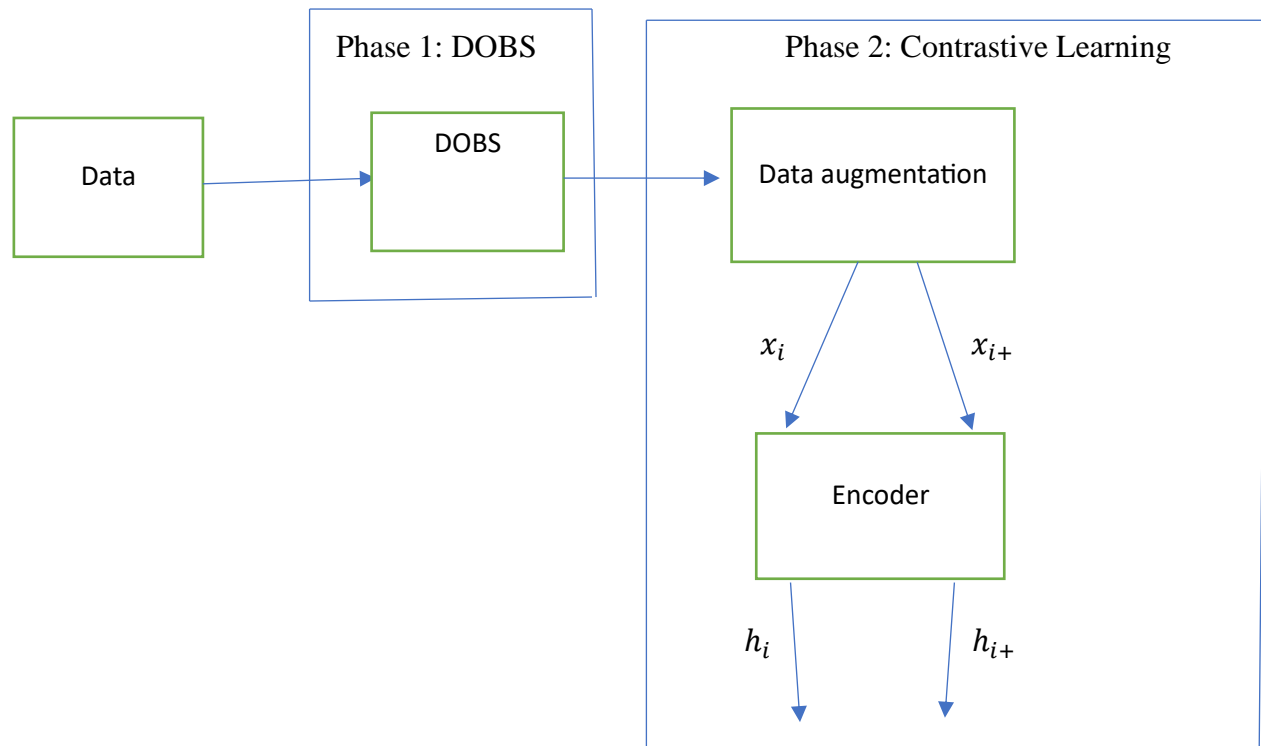
DOBF (Deobfuscation) được xem là tốt hơn MLM (Masked Language Modeling) vì nó cung cấp một tín hiệu huấn luyện tốt hơn đối với mô hình ngôn ngữ. Các lý do chính bao gồm:

- ✓ **Khó khăn hơn trong giải mã:** Việc che đi tên của các hàm và biến là một thử thách lớn hơn so với việc đơn giản che các mã thông báo ngẫu nhiên. Điều này yêu cầu mô hình phải hiểu và phân tích cú pháp, cấu trúc và logic của mã nguồn để có thể khôi phục lại thông tin gốc.
- ✓ **Tín hiệu huấn luyện rõ ràng hơn:** Khi một biến được che đi, việc phải tìm cách giải mã lại các lần xuất hiện của biến này yêu cầu mô hình phải học cách biểu diễn các khái niệm ngữ nghĩa hơn là chỉ dựa vào nhận diện cú pháp. Điều này cung cấp cho mô hình một tín hiệu học tập rõ ràng hơn và giúp nó phát triển khả năng phân tích sâu hơn về cấu trúc và ý nghĩa của mã nguồn.
- ✓ **Áp dụng thực tế cao hơn:** Việc áp dụng DOBF hữu ích trong các bối cảnh thực tế hơn vì nó giúp cải thiện hiểu biết của mô hình về mã nguồn và giảm thiểu sự phụ thuộc vào các chi tiết cú pháp không quan trọng..

## 7. LUỒNG HOẠT ĐỘNG CỦA CODESAGE

Để hiểu rõ các yếu tố góp phần vào sự thành công trong việc biểu diễn mã, chúng tôi sẽ loại bỏ các chi tiết và chia sẻ sự phát hiện của chúng tôi về (i) sơ đồ khử nhiễu mức mã thông báo tùy chỉnh và hiệu quả cho mã nguồn; (ii) tầm quan trọng của hard negative và hard positive; (iii) bimodal contrastive learning trong việc thúc đẩy hiệu suất tìm kiếm ngữ nghĩa đa ngôn ngữ; (iv) cách các pretraining quyết định các hiệu suất của downstream task theo kích thước mô hình.

Data sẽ được xử lý ngẫu nhiên qua DOBS hoặc model với từng training example. Theo bài gốc về DOBS thì họ đánh giá DOBS tốt hơn MLM nên chúng tôi sẽ sử dụng DOBS trong mô hình CodeSage. Sơ đồ của mô hình CodeSage như sau:



Work flow của CodeSage:

- **Phase 1:** Data được đưa qua DOBS để che tên hàm và biến bằng cách thay thế kí tự nào đó.
- **Phase 2:** Data sẽ được tăng cường dữ liệu và sinh ra hai data mới là  $x_i$  và  $x_{i+}$  bằng cách áp dụng hai cách data augmentation khác nhau. Sau đó hai data này sẽ được cho qua một bộ Encoder để embedding (bài báo sử dụng embedding của openai). Sau khi embedding ta sẽ được hai vector biểu diễn data là  $h_i$  và  $h_{i+}$ . Đây chính là 1 cặp positive để đưa vào hàm Loss.
- Tính tương tự với tất cả training examples, ta sẽ thu được hàm Loss  $L(h_i, h_{i+})$ . Vì mục tiêu của chúng ta là tối thiểu hóa hàm Loss mà hàm Loss lại là hiệu của các hàm log. Tối thiểu hóa hàm loss thì optimizer sẽ cần đưa giá trị tuyệt đối của hàm loss về gần 0 nhất có thể, nghĩa là giá trị của hàm log phải xấp xỉ 1. Tức là tử số phải gần với mẫu số. Để làm điều này thì buộc phải ép cho vector biểu diễn của cặp positive càng giống nhau càng tốt và ép cho vector biểu diễn của các cặp negative càng khác nhau càng tốt. Nên việc tối thiểu được hàm Loss cũng chính là việc ta “phân cụm” dữ liệu. Ta dùng Backpropagation để tính đạo hàm các tham số ở các block như Encoder và Data augmentation, và giảm gradient để cập nhật tham số.
- Mục đích của CodeSage là để embedding các mã nguồn về vector sao cho các cặp vector positive thì nằm “gần nhau”, các cặp vector negative thì nằm “xa nhau”, đem lại cách biểu diễn tốt để phân loại mã bình thường và mã chứa lỗ hổng.
- Sau khi training xong thì ta sẽ không cần phải Data augmentation để tạo ra hai vector nữa mà chỉ cần 1 vector để biểu diễn mã nguồn.