META-GENERATION METHOD FOR LARGE LANGUAGE MODELS

Hoang Nhat Duong
Institute of Information Technology - Vietnam Academy of Science and Technology

ARTICLE INFO

Received: 21/3/2025
Revised: 26/6/2025
Published: 28/6/2025

KEYWORDS

Meta-generation
Chain-of-Thought
Reinforcement learning
Generator
Fine-tuning

ABSTRACT

This study addresses the question: How can we enhance the accuracy and efficiency of natural language processing by optimizing the output generation process? The goal is to develop a meta-generation method that improves the quality of large language model outputs through systematic feedback and refinement steps. The research methodology is structured around a three-stage process: (1) generating an initial output from the model, (2) collecting feedback to identify errors, and (3) refining the output based on the feedback to produce a more accurate result. A key innovation of this approach lies in decomposing the problem into smaller sub-tasks, generating multiple candidate outputs, and then applying a reward model or voting mechanism to select the optimal answer. The results indicate that the meta-generation approach significantly improves model accuracy by incorporating step-by-step verification, feedback, and candidate selection. Experimental data (if available) demonstrate that the refined model outperforms single-pass generation models in terms of output quality. This approach demonstrates clear potential in enhancing reasoning performance and the output quality of language models.

PHƯƠNG PHÁP META-GENERATION CHO CÁC MÔ HÌNH NGÔN NGỮ LỚN

Hoàng Nhật Dương
Viện Công nghệ Thông tin – Viện Hàn lâm Khoa học và Công nghệ Việt Nam

THÔNG TIN BÀI BÁO

Ngày nhận bài: 21/3/2025
Ngày hoàn thiện: 26/6/2025
Ngày đăng: 28/6/2025

TỪ KHÓA

Meta-generation
Chuỗi suy luận
Học tăng cường
Mô hình sinh
Tinh chỉnh

TÓM TẮT

Nghiên cứu đặt ra câu hỏi: Làm thế nào để cải thiện độ chính xác và hiệu quả trong xử lý ngôn ngữ tự nhiên bằng cách tối ưu hóa quy trình sinh đầu ra? Mục đích là phát triển một phương pháp meta-generation giúp nâng cao chất lượng đầu ra của mô hình ngôn ngữ lớn thông qua các bước phản hồi và điều chỉnh có hệ thống. Phương pháp nghiên cứu tập trung vào việc xây dựng quy trình ba giai đoạn: (1) sinh đầu ra ban đầu từ mô hình, (2) thu thập phản hồi để phát hiện sai sót, và (3) điều chỉnh đầu ra dựa trên phản hồi nhằm tạo kết quả chính xác hơn. Điểm nổi bật của phương pháp là chia nhỏ bài toán thành các bước cụ thể, sinh ra nhiều ứng viên đầu ra, sau đó sử dụng mô hình thưởng hoặc cơ chế bỏ phiếu để chọn đáp án tối ưu. Kết quả nghiên cứu cho thấy cách tiếp cận meta-generation giúp cải thiện độ chính xác của mô hình thông qua kiểm tra, phản hồi và chọn lọc theo từng bước. Số liệu thực nghiệm (nếu có) minh chứng rằng mô hình điều chỉnh có hiệu suất vượt trội so với các mô hình chỉ sinh đầu ra một lần duy nhất. Cách tiếp cận này cho thấy tiềm năng rõ rệt trong việc nâng cao hiệu suất suy luận và chất lượng đầu ra của mô hình ngôn ngữ.

DOI: https://doi.org/10.34238/tnu-jst.12364

Email: nhatduonghoang59@gmail.com

1. Giới thiệu

Gần đây, hậu huấn luyện đã trở thành một thành phần quan trọng trong toàn bộ quy trình đào tạo mô hình, giúp cải thiện độ chính xác trong các nhiệm vụ suy luận, điều chỉnh mô hình phù hợp với các giá trị xã hội và thích ứng với sở thích của người dùng. Đáng chú ý, hậu huấn luyện đạt được những cải tiến này với chi phí tính toán tương đối thấp hơn so với quá trình tiền huấn luyện. Trong bối cảnh khả năng suy luận, Meta-generation đã nổi lên như một phương pháp đầy hứa hẹn nhằm mở rộng quy mô thời gian suy luận bằng cách tối ưu hóa và tăng cường quá trình Chuỗi suy luận (Chain-of-Thought - CoT). Một số nghiên cứu trước đây đã có được những kết quả nhất định. Các tác giả trong [1] giới thiệu mô hình Switch Transformer, một biến thể của kiến trúc Transformer sử dụng cơ chế "mixture of experts" với độ thưa (sparsity) cao. Thay vì mọi token được chuyển qua cùng một khối Transformer đầy đủ, Switch Transformer điều hướng mỗi token tới đúng "expert" phù hợp, qua đó giảm lượng tính toán và cho phép mở rộng quy mô mô hình lên tới hàng ngàn tỉ tham số. Feng và cộng sự [2] phân tích chuỗi suy luận trong các mô hình ngôn ngữ từ góc độ lý thuyết. Họ đưa ra khuôn khổ toán học và các giả thiết để giải thích cách mô hình có thể tận dụng chuỗi suy luận nhằm phân tách quy trình tư duy thành nhiều bước nhỏ, đồng thời hạn chế sai sót do ghi nhớ hoặc suy luận nội bộ không đủ dung lượng. Kết quả nghiên cứu góp phần làm rõ tính hiệu quả của chain-of-thought, vốn đang được áp dụng rộng rãi trong lĩnh vực xử lý ngôn ngữ tự nhiên. Finlayson và cộng sự [3] phân tích những nguyên nhân cốt lõi dẫn đến cơ chế lấy mẫu (sampling) hoặc việc mô hình bị "nghẽn" trong một kiểu phân phối hẹp. Nhóm nghiên cứu đề xuất một số kỹ thuật và điều chỉnh quy trình sinh nhằm khắc phục, bao gồm chiến lược thay đổi nhiệt (temperature), cắt chuỗi suy luận, hoặc thêm bộ lọc đánh giá liên tục để tránh mô hình rơi vào vòng lặp lạc lối. Kết quả chỉ ra rằng với những giải pháp hợp lý, ta có thể "đóng" hiện tượng suy thoái văn bản và cải thiện rõ rệt chất lượng nội dung sinh. Hobbahn và các cộng sự [4] phân tích các xu hướng gần đây về phần cứng phục vụ học máy, bao gồm sự phát triển của bộ xử lý chuyên dụng (như GPU, TPU), các kiến trúc đa lõi và giải pháp tăng tốc bằng FPGA, cũng như mối quan hệ giữa thiết kế chip với yêu cầu huấn luyện mô hình lớn. Họ thảo luận về tác động của quy mô mô hình trí tuệ nhân tạo (AI) đến chi phí điện năng, khả năng tản nhiệt và hướng tối ưu hoá hiệu suất/tài nguyên. Tulchinskii cùng cộng sự [5] phân tích nguyên nhân từ cách lấy mẫu (sampling) cùng các thiết lập tham số (như nhiệt độ hay top-k). Họ đề xuất phương pháp điều chỉnh (chẳng hạn nucleus sampling) nhằm giảm thiểu tình trạng văn bản thoái hoá, đồng thời chỉ ra tầm quan trọng của cách tiếp cận cân bằng giữa sáng tạo và tính ổn định khi sinh văn bản. Li và cộng sự [6] tập trung vào khả năng tự phát hiện và tự sửa lỗi lập luận của các mô hình ngôn ngữ lớn. Công trình gợi ý rằng "self-correction" đòi hỏi cả cơ chế suy luận rõ ràng hơn và khả năng đánh giá trung gian, chứ không chỉ dựa vào tham số đã học. He và cộng sự [7] đề xuất cách tiếp cận "Draft-Sketch-Prove" để tận dụng bản chứng minh không chính thức (informal proof) nhằm điều hướng trình chứng minh định lý (theorem prover) chính thức. Họ chia quy trình thành ba giai đoạn: phác thảo lời giải bằng ngôn ngữ tự nhiên, chuyển phác thảo sang dạng sketch gần với cú pháp formal, rồi để trình prover kiểm chứng. Qua đó, nhóm tác giả cho thấy việc cung cấp lời giải không chính thức có thể rút ngắn thời gian tìm được chứng minh formal và giảm lỗi logic. Juravsky và cộng sự [8] giới thiệu kỹ thuật Hydragen, tập trung tối ưu hóa quá trình suy luận (inference) cho các mô hình ngôn ngữ lớn (LLM) bằng cách chia sẻ "prefix" trong những trường hợp câu đầu hoặc ngữ cảnh giống nhau. Thay vì tính toán lặp cho từng đầu vào độc lập, hệ thống gom các phần trùng lặp lại một lần, qua đó tăng thông lượng (throughput) và tiết kiệm thời gian. Kết quả thực nghiệm chỉ ra rằng việc chia sẻ prefix có thể giúp tận dụng tài nguyên tốt hơn, đặc biệt khi phải xử lý nhiều truy vấn tương đồng trong thời gian ngắn.

Trong những năm gần đây, đã có sự quan tâm ngày càng lớn đối với việc tinh chỉnh các mô hình ngôn ngữ lớn thông qua những phương pháp hậu huấn luyện vượt xa cách tiếp cận tinh chỉnh truyền thống [9]. Các kỹ thuật mở rộng này thường nhấn mạnh vào tính giải thích được, khả năng hoạt động ổn định trên nhiều lĩnh vực, và sự tuân thủ về mặt đạo đức với các chuẩn mực xã hội. Bằng cách liên tục cập nhật và hoàn thiện hành vi của mô hình, nhiều nghiên cứu đã cho thấy hậu

huấn luyện có thể cải thiện đáng kể khả năng thích ứng của mô hình với nhu cầu phức tạp của người dùng, xử lý các đầu vào mơ hồ và sinh ra những phản hồi đáng tin cậy hơn. Hơn nữa, việc tích hợp các chiến lược lấy mẫu nâng cao cùng phương pháp chuỗi suy luận góp phần nâng cao năng lực suy luận phức tạp của mô hình [10]. Khi được kết hợp với các tiếp cận meta-generation, những phương pháp này giúp nâng cao hiệu quả và độ chính xác của quá trình suy luận, thu hẹp khoảng cách giữa đổi mới lý thuyết và triển khai thực tế. Thách thức quan trọng hiện nay là thiết kế những kiến trúc và thuật toán có khả năng mở rộng để xử lý khối lượng tính toán khổng lồ khi đào tạo quy mô lớn [11]. Trong tương lai, nghiên cứu có thể hướng đến hệ thống lai ghép giữa các tuyến suy luận tường minh với các thành phần học sâu, cũng như khai thác giải pháp phần cứng hiệu quả để giảm thiểu độ trễ. Với đà phát triển hiện tại, chúng ta có thể kỳ vọng thế hệ hệ thống thông minh mới vừa sâu về mặt hiểu biết, vừa tối ưu về hiệu năng tính toán.

Mục tiêu của nghiên cứu này là thiết kế một hệ thống G có khả năng sinh ra các chuỗi chấp nhận được. Điều này được biểu diễn bằng bài toán tối ưu hóa: $\arg\max_G E_{y\sim G(.)}A(y)$ trong đó G là hệ thống sinh chuỗi cần thiết kế, y là các chuỗi được tạo ra bởi hệ thống G, A(*y*) là hàm đánh giá chất lượng của chuỗi *y*. Trong nghiên cứu này, chúng tôi sẽ tập trung vào việc tối ưu hóa mô hình sinh dữ liệu, sử dụng các kỹ thuật học máy (Machine Learning) như học tăng cường (Reinforcement Learning), học có giám sát (Supervised Learning) hoặc các phương pháp tinh chỉnh mô hình (Fine-tuning) để đảm bảo rằng đầu ra của hệ thống đạt được các tiêu chí chất lượng mong muốn.

2. Phương pháp luận và mô hình

Mô hình được đề xuất nhằm mục tiêu xây dựng một hệ thống G có khả năng sinh (generate) ra các chuỗi đầu ra y mà mức độ chấp nhận được (acceptability) là cao nhất theo một hàm đánh giá A(·). Ta có thể hình dung G như một "chính sách" hoặc "phân phối" quyết định cách lấy mẫu chuỗi y từ không gian vô cùng lớn các dãy ký tự hoặc token. Công thức tổng quát được viết dưới dạng:

$$\arg\max_G \mathsf{E}_{y\sim G(\cdot)}\left[A(y)\right] \quad (1)$$

Nghĩa là ta muốn tìm một cách thức sinh các chuỗi sao cho, trung bình (kỳ vọng) giá trị A(y) của các chuỗi ấy là lớn nhất. Để cụ thể hơn, ta có thể xem A(*y*) giống như một hàm thưởng hoặc hàm "lợi ích" trong học tăng cường [12]: một chuỗi y càng được xem là phù hợp hoặc đúng đắn thì A(*y*) càng cao. Khi thay đổi G, tức thay đổi quy luật sinh hay "phân phối" trên không gian các chuỗi, kỳ vọng $E_{y\sim G(.)}\left[A(y)\right]$ cũng thay đổi. Nhiệm vụ của ta là điều chỉnh hệ thống G – có thể bằng cách huấn luyện, tinh chỉnh, hay thiết kế bộ giải mã (decoding) – để tối đa hóa giá trị này.

Meta-generator đề cập đến một cách tiếp cận trong đó thay vì chỉ sử dụng một mô hình sinh duy nhất, ta triển khai nhiều mô hình sinh chuyên biệt khác nhau và điều phối chúng thông qua một chiến lược tổng quát gọi là G. Trong kịch bản này, G đảm nhiệm vai trò quản lý, quyết định xem nên gọi mô hình sinh nào (ví dụ $g_1, g_2, ..., g_G$) vào thời điểm nào, trình tự ra sao, hoặc kết hợp kết quả của chúng thế nào để tạo ra một đầu ra y đáp ứng tốt nhất yêu cầu đề ra. Về mặt công thức, ta có thể hình dung quá trình tạo ra đầu ra y như sau: $y \sim G(y, x; g_1, g_2, ..., g_G, \phi)$, trong đó x là ngữ cảnh hoặc dữ liệu đầu vào, còn ϕ bao gồm các tham số phụ như số lượng token cần sinh, cách trộn các đầu ra, quy tắc ngắt câu, mô hình lọc độ độc hại, hoặc cách thức tích hợp tri thức bên ngoài. Ý tưởng cốt lõi là mỗi mô hình sinh g_i có thể được huấn luyện hoặc tinh chỉnh cho một mục tiêu riêng, chẳng hạn mô hình tóm tắt, mô hình dịch, mô hình trả lời câu hỏi, hay mô hình sinh văn bản phong cách trang trọng và G sẽ cân nhắc gọi đúng mô hình phù hợp theo bối cảnh.

Trong thực hành, ta có thể biểu diễn quá trình meta-sinh như sau:

$$y = G(g_1, g_2, \ldots, g_n; x, \varphi) \quad (2)$$

trong đó:

- x: Ngữ cảnh hoặc dữ liệu đầu vào (ví dụ: văn bản, đoạn hội thoại)

- g_1, g_2, ..., g_n: Các mô hình sinh chuyên biệt (ví dụ: mô hình dịch, mô hình tóm tắt, mô hình viết văn trang trọng,...)
- φ: Các tham số phụ điều chỉnh quá trình sinh, chẳng hạn:

a) Số lượng token cần sinh
b) Chiến lược trộn đầu ra
c) Quy tắc ngắt câu
d) Bộ lọc độ độc hại
e) Cách tích hợp tri thức từ nguồn bên ngoài

Hình 1. *Hệ thống mô hình xử lý tuần tự nhiều giai đoạn*

Bộ điều khiển G sử dụng những yếu tố này để linh hoạt điều phối việc sử dụng mô hình phù hợp, từ đó sinh ra đầu ra y tốt nhất theo yêu cầu.

Trong **chatbot đa ngôn ngữ**, G có thể:

a) Gọi mô hình **dịch ngôn ngữ** để hiểu đầu vào
b) Gọi mô hình **trả lời câu hỏi** cho truy vấn kiến thức
c) Gọi mô hình **điều chỉnh phong cách** để thay đổi giọng điệu (trang trọng/thân mật)

Trong **trợ lý viết tài liệu**, G có thể:

a) Gọi mô hình **lập dàn ý**
b) Sau đó là mô hình **soạn thảo**
c) Cuối cùng là mô hình **hiệu đính hoặc viết lại**

Ở dạng đơn giản nhất, G có thể là một **hàm toán học xác định**, ánh xạ từ đầu vào x và các tham số φ sang **một chuỗi các mô hình được gọi** theo thứ tự cụ thể:

$$G:(x,\varphi)\rightarrow \text{serie models } \{g_i\} \tag{3}$$

Ví dụ: $G(x,\varphi)=g_3(g_2(g_1(x)))$ nghĩa là x được xử lý bởi g_1, kết quả được chuyển sang g_2, cuối cùng được đưa vào g_3 để cho ra đầu ra y.

Việc kết hợp các mô hình sinh là một cách tiếp cận nhằm tận dụng từng thế mạnh riêng của nhiều mô hình trong quá trình tạo ra đầu ra cuối cùng. Thay vì chỉ sử dụng một mô hình duy nhất để xử lý từ đầu đến cuối, ta chia quá trình thành các bước trung gian, trong đó mỗi bước được thực hiện bởi một mô hình sinh khác nhau (gọi là g_1, g_2, g_3,…). Khi có một đầu vào x, mô hình g_1 sẽ sinh ra kết quả trung gian y_1. Tiếp theo, mô hình g_2 tiếp nhận cả thông tin ban đầu x và kết quả y_1 để cho ra y_2. Cứ thế, mô hình g_3 lại dựa trên x và y_2, và quá trình tiếp diễn cho đến khi ta đạt được đầu ra mong muốn giống như trong Hình 1. Ý tưởng then chốt của phương pháp Chain-of-thought nằm ở chỗ ta buộc mô hình phải sinh ra một chuỗi lập luận trung gian - được gọi là "thought" (hay lời giải thích), trước khi kết luận kết quả. Cách làm này khác so với mô hình truyền thống vốn chỉ học mối quan hệ "đầu vào x $\rightarrow$ đầu ra a". Giờ đây, ta thêm bước trung gian: trước hết sinh "thought" "$a \sim g(.,x,z)$", tức là mô hình phải viết ra hoặc hình thành quá trình lý giải, phân tích, hay mô tả từng bước. Kế đó, mô hình mới sinh đáp án "$a \sim g(.,x,z)$", nghĩa là sử dụng cả thông tin ban đầu lẫn lời giải thích để đưa ra kết luận cuối cùng. Lợi ích của việc này là mô hình học được cách diễn giải mạch lạc, từ đó giảm thiểu sai sót do "nhảy cóc" hoặc thiếu liên kết giữa các dữ liệu. Hơn nữa, khi yêu cầu mô hình ghi lại quá trình suy nghĩ, ta có thể kiểm tra và đánh giá tính hợp lý của từng bước. Nếu phát hiện điểm nào vô lý, ta điều chỉnh hoặc yêu cầu mô hình suy luận lại. Điều này cũng giúp người dùng hiểu rõ cách mô hình đi đến câu trả lời, tăng tính minh bạch và tin cậy cho hệ thống. Chẳng hạn, khi mô hình tự giải thích

rằng "có 23 quả táo, dùng 20, còn lại 3, sau đó mua 6 quả, thành 9", ta dễ dàng xác minh lập luận này đúng với thực tế và con số 9 hoàn toàn chính xác (Hình 2).

```python
def chain_of_thought_add(a: int, b: int):
    """
    Hàm mô phỏng quá trình 'chain-of-thought' khi cộng hai số.
    Trả về cả lời giải thích lẫn đáp án.
    """
    # Chuỗi lập luận (thought)
    explanation = (
        f"Bước 1: Ta có hai số {a} và {b}.\n"
        f"Bước 2: Thực hiện phép cộng: {a} + {b}.\n"
        f"Bước 3: Kết quả: {a + b}."
    )

    # Đáp án cuối (answer)
    answer = a + b

    return explanation, answer
```

```python
# Ví dụ sử dụng
if __name__ == "__main__":
    x, y = 23, 6

    # Gọi hàm, thu được cả chain-of-thought lẫn kết
    reasoning, result = chain_of_thought_add(x, y)

    # In ra quá trình suy luận
    print("Chuỗi tư duy (Chain-of-thought):")
    print(reasoning)

    # In ra kết quả
    print("Kết quả cuối (Answer):", result)
```

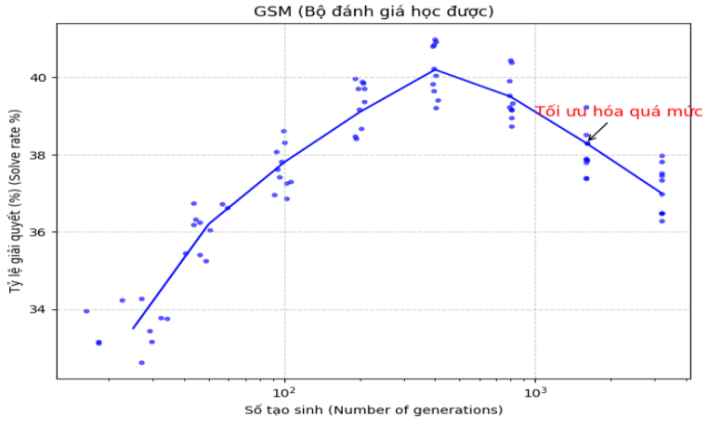
Hình 2. *Ví dụ về chuỗi suy luận*

Gần đây, Sean Welleck và các cộng sự [13] công bố trên arXiv vào tháng 6 năm 2024, tập trung vào việc mở rộng các phương pháp sinh văn bản trong các mô hình ngôn ngữ lớn (LLMs) trong giai đoạn suy luận. Bài báo khám phá ba lĩnh vực dưới một hình thức toán học thống nhất: các thuật toán sinh cấp độ token, các thuật toán meta-generation, và các phương pháp sinh hiệu quả. Các thuật toán sinh cấp độ token, thường được gọi là các thuật toán giải mã, hoạt động bằng cách lấy mẫu một token tại một thời điểm hoặc xây dựng không gian tìm kiếm cấp độ token và sau đó chọn một đầu ra. Các phương pháp này thường giả định có quyền truy cập vào các logit của mô hình ngôn ngữ, các phân phối token tiếp theo, hoặc các điểm xác suất. Các thuật toán meta-generation làm việc trên các chuỗi một phần hoặc toàn bộ, kết hợp kiến thức chuyên ngành, cho phép quay lui và tích hợp thông tin bên ngoài. Các phương pháp sinh hiệu quả giúp giảm chi phí token và cải thiện tốc độ sinh. Những chủ đề trong bài báo tương đồng chặt chẽ với nghiên cứu này về các chiến lược khác nhau (ví dụ: chain-of-thought, song song, tìm kiếm theo cây và tinh chỉnh) và cách điều phối chúng để tối ưu đầu ra của mô hình ngôn ngữ.

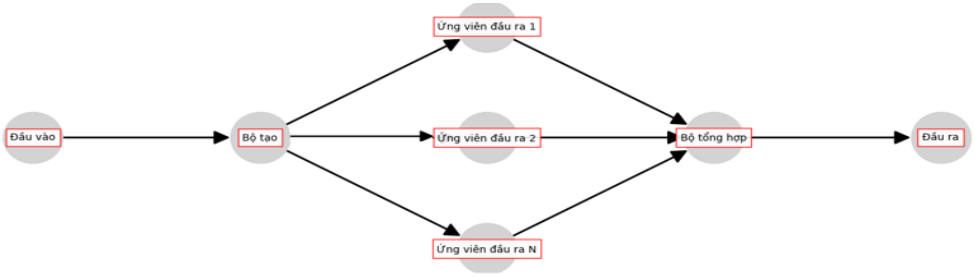
Thông thường các tác nhân AI tạo sinh sẽ sinh ra dữ liệu từ Prompt đầu vào nhưng độ chính xác thường rất thấp. Hình 3 minh họa cách tỷ lệ giải quyết (%) của GSM (một bộ đánh giá học được) thay đổi khi số tạo sinh tăng từ 10 đến hơn 1000. Mỗi chấm màu xanh thể hiện một phép đo tỷ lệ giải quyết và đường xu hướng màu xanh cho thấy hiệu năng ban đầu tăng lên, đạt đến đỉnh rồi giảm dần. Tỷ lệ giải quyết cao nhất, khoảng 40% hoặc hơn một chút, xuất hiện ở vùng giữa của số thế hệ, cho thấy hệ thống học hiệu quả đến một mức nhất định. Sau điểm cực đại này, hiệu năng giảm, cho thấy dấu hiệu quá khớp hoặc "tối ưu hóa quá mức," khi hệ thống trở nên quá chuyên biệt vào dữ liệu đã được huấn luyện, dẫn đến giảm khả năng giải quyết các trường hợp mới. Mô hình này nhấn mạnh tầm quan trọng của việc lựa chọn số thế hệ tối ưu trong các quy trình tiến hóa hoặc học máy, vì số thế hệ quá ít có thể gây thiếu khớp (chưa học đủ), trong khi quá nhiều lại dẫn đến quá khớp và suy giảm hiệu năng.

Chọn ứng viên có điểm cao nhất $\arg\max_{\{y^{(1)},y^{(2)},...,y^{(N)}\}} v(y)$, trong đó, $v(y)$ là mô hình phần thưởng đánh giá chất lượng đầu ra. Trong nhiều hệ thống sinh (generator) hiện đại, thay vì chỉ tạo ra một kết quả duy nhất, mô hình có thể sinh ra một tập các đầu ra tiềm năng, thường gọi là "các ứng viên" $y^{(1)}$, $y^{(2)},...,y^{(N)}$. Mỗi ứng viên là một phiên bản khác nhau của lời đáp, văn bản, hay thông tin mà mô hình đề xuất, dựa trên cùng một đầu vào x. Tuy nhiên, để tận dụng hiệu quả danh sách ứng viên trên, ta cần một cơ chế "tổng hợp" (aggregator) để chọn hoặc kết hợp chúng thành kết quả cuối cùng yyy. Có nhiều phương pháp tổng hợp khác nhau. Một cách đơn giản là ta xây dựng một hàm $h\left(y^{(1)}, y^{(2)},..., y^{(N)}\right)$ (Hình 4) để tính điểm, xếp hạng và chọn ra ứng viên tốt nhất theo tiêu chí mong muốn (ví dụ: ngắn gọn, chính xác, hợp ngữ cảnh). Trong mô hình học

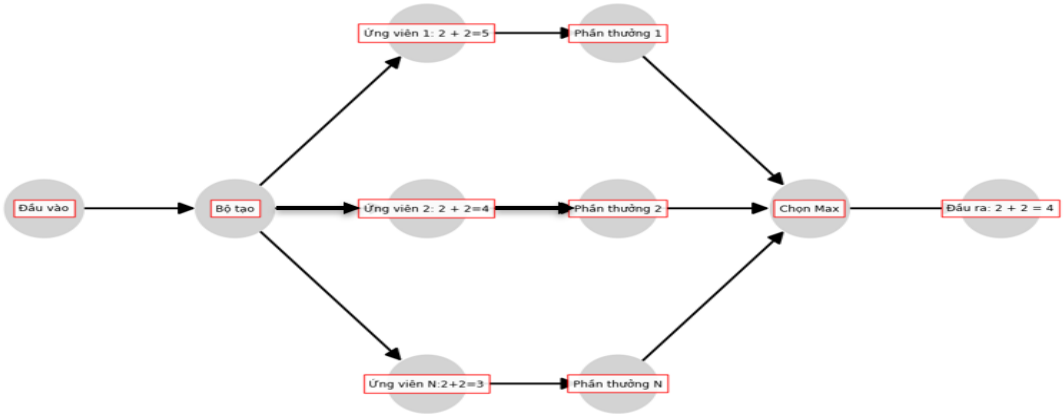
máy, ta có thể dùng một "reward model" (mô hình đánh giá) để gán điểm số cho mỗi ứng viên. Điểm số này có thể phản ánh mức độ đúng về mặt ngữ nghĩa, phong cách phù hợp, hay tuân thủ các quy định chính sách. Khi đó, "aggregator" sẽ lựa chọn ứng viên có điểm số cao nhất, đồng thời loại bỏ hoặc giảm trọng số của các ứng viên kém chất lượng (Hình 5).



Hình 3. *Mối quan hệ giữa số tạo sinh và tỷ lệ giải quyết (%) trong hệ thống đánh giá học được (GSM)*



Hình 4. *Quy trình tạo và tổng hợp ứng viên*

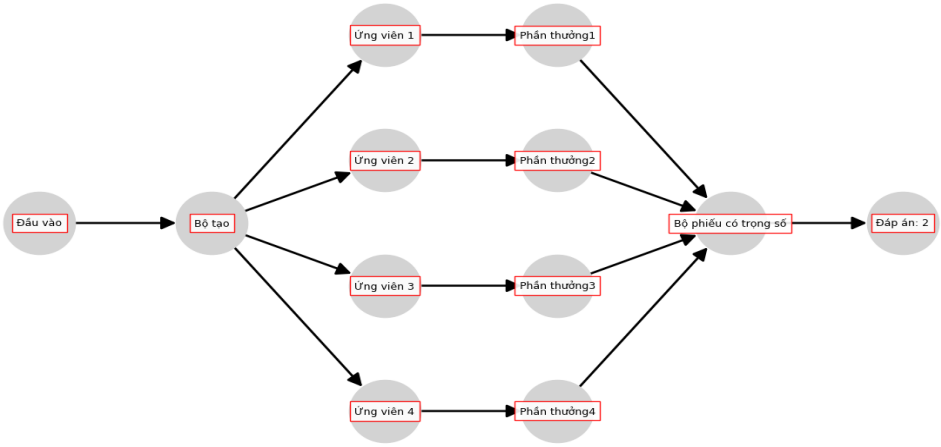


Hình 5. *Quy trình chọn đầu ra tối ưu với mô hình phần thưởng*

Tạo các ứng viên: $\left\{y^{(1)}, y^{(2)}, ..., y^{(N)}\right\} \sim G\left(. \setminus x\right)$. Tổng hợp: $y = h\left(y^{(1)}, y^{(2)}, ..., y^{(N)}\right)$

Ngoài việc chọn ứng viên tốt nhất, aggregator còn có thể "trộn" hoặc "trung bình" nhiều ứng viên khác nhau để tạo ra kết quả hợp nhất. Đây là ý tưởng của các phương pháp "ensemble" trong học máy. Chẳng hạn, nếu có nhiều đầu ra gần giống nhau nhưng mỗi câu lại có điểm mạnh riêng, aggregator có thể kết hợp các đoạn hay để tạo thành câu trả lời hoàn chỉnh. Cách này thường giúp giảm sai sót và mang lại kết quả ổn định hơn so với chỉ dựa vào một lần sinh duy nhất.

Nhờ cơ chế tổng hợp, mô hình có khả năng sửa hoặc hạn chế các lỗi đơn lẻ, đồng thời gia tăng tính linh hoạt trong phản hồi.



Hình 6. *Ví dụ về quy trình bỏ phiếu có trọng số với mô hình phần thưởng*

Biểu thức toán học trong Hình 6 có thể được diễn đạt như sau:

$$\arg\max_a \sum_{i=1}^{N} \underset{reward\ model}{v\left(y^{(i)}\right)} .1\left\{y^{(i)} = a\right\} \quad (4)$$

Weighted Voting (bỏ phiếu theo trọng số) là một kỹ thuật nhằm tổng hợp nhiều đầu ra khác nhau do mô hình sinh tạo ra, trong đó mỗi "ứng viên" $y^{(i)}$ được gán một điểm số (reward) phản ánh chất lượng hoặc mức độ phù hợp. Thay vì chọn đầu ra chỉ dựa trên số lượng phiếu "bầu" như voting thông thường, phương pháp này tính "tổng trọng số" để xác định đáp án tối ưu. Cụ thể, giả sử mô hình sinh đưa ra danh sách $\left\{y^{(1)}, y^{(2)}, ..., y^{(N)}\right\}$ mỗi ứng viên đi kèm một giá trị $v\left(y^{(i)}\right)$ thể hiện độ tin cậy hay độ đúng đắn do "reward model" ước lượng. Khi đó, Weighted Voting sẽ nhóm các ứng viên theo cùng một đáp án rồi tính tổng trọng số của nhóm đó. Đáp án có tổng trọng số cao nhất được chọn làm kết quả cuối cùng. Ưu điểm của phương pháp này nằm ở chỗ nó vừa tận dụng sự đa dạng của các mô hình sinh, vừa giảm ảnh hưởng của những ứng viên kém chính xác (thường có điểm số thấp). Nhờ vậy, Weighted Voting đảm bảo người "nói đúng" hay "mạnh" được ưu tiên hơn, đồng thời ngăn những đáp án sai làm lệch kết quả. Phương pháp này có thể áp dụng trong nhiều bài toán, chẳng hạn giải toán, câu hỏi trắc nghiệm, hoặc sinh văn bản dài, nơi mỗi ứng viên có cách diễn đạt riêng nhưng vẫn cần một cơ chế đánh giá để tìm ra lựa chọn chất lượng nhất.

Về cơ bản, có 4 chiến lược meta-generator khác nhau được sử dụng để tạo hoặc khám phá các lời giải. Mỗi chiến lược mang lại một cách tiếp cận riêng trong việc tổ chức và tinh chỉnh đầu ra, dù đó là văn bản, lời giải bài toán, hay các tác vụ sáng tạo khác.

i) Chiến lược chuỗi (Chain-of-Thought) bao gồm việc sinh ra lời giải theo một quy trình tuyến tính, trong đó mỗi bước được xây dựng dựa trên bước trước đó, tạo thành một chuỗi lập luận liên tục. Phương pháp này đơn giản và giúp duy trì tính mạch lạc, vì mỗi bước sau đều dựa trên thông tin từ bước trước. Tuy nhiên, nó cũng có rủi ro lan tỏa lỗi; một sai sót ở bước đầu có thể ảnh hưởng tiêu cực đến tất cả các bước tiếp theo. Ví dụ, khi giải một bài toán, nếu có lỗi tính toán trong giai đoạn phân tích bài toán ban đầu, tất cả các bước tính toán sau đó và kết quả cuối cùng đều có thể bị sai lệch.

ii) Chiến lược song song tạo ra nhiều đầu ra cùng lúc thay vì dựa trên một chuỗi lập luận duy nhất, cho phép hệ thống so sánh hoặc kết hợp các giải pháp khả thi khác nhau. Phương pháp này tăng cường tính bền vững bằng cách giảm thiểu nguy cơ mắc kẹt trong một hướng sai và cho phép sử dụng cơ chế bỏ phiếu để chọn ra câu trả lời tốt nhất. Tuy nhiên, nó đòi hỏi nhiều tài nguyên tính toán hơn và gây ra sự phức tạp khi tích hợp các đầu ra khác nhau. Ví dụ, trong viết

sáng tạo, hệ thống có thể tạo ra nhiều bản thảo truyện cùng lúc và sau đó lựa chọn bản có mạch truyện chặt chẽ và tính độc đáo cao nhất.

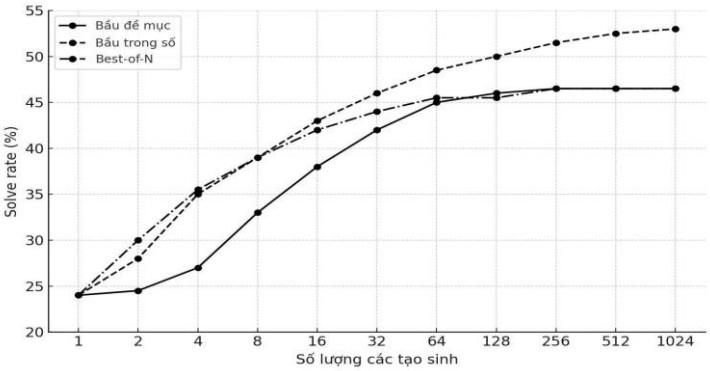
iii) Chiến lược tìm kiếm theo cây tổ chức quá trình giải quyết vấn đề như một cây quyết định, trong đó nhiều nhánh biểu diễn các giải pháp tiềm năng khác nhau được khám phá đồng thời. Phương pháp này cho phép khám phá một cách có hệ thống một không gian lời giải rộng lớn và cung cấp tính linh hoạt khi xem xét nhiều lựa chọn thay thế, nhưng lại đòi hỏi nhiều tài nguyên tính toán và cần các chiến lược sàng lọc tinh vi để loại bỏ những nhánh không hứa hẹn. Ví dụ, trong các hệ thống chơi cờ của AI như cờ vua, một cây các bước đi tiềm năng rất lớn được tạo ra và lược bỏ dần ở mỗi cấp độ, cuối cùng chọn ra bước đi có khả năng thành công cao nhất.

iv) Chiến lược tinh chỉnh bắt đầu với một bản nháp hoặc lời giải sơ bộ và dần dần cải thiện thông qua các vòng lặp liên tiếp. Ở mỗi bước, hệ thống rà soát đầu ra hiện tại, xác định các lỗi hoặc điểm cần cải thiện, và điều chỉnh lời giải cho phù hợp, cho phép tối ưu hóa dần dần theo thời gian. Mặc dù phương pháp này có thể nâng cao đáng kể chất lượng và độ chính xác, nhưng nó có thể tốn thời gian và không luôn đảm bảo hội tụ đến kết quả tối ưu nếu quá trình tinh chỉnh không được định hướng đúng. Ví dụ, trong dịch máy, một bản dịch sơ khởi được liên tục cải tiến bằng cách kiểm tra ngữ pháp, ngữ cảnh và độ tự nhiên cho đến khi đạt được bản dịch cuối cùng mượt mà và chính xác.

Việc lựa chọn chiến lược phụ thuộc vào yêu cầu cụ thể của tác vụ, nguồn lực tính toán sẵn có và mức độ chấp nhận rủi ro do lan tỏa lỗi so với nhu cầu khám phá nhiều hướng giải pháp.

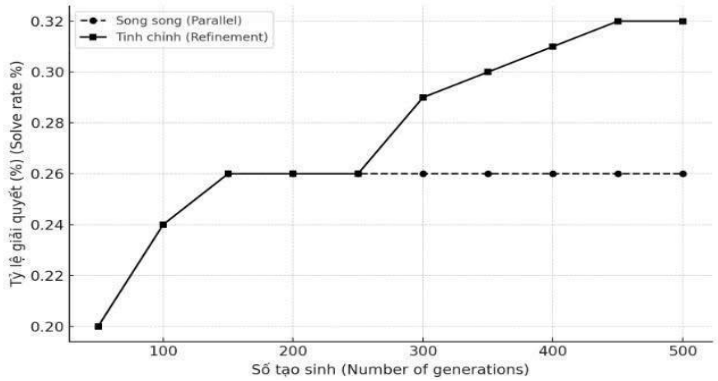
3. Kết quả thực nghiệm

Các kết quả thực nghiệm được chạy trên máy chủ với GPU NVIDIA A100 đặt tại Viện Công nghệ Thông tin – Viện Hàn lâm Khoa học và Công nghệ Việt Nam. Dưới đây chúng tôi trình bày các kết quả về hiệu suất của các phương pháp lựa chọn ứng viên tạo sinh.



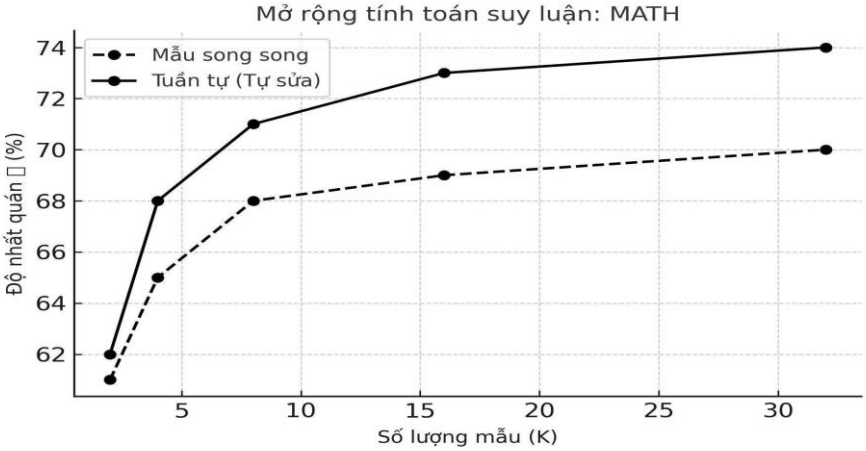
Hình 7. *So sánh tỉ lệ giải đúng cho ba phương pháp Bầu đề mục, Bầu trọng số và Best-of-N*

Hình 7 so sánh tỉ lệ giải đúng, được biểu thị bằng phần trăm, cho ba phương pháp khác nhau - "Bầu đề mục," "Bầu trọng số" và "Best-of-N" - với các kích cỡ mẫu khác nhau từ 1 đến 1024. Trục ngang hiển thị số lượng mẫu theo thang logarit, trong khi trục dọc biểu diễn tỉ lệ giải đúng. Ta nhận thấy cả ba phương pháp đều có xu hướng tăng khi số lượng mẫu tăng. Ban đầu, khi số lượng mẫu còn ít, "Bầu đề mục" cho hiệu suất thấp nhất, bắt đầu khoảng 24%, trong khi "Bầu trọng số" cao hơn một chút và Best-of-N nổi lên như phương pháp hiệu quả nhất ngay từ đầu. Tuy nhiên, khi số lượng mẫu tăng đến mức trung bình, hai đường đỏ và xanh dương dần thu hẹp khoảng cách với đường xanh lá, dù Best-of-N vẫn luôn dẫn đầu. Đến khi số lượng mẫu đạt mức cao nhất là 1024, cả ba phương pháp đều hội tụ quanh mức tỉ lệ giải đúng gần 50% với "Bầu đề mục" thấp hơn một chút, "Bầu trọng số" ở mức trung và Best-of-N vẫn giữ vị trí cao nhất. Điều này cho thấy việc tăng số lượng mẫu có thể cải thiện đáng kể tỉ lệ giải đúng cho mọi phương pháp, nhưng Best-of-N vẫn vượt trội hơn. Cuối cùng, kết quả cũng nhấn mạnh tầm quan trọng của kích cỡ mẫu và chiến lược lựa chọn trong việc tối ưu hóa kết quả.



Hình 8. *So sánh hiệu suất của hai phương pháp song song và tinh chỉnh*

Hình 8 so sánh hiệu suất của hai phương pháp song song và tinh chỉnh khi tăng dần số lượng thế hệ đem lại những thông tin giá trị về cách mỗi phương pháp mở rộng quy mô với nhiều vòng lặp bổ sung. Phương pháp song song, được biểu thị bằng đường màu xanh lá, bắt đầu với tỉ lệ giải quyết gần 26% và hầu như không thay đổi trong suốt dải quan sát. Ngược lại, phương pháp tinh chỉnh, thể hiện bằng đường màu cam, khởi đầu ở khoảng 20%, tức thấp hơn song song, nhưng lại tăng dần và liên tục khi số tạo sinh tăng. Đáng chú ý, vào khoảng 120 đến 200 thế hệ, đường tinh chỉnh vượt qua đường song song, phản ánh rằng quá trình tinh chỉnh lặp đi lặp lại mang lại những cải thiện đáng kể về tỉ lệ giải quyết. Khi số thế hệ đạt khoảng bốn đến năm trăm, phương pháp tinh chỉnh ổn định ở mức xấp xỉ ba mươi phần trăm, vượt xa phương pháp song song. Sự chênh lệch này cho thấy, mặc dù song song có thể là lựa chọn hấp dẫn cho số tạo sinh nhỏ nhờ tỉ lệ giải quyết ban đầu cao, nó lại thiếu khả năng cải thiện dài hạn. Trong khi đó, tinh chỉnh, dù ban đầu yếu hơn, vẫn cải thiện đều đặn qua mỗi thế hệ bổ sung, khiến nó trở nên hiệu quả hơn khi tài nguyên tính toán cho phép chạy nhiều vòng. Nhìn chung, so sánh này nhấn mạnh tầm quan trọng của việc lựa chọn phương pháp dựa trên nguồn lực và mục tiêu hiệu suất với tinh chỉnh là chiến lược hứa hẹn hơn để tối đa hóa tỉ lệ giải quyết khi số thế hệ lớn.



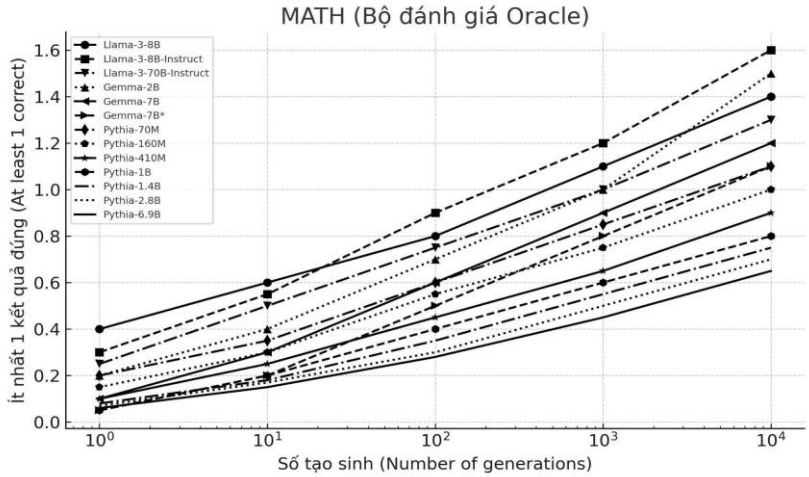
Hình 9. *So sánh tỉ lệ nhất quán của hai phương pháp song song và tuần tự*

Hình 9 so sánh "Độ nhất quán @K" cho hai phương pháp - "Mẫu song song" (đường nét đứt) và "Tuần tự (Tự sửa)" (đường nét liền) - khi số lượng mẫu (K) tăng từ khoảng 2 đến 30. Trục ngang biểu thị giá trị K, trong khi trục dọc cho thấy tỉ lệ nhất quán, dao động từ khoảng 60% đến 75%. Xu hướng tổng quát là cả hai phương pháp đều cải thiện độ nhất quán khi K tăng, phản ánh rằng càng có nhiều mẫu thì khả năng đạt được kết quả chính xác hoặc nhất quán càng cao. Quan sát nét đứt ("Mẫu song song"), nó bắt đầu ở mức khoảng 60% khi K còn nhỏ (chừng 2 mẫu) và tăng dần đến khoảng 74% khi

K đạt 30. Tốc độ cải thiện khá nhanh trong giai đoạn K = 2 đến K = 8, cho thấy việc bổ sung thêm một vài mẫu ban đầu có thể mang lại lợi ích đáng kể. Sau K = 16, đường này tiếp tục tăng nhưng chậm dần, cho thấy lợi ích biên giảm khi số mẫu ngày càng nhiều.

Trong khi đó, đường nét liền ("Tuần tự (Tự sửa)") bắt đầu khoảng 62% tại K = 2 và duy trì tỉ lệ giải quyết nhỉnh hơn so với phương pháp song song ở mọi kích thước mẫu. Đường này cũng tăng nhanh từ K = 2 đến K = 8, vượt mức 70% khi đến khoảng giữa dải K. Đến K = 30, nó ổn định quanh mức 75%. Mặc dù khoảng cách giữa hai phương pháp thu hẹp lại khi K cao hơn, "Tuần tự (Tự sửa)" vẫn luôn dẫn đầu với chênh lệch từ một đến ba điểm phần trăm.

Nhìn chung, biểu đồ cho thấy cả hai phương pháp "Mẫu song song" và "Tuần tự (Tự sửa)" đều được hưởng lợi từ việc tăng K, nhưng phương pháp tuần tự tỏ ra hiệu quả hơn trong việc tận dụng thêm số mẫu. Về mặt ứng dụng, điều này gợi ý rằng một chiến lược tự sửa (lặp đi lặp lại) có thể mang lại ưu thế nhỏ so với phương pháp song song thuần túy, nhất là khi cần độ chính xác hoặc nhất quán cao hơn và có đủ tài nguyên để chạy số lượng mẫu lớn.



Hình 10. *Đo lường xác suất thu được ít nhất một lời giải đúng của các mô hình ngôn ngữ khác nhau*

Hình 10 so sánh nhiều mô hình ngôn ngữ khác nhau với quy mô đa dạng (ví dụ: Llama-3.8B, Llama-3B-Instruct, Gemma-2.7B, Pythia-70M, Pythia-1.4B, Pythia-12B, v.v.) trên bộ đánh giá MATH, đo lường xác suất thu được *ít nhất một lời giải đúng* khi số lượng thế hệ (từ 1 đến 10.000) tăng dần. Trục hoành biểu thị số lượng thế hệ theo thang log (1, 10, 100, 1.000, 10.000), trong khi trục tung biểu thị tỉ lệ bài thử nghiệm mà có ít nhất một kết quả đúng. Tất cả các đường đều đi lên, cho thấy việc tạo nhiều đáp án hơn (nhiều "mẫu" hoặc "thế hệ" hơn) sẽ làm tăng khả năng tìm ra ít nhất một đáp án chính xác. Đây là điều hợp lý vì số lần thử lớn hơn sẽ nâng xác suất có đáp án đúng. Nhìn chung, các mô hình lớn (chẳng hạn Llama-3B-Instruct hay Pythia-12B) bắt đầu với tỉ lệ thành công cao hơn và đạt đỉnh cao hơn, phản ánh khả năng cơ bản mạnh mẽ và tốt hơn khi mở rộng số lần sinh. Các phiên bản đã được tinh chỉnh theo hướng dẫn (chẳng hạn "Instruct") thường thể hiện tốt hơn so với phiên bản gốc có kích thước tương đương. Điều này cho thấy việc tinh chỉnh mô hình giúp cải thiện độ chính xác. Những mô hình nhỏ như Pythia-70M hoặc Gemma-2.7B bắt đầu với tỉ lệ thành công thấp hơn và tăng chậm hơn. Mặc dù chúng cũng tiến bộ khi số thế hệ tăng, nhưng không bắt kịp mô hình lớn nhất ở mức 10.000 thế hệ.

Khi số lượng thế hệ vượt mốc 100, 1.000 và tiến tới 10.000, khoảng cách giữa các mô hình mạnh và yếu trở nên rõ rệt hơn. Đa số mô hình có dấu hiệu giảm tốc độ cải thiện khi tiến tới ngưỡng rất cao (10.000 thế hệ), nhưng các mô hình lớn vẫn giữ mức hiệu suất tổng thể cao hơn.

4. Kết luận

Hiệu suất của mô hình sinh thường tăng khi tài nguyên tính toán được cải thiện, nhưng mức độ gia tăng này còn phụ thuộc vào kích thước mô hình và chiến lược sử dụng meta-generator.

Kích thước mô hình tối ưu và chiến lược tốt nhất có thể thay đổi theo ngân sách tính toán, trong đó có những trường hợp mô hình nhỏ hơn lại mang lại hiệu suất cao hơn. Mục tiêu cuối cùng là thiết kế các chiến lược có tính tối ưu tổng quát, phù hợp với nhiều điều kiện tính toán khác nhau. Meta-generator đóng vai trò như một chiến lược điều phối các mô hình sinh và kết hợp thông tin từ nhiều nguồn, với nhiều mô hình tổ chức khác nhau như dạng chuỗi, song song, tìm kiếm theo cây và tinh chỉnh dần dần. Những phương pháp này có thể kết hợp và sử dụng đồng thời để tối ưu hóa hiệu suất theo yêu cầu của nhiệm vụ và chi phí tính toán. Trong bước tiếp theo, các meta-generator sẽ được mở rộng để sinh ra nhiều token hơn theo những cách tiếp cận đa dạng như tìm kiếm theo cây. Thách thức quan trọng là làm sao để thực hiện điều này một cách nhanh chóng và hiệu quả, cân bằng giữa tốc độ xử lý và chất lượng đầu ra.

TÀI LIỆU THAM KHẢO/REFERENCES

[1] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *arXiv preprint arXiv:2101.03961*, 2022.

[2] G. Feng, B. Zhang, Y. Gu, H. Ye, D. He, and L. Wang, "Towards revealing the mystery behind chain of thought: a theoretical perspective," *Advances in Neural Information Processing Systems*, vol. 36, pp. 70757–70798, 2023.

[3] M. Finlayson, J. Hewitt, A. Koller, S. Swayamdipta, and A. Sabharwal, "Closing the curious case of neural text degeneration," *arXiv preprint arXiv:2310.01693*, 2023.

[4] M. Hobbahn, L. Heim, and G. Aydos, "Trends in machine learning hardware," Epoch AI, Nov. 9, 2023. [Online]. Available: https://epoch.ai. [Accessed Nov. 26, 2024].

[5] E. Tulchinskii, K. Kuznetsov, L. Kushnareva, D. Cherniavskii, S. Nikolenko, E. Burnaev, *et al*., "Intrinsic dimension estimation for robust detection of AI-generated texts," *Advances in Neural Information Processing Systems*, vol. 36, pp. 39257–39276, 2023.

[6] M. Li, W. Wang, F. Feng, F. Zhu, Q. Wang, and T. S. Chua, "Think twice before trusting: Self-detection for large language models through comprehensive answer reflection," *arXiv preprint arXiv:2403.09972*, 2024.

[7] Y. He, J. Zhang, J. Bao, F. Lin, C. Yang, B. Qin, *et al*., "BC-Prover: Backward Chaining Prover for Formal Theorem Proving," in *Proc. 2024 Conf. Empirical Methods in Natural Language Processing (EMNLP)*, Miami, Florida, Nov. 2024, pp. 3059–3077.

[8] D. Jurafsky, B. Brown, R. Ehrlich, D. Y. Fu, C. Ré, and A. Mirhoseini, "Hydragen: High-throughput LLM inference with shared prefixes," *arXiv preprint arXiv:2402.03467*, 2024.

[9] L. Wang, S. Chen, L. Jiang, S. Pan, R. Cai, S. Yang, and F. Yang, "Parameter-efficient fine-tuning in large language models: A survey of methodologies," *Artificial Intelligence Review*, vol. 58, no. 8, 2025, Art. no. 227.

[10] B. Weng, "Navigating the landscape of large language models: A comprehensive review and analysis of paradigms and fine-tuning strategies," *arXiv preprint arXiv:2404.09022*, 2024.

[11] A. D. Cohen, A. Roberts, A. Molina, A. Butryna, A. Jin, A. Kulshreshtha, *et al*., "LaMDA: Language models for dialog applications," *arXiv preprint arXiv:2201.08239*, 2022.

[12] T. Kaufmann, P. Weng, V. Bengs, and E. Hüllermeier, "A survey of reinforcement learning from human feedback," *arXiv preprint arXiv:2312.14925*, 2023.

[13] S. Welleck, A. Bertsch, M. Finlayson, *et al.*, "From Decoding to Meta-Generation: Inference-time Algorithms for Large Language Models," *arXiv preprint arXiv:2406.16838v2***,** 2024.