AI VIET NAM
@aivietnam.edu.vn

AI VIET NAM – AI COURSE 2025

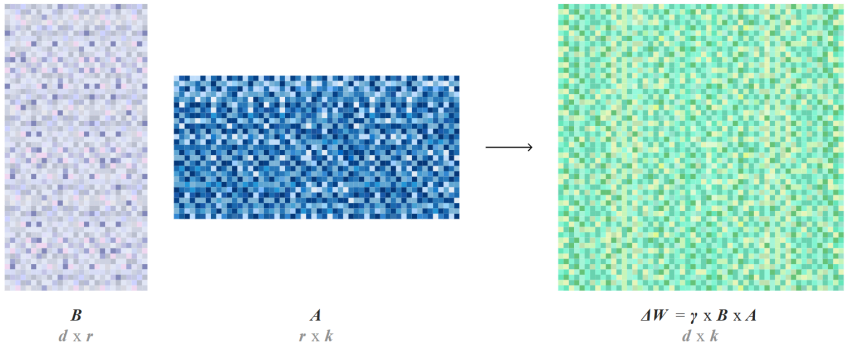
Tutorial: LoRA Without Regret in Fine-tuning a LLM

Trần Hoàng Duy Dương Đình Thắng Đinh Quang Vinh

I. Giới thiệu

Sự phát triển của các mô hình ngôn ngữ lớn đã kéo theo một thay đổi đáng kể trong cách ta huấn luyện sau tiền huấn luyện. Nếu pretraining là giai đoạn hấp thụ kiến thức tổng quát từ dữ liệu cực lớn, thì post-training thường nhắm vào việc điều chỉnh hành vi của mô hình trong những phạm vi hẹp hơn, với dữ liệu ít hơn và mục tiêu rõ hơn. Vì vậy, các kỹ thuật tinh chỉnh hiệu quả tham số trở nên quan trọng, vì chúng tận dụng mô hình nền sẵn có mà không phải gánh toàn bộ chi phí của fine-tune toàn tham số.

Trong nhóm PEFT, LoRA được dùng rộng rãi vì triển khai gọn, dễ mở rộng và thuận tiện khi cần nhiều biến thể mô hình. Tuy nhiên, câu hỏi then chốt vẫn là: trong supervised fine-tuning, LoRA có thể tiệm cận FullFT đến mức nào, và các lựa chọn như learning rate, batch size hay vị trí gắn adapter ảnh hưởng ra sao.



Hình 1: Minh họa cơ chế cốt lõi của LoRA: Sử dụng các ma trận hạng thấp nhỏ (B và A) để biểu diễn sự thay đổi trọng số lớn (ΔW).

Bài viết LoRA Without Regret của Thinking Machines Lab đặt trọng tâm trực tiếp vào câu hỏi này và đề xuất một khung thực nghiệm tương đối chặt: theo dõi bằng test negative log-likelihood thay vì sampling, quét learning rate để tránh lệch do chọn LR, và khảo sát có hệ thống các yếu tố như rank, batch size và vị trí đặt LoRA trong mô hình. Dựa trên tinh thần đó, bài viết này chạy lại các thực nghiệm chính và đối chiếu LoRA với FullFT trong điều kiện tương đương, tập trung vào cách đọc đường cong học và thiết lập thực nghiệm để làm rõ khi nào LoRA có thể tiệm cận FullFT và khi nào khoảng cách bắt đầu xuất hiện do cấu hình hoặc siêu tham số.

Mục lục

Nhằm hỗ trợ việc đọc xuyên suốt bài viết, bảng chú thích dưới đây tóm tắt các thuật ngữ chính được dùng trong bài.

Bảng chú thích thuật ngữ

Thuật ngữ	Giải thích
Pretraining	Giai đoạn huấn luyện trên dữ liệu cực lớn để mô hình học tri thức tổng quát.
Post-training	Giai đoạn huấn luyện sau pretraining nhằm điều chỉnh hành vi mô hình trong phạm vi hẹp hơn với dữ liệu ít hơn.
Fine-tuning	Tinh chỉnh mô hình trên dữ liệu tác vụ/miền cụ thể.
FullFT (Full Fine-Tuning)	Fine-tune toàn bộ tham số của mô hình.
PEFT (Parameter-Efficient Fine-Tuning)	Nhóm phương pháp tinh chỉnh hiệu quả tham số: đóng băng mô hình nền và chỉ học một phần tham số bổ sung nhỏ.
LoRA (Low-rank Adaptation)	Phương pháp PEFT biểu diễn cập nhật trọng số dưới dạng hạng thấp $\Delta\mathbf{W} = \mathbf{BA}$.
Adapter	Thành phần tham số bổ sung giúp mô hình thích nghi; trong ngữ cảnh này chính là nhánh LoRA được học.
SFT (Supervised Fine-Tuning)	Tinh chỉnh có giám sát bằng dữ liệu cặp (đầu vào, mục tiêu).
Test NLL / NLL	Negative log-likelihood trên tập test; thước đo chính để theo dõi động lực học huấn luyện trong thí nghiệm.
Step	Bước tối ưu/huấn luyện (thường tương ứng một lần cập nhật tham số).
Epoch	Một vòng lặp đi qua toàn bộ dữ liệu huấn luyện.
Learning rate (LR)	Tốc độ học; siêu tham số điều khiển độ lớn bước cập nhật của tối ưu hoá.
LR sweep	Quét nhiều giá trị learning rate để tránh kết luận bị lệch do chọn LR chưa tối ưu.
Best-of-sweep (theo step)	Khi vẽ đường cong theo step, tại mỗi step chọn LR cho NLL nhỏ nhất để tạo "cận dưới thực nghiệm".
Rank (r)	Siêu tham số quyết định số bậc tự do của LoRA; rank cao thường tiệm cận FullFT tốt hơn trong SFT.

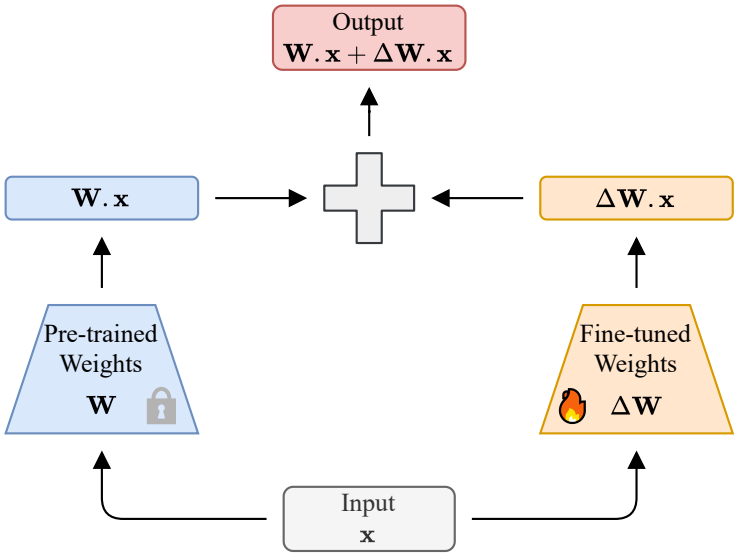
Để việc đọc thuận tiện hơn, bảng sau liệt kê và chuẩn hoá các ký hiệu sẽ xuất hiện xuyên suốt nội dung bài.

Bảng ký hiệu toán học

Ký hiệu	Ý nghĩa
$\mathbf{W} \in \mathbb{R}^{d\times k}$	Ma trận trọng số gốc của một lớp tuyến tính.
$\mathbf{W}'$	Trọng số hiệu dụng sau khi gắn LoRA.
$\Delta\mathbf{W}$	Phần cập nhật trọng số được học trong quá trình tinh chỉnh.
$\mathbf{B} \in \mathbb{R}^{d\times r}$	Ma trận LoRA "up-projection" (từ không gian rank r lên d).
$\mathbf{A} \in \mathbb{R}^{r\times k}$	Ma trận LoRA "down-projection" (từ k xuống rank r).
r	Rank của LoRA (hạng thấp), thường $r \ll \min(d, k)$.
α	Hệ số scale của LoRA (thường dùng dưới dạng α/r).
d	Số chiều đầu ra của lớp tuyến tính (số hàng của $\mathbf{W}$).
k	Số chiều đầu vào của lớp tuyến tính (số cột của $\mathbf{W}$).
$\mathbf{x} \in \mathbb{R}^k$	Vector đầu vào của lớp tuyến tính.
$\mathbf{u} = \mathbf{A}\mathbf{x} \in \mathbb{R}^r$	Biến trung gian trong nhánh LoRA.
$\mathbf{h}$	Đầu ra lớp tuyến tính sau khi cộng nhánh LoRA.
dk	Số tham số của $\mathbf{W}$ (nếu fine-tune toàn bộ ở mức lớp).
$r(d + k)$	Số tham số LoRA cần học ở mức lớp (xấp xỉ).
$\mathbf{W}' = \mathbf{W} + \Delta\mathbf{W}$	Dạng tổng quát: trọng số sau cập nhật.
$\Delta\mathbf{W} = \mathbf{B}\mathbf{A}$	Ràng buộc hạng thấp trong LoRA.
$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{B}\mathbf{A}\mathbf{x}$	Đầu ra gồm nhánh gốc và nhánh LoRA cộng lại.
$\mathbf{W}' = \mathbf{W} + \left(\frac{\alpha}{r}\right)\mathbf{B}\mathbf{A}$	Dạng có hệ số scale để điều khiển biên độ cập nhật.

II. Low-rank Adaptation

Khi các mô hình ngôn ngữ lớn tăng lên đến hàng tỉ hoặc hàng chục tỉ tham số, fine-tune toàn tham số dần trở thành một lựa chọn đắt đỏ theo nghĩa rất thực dụng. Chi phí không chỉ nằm ở việc lưu trọng số của mô hình, mà nằm ở bộ nhớ huấn luyện: ngoài trọng số, ta còn phải lưu gradient và trạng thái của bộ tối ưu (ví dụ các moment của Adam) cho toàn bộ tham số, thường ở độ chính xác cao hơn so với trọng số dùng khi suy luận. Vì vậy, ngay cả khi một mô hình có thể chạy inference ổn trên một cấu hình GPU nhất định, việc fine-tune toàn bộ thường đòi hỏi cấu hình lớn hơn đáng kể. Trong bối cảnh post-training, dữ liệu tinh chỉnh thường nhỏ hơn rất nhiều so với dữ liệu pretraining và tập trung vào một phạm vi hành vi hẹp hơn, nên việc cập nhật toàn bộ tham số còn kéo theo rủi ro overfitting, đồng thời gây khó khăn khi triển khai nhiều biến thể mô hình theo tác vụ: mỗi phiên bản fine-tune phải lưu trữ một bản sao gần như đầy đủ của mô hình gốc.



Hình 2: Minh hoạ cấu trúc LoRA trong một lớp tuyến tính. Trọng số gốc **W** được giữ cố định, phần cập nhật được biểu diễn dưới dạng tích **BA**, và đầu ra là tổng của hai nhánh **Wx** và **BAx**.

Những nút thắt đó dẫn tới một hướng tiếp cận là thay vì cập nhật toàn bộ mô hình, ta giữ nguyên phần lớn trọng số đã học từ pretraining và chỉ cho phép mô hình thay đổi thông qua một phần tham số nhỏ, đủ để "bẻ" hành vi theo tác vụ mới. Đây là tinh thần của Parameter Efficient Fine-Tuning (PEFT). Điểm chung của các phương pháp PEFT là đóng băng mô hình nền và thêm vào một cơ chế thích nghi có kích thước nhỏ để học trên dữ liệu tinh chỉnh. Khi số tham số được cập nhật giảm, bộ nhớ huấn luyện giảm theo vì chỉ cần gradient và optimizer states cho phần tham số bổ sung; việc lưu trữ và phân phối mô hình cũng nhẹ hơn vì có thể lưu phần thích nghi thay vì toàn bộ trọng số. PEFT không phải một kỹ thuật đơn lẻ mà là một nhóm kỹ thuật, trong đó LoRA là một đại diện rất phổ biến vì cấu trúc đơn giản, dễ gắn vào các lớp tuyến tính của Transformer, và thường đạt hiệu quả tốt nếu cấu hình hợp lý.

LoRA xuất phát từ một quan sát mang tính cấu trúc: trong một lớp tuyến tính, fine-tune toàn

tham số tương đương với việc cho phép ma trận trọng số thay đổi tự do trong không gian rất lớn, nhưng phần cập nhật thực sự cần thiết để thích nghi theo tác vụ có thể chỉ nằm trong một không gian con có số chiều nhỏ hơn nhiều. Từ đó, LoRA biểu diễn cập nhật của một ma trận trọng số bằng một thành phần hạng thấp. Xét một lớp tuyến tính với ma trận trọng số gốc

$$\mathbf{W} \in \mathbb{R}^{d\times k}.$$

Fine-tune toàn tham số cập nhật trực tiếp $\mathbf{W}$. LoRA đóng băng $\mathbf{W}$ và chỉ học một phần cập nhật $\Delta\mathbf{W}$, sau đó cộng vào khi chạy mô hình:

$$\mathbf{W}' = \mathbf{W} + \Delta\mathbf{W}.$$

Trong đó:

- $\mathbf{W} \in \mathbb{R}^{d\times k}$: ma trận trọng số gốc.
- $\mathbf{W}'$: trọng số hiệu dụng sau khi gắn LoRA (dùng để tính forward).
- $\Delta\mathbf{W}$: phần cập nhật trọng số được học trong quá trình tinh chỉnh.

Thay vì để $\Delta\mathbf{W}$ tự do (kích thước $d \times k$), LoRA ép $\Delta\mathbf{W}$ có dạng tích của hai ma trận mỏng:

$$\Delta\mathbf{W} = \mathbf{BA}, \qquad \mathbf{B} \in \mathbb{R}^{d\times r}, \quad \mathbf{A} \in \mathbb{R}^{r\times k}, \quad r \ll \min(d, k).$$

Trong đó:

- $\mathbf{B} \in \mathbb{R}^{d\times r}$: ma trận LoRA up-projection (từ không gian rank r lên d).
- $\mathbf{A} \in \mathbb{R}^{r\times k}$: ma trận LoRA down-projection (từ k xuống rank r).
- r: rank của LoRA (hạng thấp), thường $r \ll \min(d, k)$.
- α: hệ số scale của LoRA (thường dùng dưới dạng α/r để điều khiển biên độ cập nhật).
- d: số chiều đầu ra của lớp tuyến tính (tương ứng số hàng của $\mathbf{W}$).
- k: số chiều đầu vào của lớp tuyến tính (tương ứng số cột của $\mathbf{W}$).

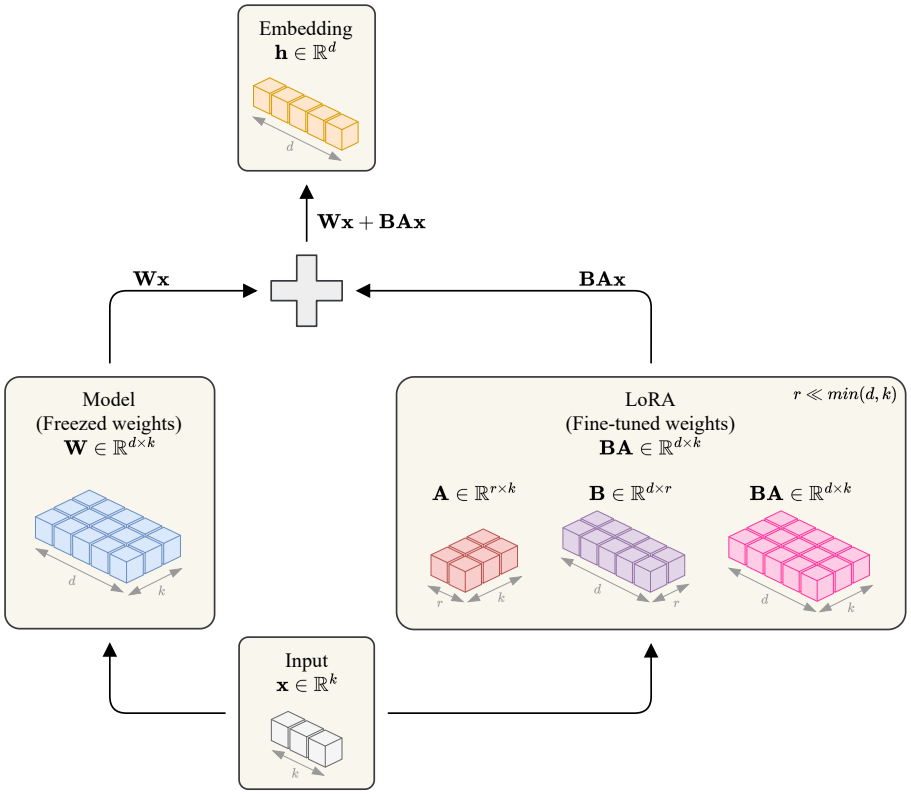
Khi đó, số tham số cần học giảm từ dk xuống $r(d+k)$. Với r nhỏ, đây là một mức giảm rất lớn, đặc biệt trong các lớp tuyến tính có kích thước lớn trong Transformer. Đồng thời, về mặt tính toán, cập nhật LoRA đi qua hai phép nhân ma trận–vector nhỏ hơn: thay vì một phép nhân $d \times k$, nhánh LoRA có thể được tính theo $\mathbf{u} = \mathbf{Ax} \in \mathbb{R}^r$ rồi $\mathbf{Bu} \in \mathbb{R}^d$, nên chi phí của nhánh thích nghi chủ yếu tỷ lệ theo r.

Ở mức biểu thức, với đầu vào $\mathbf{x} \in \mathbb{R}^k$, đầu ra lớp tuyến tính có LoRA là tổng của hai nhánh:

$$\mathbf{h} = \mathbf{Wx} + \mathbf{BAx},$$

trong đó nhánh $\mathbf{Wx}$ dùng trọng số gốc (được giữ cố định), còn nhánh $\mathbf{BAx}$ là phần thích nghi được học trong quá trình tinh chỉnh. Hình 3 minh hoạ trực quan đúng cấu trúc này: mô hình

nền giữ nguyên trọng số, LoRA bổ sung một nhánh cập nhật hạng thấp, và hai nhánh được cộng để tạo embedding đầu ra.



Hình 3: Minh hoạ cấu trúc LoRA trong một lớp tuyến tính. Trọng số gốc $\mathbf{W}$ được giữ cố định, phần cập nhật được biểu diễn dưới dạng tích $\mathbf{BA}$, và đầu ra là tổng của hai nhánh $\mathbf{Wx}$ và $\mathbf{BAx}$.

Một chi tiết thiết kế quan trọng của LoRA là cách khởi tạo để mô hình ban đầu hành xử gần giống mô hình gốc. Thực hành phổ biến là khởi tạo một trong hai ma trận sao cho $\Delta\mathbf{W}$ ban đầu xấp xỉ 0, thường là đặt $\mathbf{B} = \mathbf{0}$ và khởi tạo $\mathbf{A}$ ngẫu nhiên với biên độ nhỏ. Khi đó, ở thời điểm bắt đầu huấn luyện, $\mathbf{BA} \approx \mathbf{0}$ nên $\mathbf{W}' \approx \mathbf{W}$; mô hình chỉ dần thay đổi khi $\mathbf{B}$ và $\mathbf{A}$ học được một cập nhật có ý nghĩa từ dữ liệu tinh chỉnh.

Trong thực tế, LoRA thường thêm một hệ số tỉ lệ để kiểm soát biên độ cập nhật và giúp tối ưu ổn định hơn:

$$\mathbf{W}' = \mathbf{W} + \left(\frac{\alpha}{r}\right)\mathbf{BA}.$$

Ở đây, r quyết định số bậc tự do của phần cập nhật: r lớn hơn cho phép biểu diễn linh hoạt hơn nhưng tăng chi phí bộ nhớ và số tham số học được; r nhỏ hơn tiết kiệm hơn nhưng có thể hạn chế khả năng thích nghi nếu cập nhật cần thiết không nằm gọn trong không gian con kích thước r. Hệ số α điều chỉnh độ mạnh của nhánh LoRA so với nhánh gốc; thay vì chỉ nói α "lớn hay nhỏ", cách hiểu trực tiếp hơn là α/r đóng vai trò như một hệ số khuếch đại cho adapter, quyết định quy mô cập nhật mà mô hình cảm nhận được trong quá trình huấn luyện.

Cuối cùng, LoRA không chỉ là một công thức cho một lớp tuyến tính, mà là một lựa chọn triển khai trong toàn bộ Transformer: ta quyết định sẽ gắn LoRA vào những ma trận nào. Các vị

trí thường gặp là các phép chiếu tuyến tính trong attention và các lớp tuyến tính trong khối feed-forward. Việc lựa chọn vị trí gắn LoRA quyết định nơi mô hình được phép thay đổi hành vi, do đó ảnh hưởng trực tiếp đến chất lượng, tốc độ học và độ ổn định. Chính vì vậy, khi chuyển sang LoRA Without Regret, câu hỏi không còn dừng ở việc LoRA có tiết kiệm hay không, mà là LoRA có thể tiến gần đến fine-tune toàn tham số đến mức nào, và cần những gì để đạt được điều đó trong các bài toán post-training.

III. LoRA Without Regret

Bài viết LoRA Without Regret (Thinking Machines Lab) đi thẳng vào câu hỏi đó bằng một thiết kế thực nghiệm có chủ đích. Thay vì đánh giá dựa trên sampling (dễ nhiễu bởi decoding, seed, ...), tác giả dùng test negative log-likelihood (test NLL) để theo dõi trực tiếp động lực học của quá trình huấn luyện. Đồng thời, để tránh kết luận bị lệch do chọn learning rate chưa hợp lý, mỗi cấu hình đều được quét learning rate, và khi vẽ đường cong theo step, mỗi đường rank được lấy theo kiểu best-of-sweep: tại mỗi step, chọn learning rate cho ra NLL nhỏ nhất. Cách đọc này rất quan trọng, vì nó biến các đường cong thành một dạng cận dưới thực nghiệm: nếu LoRA vẫn kém trong điều kiện đã quét LR, thì khoảng cách đó phản ánh khác biệt thực sự của cấu hình (rank, batch size, vị trí đặt LoRA), chứ không phải lỗi tinh chỉnh.

Các thực nghiệm SFT trong bài chủ yếu dựa trên hai bộ dữ liệu đại diện cho post-training: Tulu3 (thiên về instruction tuning) và OpenThoughts3 (thiên về reasoning). Chúng khác nhau đáng kể về cấu trúc và phạm vi, nên nếu cùng một xu hướng xuất hiện ổn định trên cả hai, ta có cơ sở mạnh hơn để xem đó là quy luật chứ không phải hiện tượng cục bộ.

III.1. Finding 1: Quan hệ giữa rank và đường cong học trong SFT

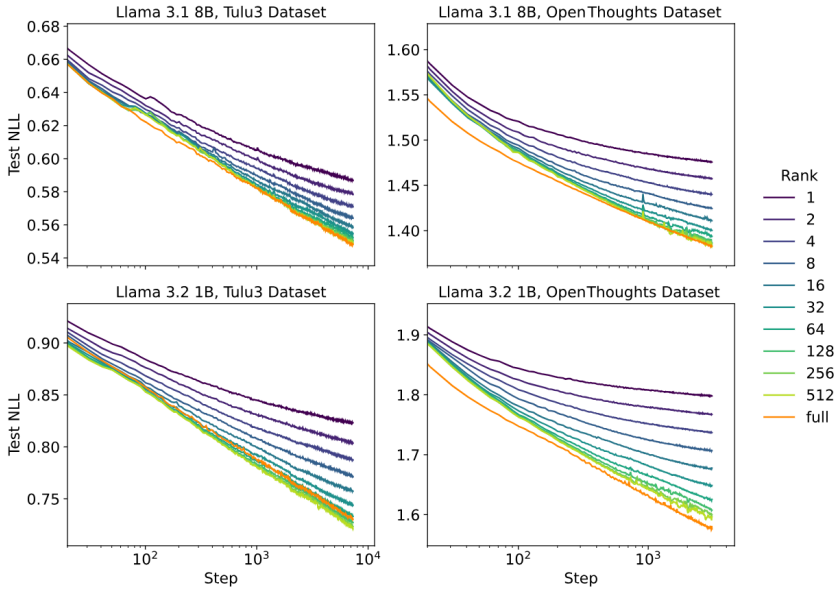
Hình **??** là bức tranh tổng quan rõ nhất về tác động của rank lên động lực học khi làm SFT. Hình gồm bốn ô: hai kích thước mô hình (8B và 1B) kết hợp với hai bộ dữ liệu (Tulu3 và OpenThoughts3). Mỗi ô có nhiều đường tương ứng các rank khác nhau, và một đường tham chiếu cho FullFT. Điểm đáng chú ý không chỉ nằm ở NLL cuối, mà nằm ở cả hình dạng và độ bám của đường cong theo step.

Trước hết, trong miền rank cao, nhiều đường LoRA nằm rất gần đường FullFT trong phần lớn quá trình huấn luyện. Trên cả Tulu3 và OpenThoughts3, ta nhìn thấy loss giảm đều theo log(step), và các rank lớn bám theo quỹ đạo giảm đó. Điều này cho thấy, ít nhất ở những cấu hình rank đủ lớn, LoRA có thể tái hiện gần như đầy đủ động lực học của FullFT trong SFT: cùng số bước, giảm loss tương tự, và mức loss cuối tiệm cận nhau.

Tiếp theo, khi rank giảm, các đường cong bắt đầu tách ra theo một cách có cấu trúc. Ở nhiều ô, có thể quan sát một ngưỡng step mà sau đó các rank trung bình và thấp rời khỏi nhóm đường tốt nhất và tạo ra khoảng cách ngày càng rõ rệt so với FullFT. Nghĩa là, ngay cả khi ta luôn chọn learning rate tối ưu cho từng rank, vẫn tồn tại những rank mà LoRA không thể đi cùng quỹ đạo với FullFT cho tới hết một epoch. Điểm này quan trọng vì nó cho ta một cách nhìn

thực dụng về rank: rank không chỉ là tham số quyết định số lượng tham số trainable, mà còn liên quan trực tiếp tới việc LoRA có theo kịp tiến trình giảm loss trong khoảng bước huấn luyện mong muốn hay không.

Cuối cùng, Hình **??** cũng cho thấy sự khác biệt theo dataset và theo kích thước mô hình. Có những trường hợp rank cao bám sát FullFT trên Tulu3 nhưng lại tạo khoảng cách nhẹ hơn hoặc khác hình dạng trên OpenThoughts3, và ngược lại. Điều này gợi ý rằng kết quả của LoRA không chỉ phụ thuộc vào rank đơn lẻ, mà còn tương tác với cấu trúc dữ liệu SFT và động lực tối ưu hóa của từng mô hình. Tuy vậy, xu hướng lớn vẫn nhất quán: rank càng cao, đường cong càng gần FullFT; rank càng thấp, khả năng tách khỏi quỹ đạo giảm loss càng sớm.



Hình 4: Learning rate so với final test NLL trên Tulu3 cho nhiều LoRA rank và FullFT. Vùng LR tối ưu của LoRA thường nằm ở LR lớn hơn FullFT khoảng một bậc độ lớn, và tương đối ổn định theo rank (ngoại trừ rank rất nhỏ).

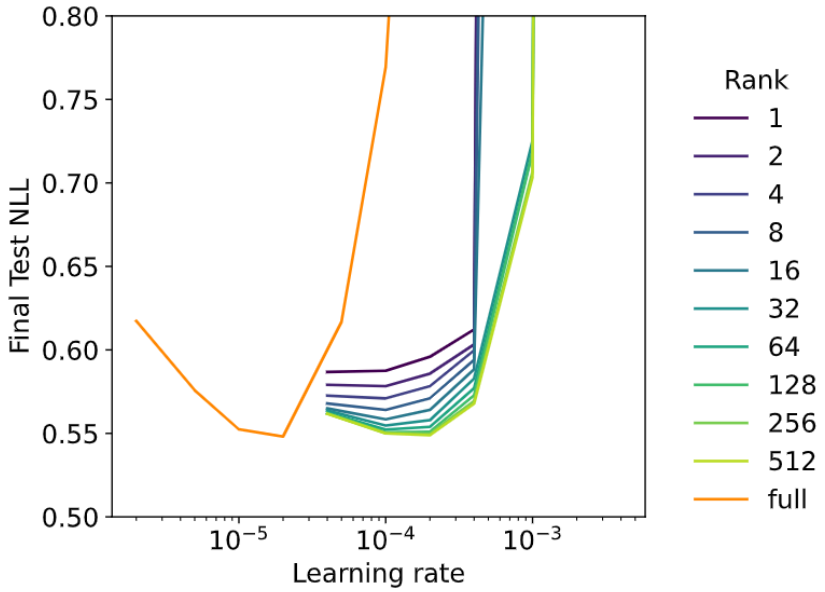
III.2. Learning rate của LoRA tỷ lệ khoảng 10 lần so với FullFT

Nếu chỉ nhìn Finding 1, ta có thể kết luận vội rằng chọn rank cao là đủ. Tuy nhiên, learning rate mới là yếu tố khiến nhiều triển khai LoRA trong thực hành bị đánh giá thấp hơn khả năng thật của nó. Hình 5 trình bày trực tiếp mối quan hệ giữa learning rate và final test NLL trên Tulu3, với nhiều rank khác nhau và cả FullFT. Đây là dạng đồ thị chữ U quen thuộc: LR quá nhỏ khiến học chậm và loss cao; LR quá lớn khiến tối ưu hóa kém ổn định và loss lại tăng; ở giữa là vùng LR tối ưu.

Điểm quan trọng nhất là vị trí vùng tối ưu của LoRA dịch sang phía learning rate lớn hơn so với FullFT. Trong hình, đường FullFT đạt điểm tốt ở LR thấp hơn, trong khi các rank LoRA tốt nhất thường đạt điểm tốt ở LR cao hơn, với tỷ lệ gần một bậc độ lớn. Nói cách khác, nếu ta lấy nguyên LR của FullFT rồi đem áp sang LoRA, khả năng cao là ta đang đặt LoRA vào phía

trái của chữ U, nơi tối ưu hóa chưa đủ "mạnh" để đạt loss thấp nhất. Khi đó, LoRA trông như kém hơn FullFT, nhưng nguyên nhân chính lại là do chọn LR chưa đúng miền.

Một chi tiết nữa là sự đồng nhất tương đối của LR tối ưu giữa các rank (trừ rank rất nhỏ). Trong Hình 5, các rank từ nhỏ đến lớn đều có vùng tối ưu nằm khá gần nhau, tức LR tối ưu không đổi quá mạnh khi tăng rank. Điều này giúp đơn giản hóa thực hành SFT: thay vì phải tune LR mới hoàn toàn cho từng rank, ta có thể coi LR LoRA là một giá trị nằm quanh một vùng tương đối ổn định, và điều chỉnh tinh hơn khi cần.



Hình 5: Learning rate so với final test NLL trên Tulu3 cho nhiều LoRA rank và FullFT. Vùng LR tối ưu của LoRA thường nằm ở LR lớn hơn FullFT khoảng một bậc độ lớn, và tương đối ổn định theo rank (ngoại trừ rank rất nhỏ).

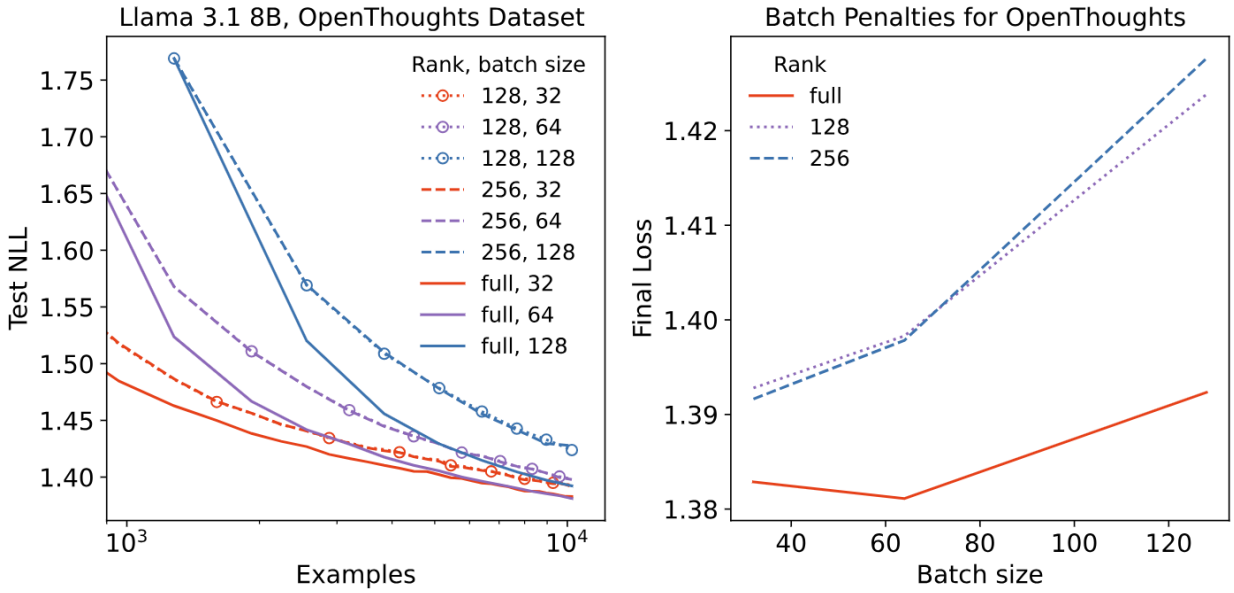
III.3. Finding 3: Batch size tạo ra khoảng cách giữa LoRA và FullFT

Sau khi đã đặt learning rate vào vùng hợp lý, một câu hỏi khác là: nếu ta thay đổi batch size, LoRA và FullFT có thay đổi như nhau không? Hình 6 cho thấy câu trả lời có thể là không, ít nhất trong một số kịch bản SFT (ví dụ trên một subset của OpenThoughts3).

Ở đồ thị bên trái, đường liền biểu diễn FullFT và đường nét đứt biểu diễn LoRA, với nhiều batch size khác nhau. Với batch nhỏ hơn (ví dụ 32), khoảng cách giữa LoRA và FullFT nhỏ hơn và có xu hướng thu hẹp theo thời gian. Nhưng khi batch tăng, khoảng cách này trở nên rõ rệt và kéo dài: ngay cả khi train tiếp, LoRA vẫn duy trì NLL cao hơn một mức nhất định so với FullFT.

Đồ thị bên phải chuyển quan sát đó thành final loss theo batch size. Khi batch tăng, loss của cả hai đều có thể tăng, nhưng LoRA tăng nhanh hơn, tức LoRA chịu mức phạt lớn hơn khi đẩy batch size lên cao. Điểm đáng chú ý là hiện tượng này không phải lúc nào cũng biến mất khi tăng rank. Nghĩa là, đây không chỉ là câu chuyện "rank thấp thì kém", mà còn phản ánh sự khác biệt về động lực tối ưu hóa giữa tham số hóa LoRA (tích ma trận) và tối ưu trực tiếp ma trận

đầy đủ như FullFT. Trong SFT, nơi ta thường muốn tối ưu hóa ổn định và giảm NLL đều theo step, khác biệt về batch size có thể trở thành một nguồn sai khác đáng kể nếu ta dùng batch quá lớn.

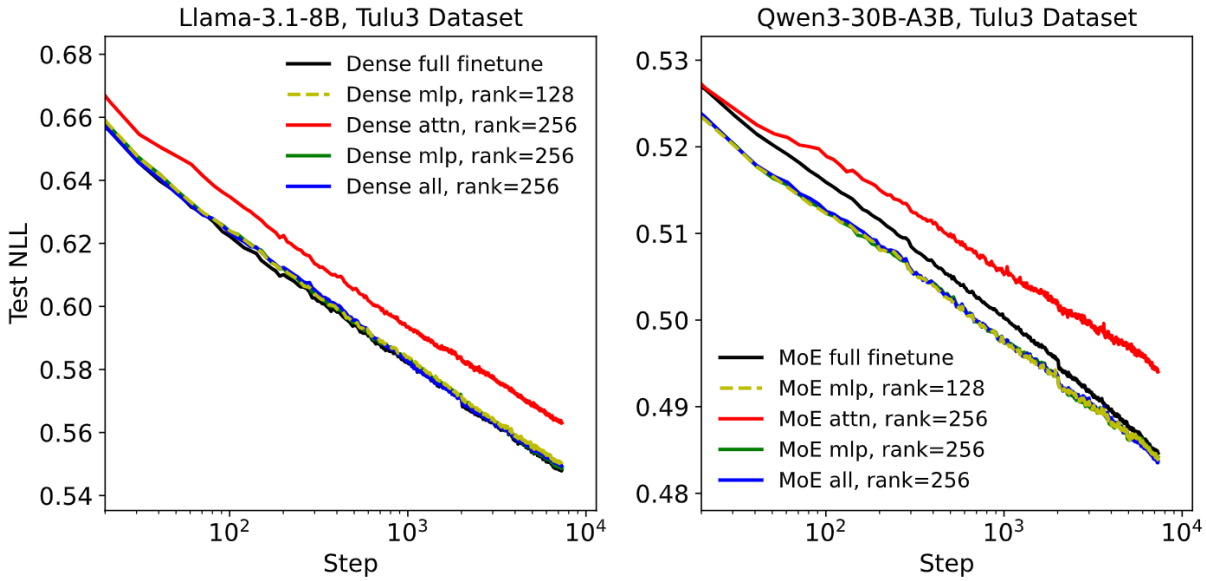


Hình 6: Ảnh hưởng của batch size lên khoảng cách LoRA và FullFT trong SFT (OpenThoughts3 subset). Trái: ở batch lớn, khoảng cách giữa LoRA và FullFT có thể duy trì theo step. Phải: final loss tăng theo batch size và LoRA chịu mức phạt lớn hơn.

III.4. Finding 4: Vị trí của LoRA quyết định chất lượng

Trong triển khai LoRA, một quyết định có tác động mạnh là đặt LoRA ở những ma trận nào. Một thực hành phổ biến trong giai đoạn đầu của LoRA là chỉ đặt vào các ma trận attention. Tuy nhiên, các thực nghiệm trong bài cho thấy lựa chọn này thường làm giảm hiệu năng trong SFT so với việc đặt LoRA vào MLP hoặc vào tất cả các lớp.

Hình 7 so sánh trực tiếp các cấu hình: FullFT, LoRA chỉ ở attention, LoRA chỉ ở MLP, và LoRA ở tất cả các lớp. Trên cả mô hình dense và mô hình MoE, cấu hình attention-only có đường cong NLL cao hơn rõ rệt. Ngược lại, MLP-only thường cải thiện mạnh và tiến gần cấu hình all-layers; đồng thời, khi đã đặt LoRA vào MLP, việc thêm LoRA vào attention không tạo ra mức cải thiện tương xứng như kỳ vọng. Điều này gợi ý rằng trong SFT, phần cập nhật quan trọng cần đi qua các khối MLP/MoE, nơi chứa phần lớn năng lực biến đổi của mô hình; nếu chỉ can thiệp attention, LoRA dễ rơi vào trạng thái cập nhật thiếu hiệu lực.



Hình 7: So sánh các cấu hình đặt LoRA theo layer trong SFT. Attention-only thường có NLL kém hơn đáng kể so với MLP-only và all-layers, và việc thêm attention sau khi đã có MLP không mang lại cải thiện rõ rệt.

Một phản biện tự nhiên là attention-only kém vì số tham số trainable ít hơn. Hình **??** cho thấy lập luận này không đủ. Trong ví dụ minh họa, attention rank 256 có số tham số trainable gần với MLP rank 128, nhưng vẫn có thể kém hơn về NLL. Điều này cho ta một thông điệp thực nghiệm rất rõ: không phải cứ tăng số tham số trainable là đủ, mà đặt LoRA vào đúng vị trí mới là yếu tố quyết định trong SFT.

Số tham số trainable của các cấu hình LoRA khác nhau. Dù attention-only rank cao có thể có số tham số gần với MLP-only rank thấp hơn, attention-only vẫn có thể kém hơn về NLL, cho thấy vị trí đặt LoRA quan trọng không kém việc đếm tham số.

Bảng cấu hình siêu tham số theo LoRA

Tham số	Giá trị
MLP, rank = 256	0.49B
Attention, rank = 256	0.25B
All, rank = 256	0.70B
MLP, rank = 128	0.24B

IV. Thực hành

Ở phần này chúng ta sẽ tái hiện lại 4 findings trong LoRA Without Regret, nhưng được điều chỉnh để phù hợp với bộ dữ liệu vi_gsm8k và pipeline SFT đang dùng và thực nghiệm trên Colab. Thay vì chỉ "chạy cho ra kết quả", mục tiêu ở đây là thiết kế thực nghiệm sao cho việc so sánh giữa LoRA và FullFT là công bằng: cùng giao thức huấn luyện, cùng cách log/đánh giá.

IV.1. Thiết lập chung

IV.1.1. Dataset và format SFT

Toàn bộ thực nghiệm dùng bộ dữ liệu `vi_gsm8k`, là phiên bản tiếng Việt của GSM8K (Grade School Math 8K) - một tập bài toán toán tiểu học dạng word problem kèm lời giải từng bước. Về cấu trúc, dữ liệu được chỉnh nhẹ so với GSM8K gốc bằng cách tách rõ `question`, `explanation` và `answer` thành các trường riêng, giúp thuận tiện khi đóng gói theo format SFT và khi đánh giá theo loss. Bộ dữ liệu có tổng cộng 8,792 mẫu, chia thành 7470 mẫu train và 1322 mẫu test, với hai tệp `vi_train.json` (train) và `vi_test.json` (test).

Vi-GSM8K

Question: Natalia đã bán kẹp tóc cho 48 người bạn của cô ấy vào tháng 4, và sau đó cô ấy đã bán nửa số lượng kẹp tóc đó vào tháng 5. Natalia đã bán tổng cộng bao nhiêu kẹp tóc trong tháng 4 và tháng 5?

Explaination: Natalia đã bán 24 kẹp trong tháng 5. Natalia đã bán tổng cộng 72 kẹp trong tháng 4 và tháng 5.

Answer: 72.

Hình 8: Minh họa về dữ liệu mẫu trong bộ dữ liệu vi-gsm8k.

Trong SFT, ta đóng gói mỗi mẫu thành một đoạn hội thoại theo kiểu instruction-following: phần `system` hướng dẫn mô hình giải từng bước; phần `user` chứa `question`; và phần `assistant` gồm `<think>explanation</think>` kèm theo `answer`. Cách đóng gói này giúp mô hình học đồng thời cả tiến trình lập luận (ở `explanation`) và đáp án cuối (ở `answer`), đồng thời giữ format nhất quán xuyên suốt các finding.

Khởi tạo conversation cho SFT

```
SYSTEM_PROMPT = (
    "Bạn là trợ lý AI chuyên giải toán. "
    "Hãy giải bài toán từng bước một cách rõ ràng, kiểm tra lại phép tính, "
    "và kết thúc bằng đáp án cuối cùng."
) # Khởi tạo system prompt để giúp mô hình trả lời tốt hơn

```

```python
def preprocess(example):
    return {
        "prompt": [
            {"role": "system", "content": SYSTEM_PROMPT},
            {"role": "user", "content": example["question"].strip()},
        ],  # Extract question
        "completion": [{
            "role": "assistant",
            "content":
            f"<think>{example['explanation'].strip()}</think>
            \n{str(example['answer']).strip()}", # Extract explaination, answer
        }],
    }
```

IV.1.2. Khởi tạo mô hình và tokenizer

Khởi tạo mô hình Qwen2.5 0.5B Instruct

```python

model_id = "Qwen2.5-0.5B-Instruct"
EOS_TOKEN = "<|im_end|>"

tokenizer = AutoTokenizer.from_pretrained(model_id, use_fast=True, trust_remote_code=
                                          True)
tokenizer.padding_side = "right"
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

use_bf16 = bool(torch.cuda.is_available() and torch.cuda.is_bf16_supported())
dtype = torch.bfloat16 if use_bf16 else torch.float16
```

IV.2. Tái thực nghiệm Finding 1

Thực nghiệm này nhằm quan sát trực tiếp động lực học của quá trình SFT khi thay đổi rank của LoRA, thông qua việc theo dõi `eval_loss` theo từng `step`. Trên cùng một giao thức huấn luyện, ta chạy rank sweep cho LoRA và đặt FullFT làm đường tham chiếu (baseline), từ đó đánh giá mức độ LoRA có thể bám sát quỹ đạo học của FullFT ở các rank khác nhau.

IV.2.1. Thiết kế thực nghiệm

Thực nghiệm Finding 1 được thiết kế theo một giao thức SFT cố định để đảm bảo việc so sánh giữa LoRA và FullFT là công bằng. Cụ thể, `micro_bs` là batch size trên mỗi GPU cho mỗi lần forward/backward (số mẫu xử lý trong một micro-batch). Do giới hạn bộ nhớ, ta không tối ưu ngay sau mỗi micro-batch mà dùng `grad_accum` (gradient accumulation steps): mô hình sẽ cộng dồn gradient qua `grad_accum` micro-batch liên tiếp rồi mới thực hiện một bước cập nhật tham

số (optimizer step). Vì vậy, `batch_eff` (effective batch size) phản ánh quy mô batch "tương đương" theo optimizer step, được tính theo

$$\texttt{batch\_eff} = \texttt{micro\_bs} \times \texttt{grad\_accum}$$

Trong Finding 1, ta cố định `micro_bs=4` và `grad_accum=4`, do đó `batch_eff=16`. Mỗi run huấn luyện trong `MAX_STEPS=200` optimizer steps, đánh giá định kỳ mỗi `EVAL_STEPS=10` steps, và giới hạn độ dài chuỗi ở `max_length=256`. Trên nền giao thức đó, ta tiến hành sweep theo rank của LoRA và learning rate: FullFT được sweep trên `LRS_FULL`, còn LoRA được sweep theo cặp `(rank, lr)` với `rank` $\in$ `RANKS` và `lr` $\in$ `LRS_LORA`.

Bảng cấu hình siêu tham số (Finding 1)

Tham số	Giá trị
`micro_bs` (batch per device)	4
`grad_accum` (accumulation steps)	4
`batch_eff` = `micro_bs` × `grad_accum`	16
`MAX_STEPS`	60
`EVAL_STEPS`	10
`max_length`	256
LoRA ranks	[1, 64, 256]
FullFT Learning rates	[1e-5, 2e-5]
LoRA Learning rates	[1e-4, 2e-4]

IV.2.2. Khung code

Khối hàm `make_sft_args(output_dir, lr)` có nhiệm vụ đóng gói toàn bộ cấu hình huấn luyện SFT vào một đối tượng `SFTConfig`, trong đó chỉ để hở hai biến mà sweep cần điều khiển là learning rate. Các thành phần còn lại được cố định để đảm bảo so sánh công bằng.

Cấu hình `SFTConfig` dùng chung cho sweep

```
def make_sft_args(output_dir, lr):
    return SFTConfig(
        output_dir=output_dir,
        learning_rate=lr,
        lr_scheduler_type="constant",
        warmup_ratio=0.0,
        per_device_train_batch_size=micro_bs,
        per_device_eval_batch_size=micro_bs,
        gradient_accumulation_steps=grad_accum,
        max_steps=MAX_STEPS,
        max_length=max_length,
        logging_strategy="steps",
```

```
        logging_steps=10,
        report_to=[],
        eval_strategy="steps",
        eval_steps=EVAL_STEPS,
        save_strategy="no",
        gradient_checkpointing=True,
        bf16=use_bf16,
        fp16=(not use_bf16),
        eos_token=EOS_TOKEN,
        seed=SEED,
        data_seed=SEED,    )
```

Khối `run_one_fullft(lr)` triển khai một run FullFT ứng với một learning rate cụ thể trong sweep. Hàm này đảm nhiệm toàn bộ vòng đời của một run: thiết lập seed để giảm nhiễu, `SFTTrainer` được khởi tạo với `make_sft_args` và chạy `train()`. Việc tách riêng FullFT theo từng LR giúp ta có baseline "đã quét LR", tránh kết luận bị lệch do chọn LR ngẫu nhiên.

Hàm chạy một run FullFT cho một learning rate

```
def run_one_fullft(lr):
    set_seed(SEED)
    model = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype=dtype,
                                                 device_map="auto")
    model.config.pad_token_id = tokenizer.pad_token_id
    if getattr(model, "generation_config", None) is not None:
        model.generation_config.pad_token_id = tokenizer.pad_token_id
        model.generation_config.eos_token_id = tokenizer.eos_token_id
    model.config.use_cache = False
    trainer = SFTTrainer(
        model=model,
        args=make_sft_args(out, lr),
        train_dataset=train_ds,
        eval_dataset=eval_ds,
        processing_class=tokenizer,    )
    trainer.train()
```

Khối `run_one_lora(rank, lr)` triển khai một run LoRA cho đúng một cặp `(rank, lr)` trong sweep. Hàm nạp mô hình nền vào biến `base`, chuẩn hoá token config như FullFT, rồi gắn adapter LoRA thông qua `LoraConfig`. Ở đây LoRA được đặt theo `target_modules="all-linear"` để can thiệp đồng bộ lên các lớp tuyến tính. Sau khi wrap mô hình bằng `get_peft_model`, ta huấn luyện với `SFTTrainer` giống hệt giao thức FullFT. Cách tổ chức theo từng cặp `(rank, lr)` tạo điều kiện để sau đó tổng hợp "best-of-LR" cho từng rank.

Hàm chạy một run LoRA cho một cặp `(rank, lr)`

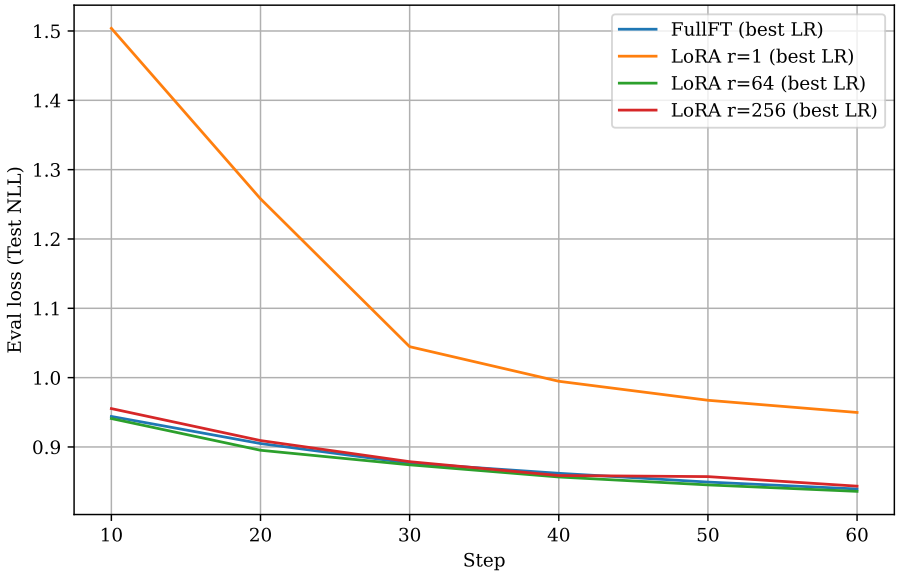
```
def run_one_lora(rank, lr):
    set_seed(SEED)
    base = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype=dtype,
                                                device_map="auto")
```

```
    base.config.pad_token_id = tokenizer.pad_token_id
    if getattr(base, "generation_config", None) is not None:
        base.generation_config.pad_token_id = tokenizer.pad_token_id
        base.generation_config.eos_token_id = tokenizer.eos_token_id
    base.config.use_cache = False
    peft_cfg = LoraConfig(
        r=rank,
        lora_alpha=rank,
        target_modules="all-linear",
        task_type="CAUSAL_LM",
        lora_dropout=0.0,
        bias="none",    )
    model = get_peft_model(base, peft_cfg)
    trainer = SFTTrainer(
        model=model,
        args=make_sft_args(out, lr),
        train_dataset=train_ds,
        eval_dataset=eval_ds,
        processing_class=tokenizer,    )
    trainer.train()
```

IV.2.3. Kết quả thực nghiệm

Nhìn vào đường cong `eval_loss` theo `step`, ta thấy rank rất thấp (`r=1`) tách hẳn khỏi nhóm tốt ngay từ đầu: loss khởi điểm cao (xấp xỉ 1.5 ở mốc đánh giá đầu) và dù giảm đều theo step vẫn duy trì khoảng cách lớn so với FullFT cho tới cuối, phản ánh việc adapter hạng quá thấp không đủ bậc tự do để đi theo quỹ đạo giảm loss của fine-tune toàn tham số, ngay cả khi đã chọn learning rate tốt nhất tại từng mốc.

Ngược lại, với rank trung bình và cao (`r=64`, `r=256`), các đường cong gần như chồng lên FullFT trong phần lớn quá trình huấn luyện. Từ khoảng step 20 trở đi, ba đường FullFT/`r=64`/`r=256` bám rất sát nhau và hội tụ về cùng một mức loss ở các mốc cuối; trong đó `r=64` đôi lúc nhỉnh hơn rất nhẹ, còn `r=256` gần như trùng với FullFT. Điều này cho thấy trong setting hiện tại, LoRA với rank đủ lớn có thể tái hiện training dynamics của FullFT một cách gần như tương đương, trong khi việc tăng rank từ 64 lên 256 mang lại lợi ích biên không đáng kể ở miền huấn luyện ngắn này.



Hình 9: Eval loss theo step trong Finding 1, với các đường LoRA được tổng hợp theo best-of-LR ở từng rank và đường tham chiếu FullFT.

IV.3. Tái thực nghiệm Finding 2

Nếu Finding 1 quan sát động lực học theo `step`, thì Finding 2 chuyển sang góc nhìn "chọn learning rate": với mỗi cấu hình (FullFT hoặc LoRA ở một rank), ta quét một dải learning rate và đo `eval_loss` ở cuối run, đồng thời so sánh miền learning rate tối ưu giữa FullFT và LoRA theo rank. Mục tiêu cốt lõi là kiểm tra thông điệp: learning rate tối ưu phụ thuộc vào capacity (mức độ "mạnh" của tinh chỉnh: FullFT và/hoặc LoRA rank lớn).

IV.3.1. Thiết kế thực nghiệm

Thực nghiệm Finding 2 giữ nguyên giao thức SFT để đảm bảo so sánh công bằng: `micro_bs` là batch size trên mỗi GPU, `grad_accum` là số micro-batch cộng dồn trước khi cập nhật tham số, và batch hiệu dụng được xác định bởi

$$\texttt{batch\_eff} = \texttt{micro\_bs} \times \texttt{grad\_accum}.$$

Các siêu tham số còn lại như `max_length`, `MAX_STEPS` (hoặc quy đổi theo `epochs_eq`) và `EVAL_STEPS` được giữ cố định theo thiết lập chung. Trên nền đó, ta thực hiện LR sweep: FullFT quét `LRS_FULL` (5 giá trị), còn LoRA quét theo cặp `(rank, lr)` với `rank` $\in$ `RANKS` và `lr` $\in$ `LRS_LORA` (5 giá trị). Mỗi điểm trên đồ thị tương ứng với `eval_loss` cuối run tại learning rate đó; "điểm ngọt" là LR cho final loss nhỏ nhất trong từng cấu hình.

Bảng cấu hình siêu tham số (Finding 2)

Tham số	Giá trị
micro_bs	4
grad_accum	4
batch_eff = micro_bs × grad_accum	16
MAX_STEPS	60
EVAL_STEPS	10
max_length	256
LoRA ranks	[16, 64, 256]
FullFT Learning rates	[5e-6, 1e-5, 2e-5, 5e-5, 1e-4]
LoRA Learning rates	[5e-5, 1e-4, 2e-4, 4e-4, 8e-4]

IV.3.2. Khung code

Đóng gói toàn bộ giao thức SFT vào SFTConfig. Hàm chỉ để hở biến sweep là lr (learning rate); các siêu tham số còn lại (batch, accumulation, steps, eval/logging, precision) được cố định để đảm bảo so sánh công bằng giữa các run.

Cấu hình SFTConfig dùng chung cho LR sweep

```
def make_sft_args(output_dir, lr):
    return SFTConfig(
        output_dir=output_dir,
        learning_rate=lr,
        lr_scheduler_type="constant",
        warmup_ratio=0.0,
        per_device_train_batch_size=micro_bs,
        per_device_eval_batch_size=micro_bs,
        gradient_accumulation_steps=grad_accum,
        max_steps=MAX_STEPS,
        max_length=max_length,
        logging_strategy="steps",
        logging_steps=10,
        report_to=[],
        eval_strategy="steps",
        eval_steps=EVAL_STEPS,
        save_strategy="no",
        gradient_checkpointing=True,
        bf16=use_bf16,
        fp16=(not use_bf16),
        eos_token=EOS_TOKEN,
        seed=SEED,
        data_seed=SEED,
    )
```

Chạy một run FullFT ứng với một lr trong sweep. Hàm nạp mô hình nền, chuẩn hoá pad_token_id/eos_token
tắt use_cache để tương thích gradient checkpointing, sau đó khởi tạo SFTTrainer và gọi train().

Hàm chạy một run FullFT cho một learning rate

```python
def run_one_fullft(output_dir, lr):
    set_seed(SEED)

    model = AutoModelForCausalLM.from_pretrained(
        model_id, torch_dtype=dtype, device_map="auto"
    )
    model.config.pad_token_id = tokenizer.pad_token_id
    if getattr(model, "generation_config", None) is not None:
        model.generation_config.pad_token_id = tokenizer.pad_token_id
        model.generation_config.eos_token_id = tokenizer.eos_token_id
    model.config.use_cache = False

    trainer = SFTTrainer(
        model=model,
        args=make_sft_args(output_dir, lr),
        train_dataset=train_ds,
        eval_dataset=eval_ds,
        processing_class=tokenizer,
    )
    trainer.train()
```

Chạy một run LoRA ứng với đúng một cặp (rank, lr) trong sweep. Hàm nạp mô hình nền vào base, chuẩn hoá token config giống FullFT, rồi gắn adapter LoRA với LoraConfig (đặt target_modules="all-linear"để can thiệp đồng bộ lên các lớp tuyến tính). Sau đó huấn luyện bằng SFTTrainer cùng giao thức như FullFT.

Hàm chạy một run LoRA cho một cặp (rank, lr)

```python
def run_one_lora(output_dir, rank, lr):
    set_seed(SEED)

    base = AutoModelForCausalLM.from_pretrained(
        model_id, torch_dtype=dtype, device_map="auto"
    )
    base.config.pad_token_id = tokenizer.pad_token_id
    if getattr(base, "generation_config", None) is not None:
        base.generation_config.pad_token_id = tokenizer.pad_token_id
        base.generation_config.eos_token_id = tokenizer.eos_token_id
    base.config.use_cache = False

    peft_cfg = LoraConfig(
        r=rank,
        lora_alpha=rank,
        target_modules="all-linear",
        task_type="CAUSAL_LM",
        lora_dropout=0.0,
        bias="none",
```

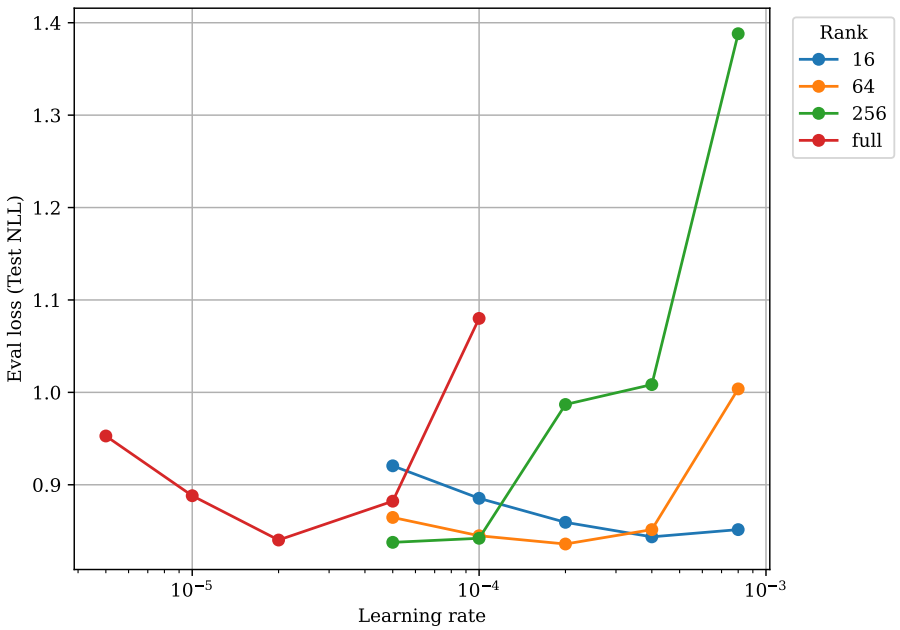
```
    )
    model = get_peft_model(base, peft_cfg)

    trainer = SFTTrainer(
        model=model,
        args=make_sft_args(output_dir, lr),
        train_dataset=train_ds,
        eval_dataset=eval_ds,
        processing_class=tokenizer,
    )
    trainer.train()
```

IV.3.3. Kết quả thực nghiệm

Nhìn vào kết quả Finding 2, xu hướng chính là learning rate tối ưu phụ thuộc vào capacity. Trước hết, FullFT đòi hỏi learning rate nhỏ hơn rõ rệt. Đường FullFT đạt mức tốt nhất quanh cỡ 10^{-5} (xấp xỉ 2×10^{-5}), nhưng khi tăng lên vùng 10^{-4} thì `eval_loss` tăng mạnh, phản ánh hiện tượng "quá tay" và suy giảm ổn định tối ưu trong miền step đang xét. Nói cách khác, FullFT là cấu hình có capacity lớn nhất (nhiều tham số được cập nhật nhất) nên miền LR ổn định hẹp hơn và sai LR sẽ hỏng nhanh hơn.

Ngược lại, LoRA với rank thấp/trung bình chịu learning rate lớn tốt hơn: các đường `r=16` và đặc biệt `r=64` cải thiện khi tăng LR trong dải sweep, và `r=64` thể hiện tốt quanh cỡ vài $\times 10^{-4}$ (xấp xỉ 2×10^{-4}), cho final loss thấp nhất trong nhóm LoRA. `r=16` ổn định hơn nhưng đáy loss thường cao hơn `r=64`, cho thấy capacity chưa đủ để đạt mức tốt nhất dù tune LR thuận lợi.

Với LoRA rank cao, độ nhạy với LR tăng lên đáng kể. `r=256` có thể hoạt động tốt ở LR nhỏ (khoảng 5×10^{-5} đến 10^{-4}), nhưng khi đẩy LR lên vùng cao hơn (đặc biệt điểm ngoài cùng phải trong sweep) thì `eval_loss` tăng vọt, gần như mất ổn định. Điều này nhất quán với luận điểm: capacity càng lớn (FullFT hoặc LoRA rank lớn) thì LR tối ưu càng phải nhỏ và vùng ổn định càng hẹp.



Hình 10: Kết quả eval loss theo learning rate cho FullFT và LoRA theo rank (LR sweep).

IV.4. Tái thực nghiệm Finding 3

Finding 3 kiểm tra ảnh hưởng của batch size lên khoảng cách giữa LoRA và FullFT trong SFT. Thay vì cố định `grad_accum` như Finding 1, ta chủ động thay đổi `grad_accum` để tạo nhiều mức `batch_eff`, rồi so sánh FullFT và LoRA ở từng mức batch. Để tránh kết luận bị lệch do chọn learning rate, ở mỗi `batch_eff` ta sweep LR và dựng đường cong *best-of-LR* theo `step` (tại mỗi `step`, lấy LR cho ra `eval_loss` nhỏ nhất) nhằm phản ánh "cận dưới thực nghiệm" của từng chế độ.

IV.4.1. Thiết kế thực nghiệm

Thực nghiệm Finding 3 giữ nguyên khung SFT của Finding 1 nhưng thay đổi `grad_accum` để điều khiển batch hiệu dụng. Cụ thể, ta cố định `MICRO_BS` và quét `grad_accum` theo danh sách `ACCUM_LIST`; với mỗi giá trị `grad_accum`, batch hiệu dụng được xác định bởi

$$\texttt{batch\_eff} = \texttt{MICRO\_BS} \times \texttt{grad\_accum}.$$

Tại từng `batch_eff`, FullFT được chạy với sweep learning rate `LR_LIST_FULL`. Với LoRA, ta giữ rank cố định (đặt `RANK=256`) và sweep learning rate theo `LR_LIST_LORA`.

Bảng cấu hình siêu tham số (Finding 3)

Tham số	Giá trị
MICRO_BS	4
ACCUM_LIST	[2, 4, 8]
MAX_STEPS	60
EVAL_STEPS	10
max_length	256
LoRA rank	256
FullFT LR sweep	[1e-5, 2e-5]
LoRA LR sweep	[1e-4, 2e-4]

IV.4.2. Khung code

Trong Finding 3, phần triển khai giữ "cốt lõi" của mỗi run: nạp mô hình, chuẩn hoá token config, gắn LoRA (nếu có), khởi tạo `SFTTrainer`, rồi chạy `train()`. Hai biến sweep là `grad_accum` (tạo `batch_eff`) và learning rate.

Cấu hình `SFTConfig` dùng chung

```python
def make_sft_args(output_dir, lr, grad_accum):
    return SFTConfig(
        output_dir=output_dir,
        learning_rate=lr,
        lr_scheduler_type="constant",
        warmup_ratio=0.0,
        per_device_train_batch_size=MICRO_BS,
        per_device_eval_batch_size=MICRO_BS,
        gradient_accumulation_steps=grad_accum,
        max_steps=MAX_STEPS,
        max_length=MAX_LENGTH,
        logging_strategy="steps",
        logging_steps=10,
        report_to=[],
        eval_strategy="steps",
        eval_steps=EVAL_STEPS,
        save_strategy="no",
        gradient_checkpointing=True,
        bf16=use_bf16,
        fp16=(not use_bf16),
        eos_token=EOS_TOKEN,
        seed=SEED,
        data_seed=SEED,
    )
```

Một run FullFT cho một cặp (grad_accum

```python
def run_one_fullft(output_dir, grad_accum, lr):
    set_seed(SEED)

    model = AutoModelForCausalLM.from_pretrained(
        model_id, torch_dtype=dtype, device_map="auto"
    )
    model.config.pad_token_id = tokenizer.pad_token_id
    if getattr(model, "generation_config", None) is not None:
        model.generation_config.pad_token_id = tokenizer.pad_token_id
        model.generation_config.eos_token_id = tokenizer.eos_token_id
    model.config.use_cache = False

    trainer = SFTTrainer(
        model=model,
        args=make_sft_args(output_dir, lr, grad_accum),
        train_dataset=train_ds,
        eval_dataset=eval_ds,
        processing_class=tokenizer,
    )
    trainer.train()
```

Một run LoRA (RANK

```python
def run_one_lora(output_dir, grad_accum, lr):
    set_seed(SEED)

    base = AutoModelForCausalLM.from_pretrained(
        model_id, torch_dtype=dtype, device_map="auto"
    )
    base.config.pad_token_id = tokenizer.pad_token_id
    if getattr(base, "generation_config", None) is not None:
        base.generation_config.pad_token_id = tokenizer.pad_token_id
        base.generation_config.eos_token_id = tokenizer.eos_token_id
    base.config.use_cache = False

    peft_cfg = LoraConfig(
        r=RANK,
        lora_alpha=RANK,              # alpha/r = 1
        target_modules="all-linear",
        task_type="CAUSAL_LM",
        lora_dropout=0.0,
        bias="none",
    )
    model = get_peft_model(base, peft_cfg)

    trainer = SFTTrainer(
        model=model,
        args=make_sft_args(output_dir, lr, grad_accum),
        train_dataset=train_ds,
        eval_dataset=eval_ds,
```

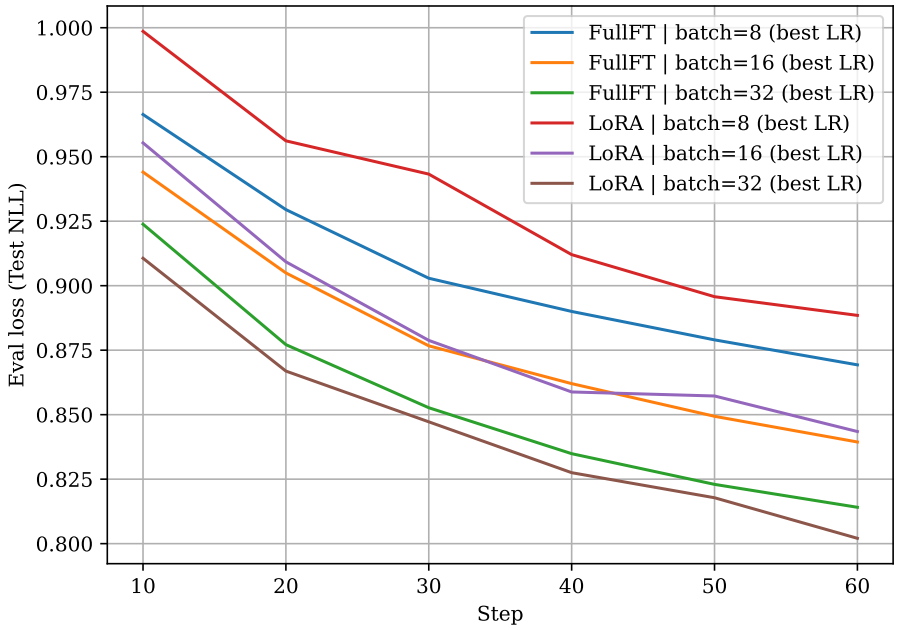
```
        processing_class=tokenizer,
    )
    trainer.train()
```

IV.4.3. Kết quả thực nghiệm

Nhìn vào kết quả khi thay đổi `batch_eff` = 8, 16, 32 cho FullFT và LoRA, xu hướng nổi bật là batch càng lớn thì loss càng thấp. Ở hầu hết các mốc đánh giá, các đường ứng với `batch_eff=32` luôn nằm thấp nhất, sau đó đến `batch_eff=16`, và cao nhất là `batch_eff=8`.

Một điểm đáng chú ý là LoRA nhạy với batch hơn FullFT trong thiết lập này. Với `batch_eff=8`, LoRA cho kết quả kém nhất và tạo khoảng cách rõ rệt. Khi tăng lên `batch_eff=32`, LoRA cải thiện mạnh và thậm chí đạt mức tốt nhất ở đoạn cuối. Trong khi đó, FullFT cũng hưởng lợi từ batch lớn nhưng mức cải thiện diễn ra đều hơn: `batch_eff=32` tốt hơn `batch_eff=16`, và `batch_eff=8` cao hơn một cách ổn định.

Ở batch trung bình/cao, khoảng cách FullFT và LoRA thu hẹp rõ rệt. Với `batch_eff=16`, hai đường gần như chồng lên nhau ở nửa sau quá trình huấn luyện; với `batch_eff=32`, LoRA nhỉnh hơn nhẹ ở đoạn cuối.



Hình 11: Ảnh hưởng của `batch_eff` lên `eval_loss` theo `step` dưới chế độ best-of-LR.

IV.5. Tái thực nghiệm Finding 4

Finding 4 kiểm tra tác động của vị trí đặt LoRA trong mô hình lên chất lượng SFT. Thay vì chỉ hỏi "LoRA có tốt không", thực nghiệm này đi sâu vào câu hỏi LoRA nên được gắn vào nhóm

lớp tuyến tính nào: `attention-only`, `mlp-only`, hay `all-linear`. Mục tiêu là so sánh trực tiếp các lựa chọn này trong cùng giao thức huấn luyện và cùng cách quét learning rate, từ đó kiểm tra liệu cấu hình attention-only có kém hơn một cách hệ thống hay không.

IV.5.1. Thiết kế thực nghiệm

Thực nghiệm Finding 4 giữ nguyên khung SFT ngắn và quét learning rate như các finding trước, nhưng thay đổi `target_modules` của LoRA theo ba chế độ: `all-linear`, `attention-only`, và `mlp-only`. Để tránh so sánh bị nhiễu bởi khác biệt số tham số trainable giữa các chế độ, ta bổ sung thêm nhánh `match-params`. Cụ thể, trước khi chạy sweep, ta nạp một probe model trên CPU để tự động phát hiện các hậu tố tên lớp tuyến tính thuộc attention và MLP.

Với mỗi nhóm, ta tính một hệ số quy mô xấp xỉ bằng cách cộng $\sum$(`in_features`+`out_features`) qua các Linear thuộc nhóm đó. Từ hai hệ số thu được, ta duyệt qua hai tập ứng viên rank và chọn cặp (`attn_match_rank`, `mlp_match_rank`) sao cho quy mô LoRA giữa attention-only và mlp-only gần bằng nhau nhất. Nhánh `match-params` nhờ vậy giúp kiểm tra xem attention-only kém là do "đặt sai chỗ" hay chỉ vì "ít tham số hơn".

Với mỗi chế độ module, ta sweep learning rate. Thực nghiệm gồm hai nhóm: (i) nhóm std dùng cùng một rank chuẩn cho cả ba chế độ `all-linear/attention-only/mlp-only`; và (ii) nhóm match dùng `attn_match_rank` cho attention-only và `mlp_match_rank` cho mlp-only để cân bằng quy mô tham số.

Bảng cấu hình siêu tham số (Finding 4)

Tham số	Giá trị
`micro_bs`	4
`grad_accum`	4
`max_length`	256
`MAX_STEPS`	60
`EVAL_STEPS`	10
LoRA modes (`target_modules`)	[`all-linear`, `attention-only`, `mlp-only`]
Rank	128

IV.5.2. Khung code (Finding 4)

Finding 4 có hai phần cốt lõi: (i) probe model để phát hiện nhóm module và chọn cặp rank match-params; (ii) hàm chạy một run LoRA với `target_modules` tương ứng (`all-linear` / danh sách suffix của attention / danh sách suffix của MLP). Phần code dưới đây chỉ giữ các thao tác tối thiểu phục vụ thí nghiệm (không lưu `final`, không `summary`).

Probe model để detect suffix và chọn match-params ranks

```
ATTN_CANDIDATES = ["q_proj", "k_proj", "v_proj", "o_proj",
                   "qkv_proj", "W_pack", "c_attn"]
MLP_CANDIDATES  = ["gate_proj", "up_proj", "down_proj",
                   "w1", "w2", "w3", "c_fc", "c_proj"]
def detect_suffixes(model, candidates):
    names = [n for n, _ in model.named_modules()]
    return [s for s in candidates if any(n.endswith(s) for n in names)]
def sum_in_out_for_suffixes(model, suffixes):
    total = 0
    for name, module in model.named_modules():
        if isinstance(module, torch.nn.Linear) and any(name.endswith(s) for s in
                                                suffixes):
            total += (module.in_features + module.out_features)
    return total
probe = AutoModelForCausalLM.from_pretrained(
    model_id, torch_dtype=torch.float32, device_map=None)
attn_suffixes = detect_suffixes(probe, ATTN_CANDIDATES)
mlp_suffixes  = detect_suffixes(probe, MLP_CANDIDATES)
attn_coef = sum_in_out_for_suffixes(probe, attn_suffixes)
mlp_coef  = sum_in_out_for_suffixes(probe, mlp_suffixes)
ATTN_RANK_CAND = [64, 96, 128, 192, 256, 320, 384, 512]
MLP_RANK_CAND  = [16, 24, 32, 48, 64, 80, 96, 112, 128, 160]
best_pair, best_diff = None, None
for ra in ATTN_RANK_CAND:
    for rm in MLP_RANK_CAND:
        diff = abs(ra * attn_coef - rm * mlp_coef)
        if best_diff is None or diff < best_diff:
            best_diff, best_pair = diff, (ra, rm)
attn_match_rank, mlp_match_rank = best_pair
```

Cấu hình `SFTConfig` dùng chung cho LR sweep

```
def make_sft_args(output_dir, lr):
    return SFTConfig(
        output_dir=output_dir,
        learning_rate=lr,
        lr_scheduler_type="constant",
        warmup_ratio=0.0,
        per_device_train_batch_size=micro_bs,
        per_device_eval_batch_size=micro_bs,
        gradient_accumulation_steps=grad_accum,
        max_steps=MAX_STEPS,
        max_length=max_length,
        logging_strategy="steps",
        logging_steps=10,
        report_to=[],
        eval_strategy="steps",
        eval_steps=EVAL_STEPS,
        save_strategy="no",
        gradient_checkpointing=True,
```

```
        bf16=use_bf16,
        fp16=(not use_bf16),
        eos_token=EOS_TOKEN,
        seed=SEED,
        data_seed=SEED,
    )
```

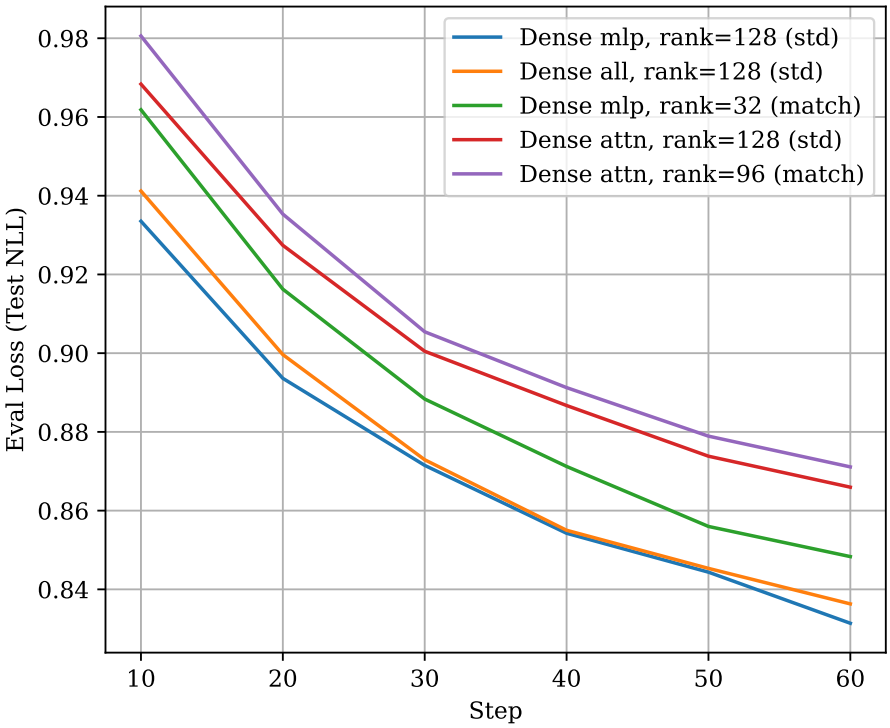
Một run LoRA theo `mode` và `rank`

```
def run_one_lora(output_dir, mode, rank, lr):
    set_seed(SEED)
    base = AutoModelForCausalLM.from_pretrained(
        model_id, torch_dtype=dtype, device_map="auto"    )
    base.config.pad_token_id = tokenizer.pad_token_id
    if getattr(base, "generation_config", None) is not None:
        base.generation_config.pad_token_id = tokenizer.pad_token_id
        base.generation_config.eos_token_id = tokenizer.eos_token_id
    base.config.use_cache = False
    if mode == "all-linear":
        target_modules = "all-linear"
    elif mode == "attention-only":
        target_modules = attn_suffixes
    elif mode == "mlp-only":
        target_modules = mlp_suffixes
    else:
        raise ValueError("Invalid mode: " + str(mode))
    peft_cfg = LoraConfig(
        r=int(rank),
        lora_alpha=int(rank),
        target_modules=target_modules,
        task_type="CAUSAL_LM",
        lora_dropout=0.0,
        bias="none",
    )
    model = get_peft_model(base, peft_cfg)
    trainer = SFTTrainer(
        model=model,
        args=make_sft_args(output_dir, lr),
        train_dataset=train_ds,
        eval_dataset=eval_ds,
        processing_class=tokenizer,
    )
    trainer.train()
```

IV.5.3. Kết quả thực nghiệm

Kết quả cho thấy khác biệt theo vị trí đặt LoRA là rất rõ rằng `attention-only` thường kém hơn `mlp-only` và `all-linear`. Trên đồ thị, nhóm tốt nhất là `mlp-only` (rank chuẩn, `std`) và `all-linear` (rank chuẩn, `std`): hai đường bám rất sát nhau và thường nằm thấp nhất. Điều này

gợi ý rằng phần lớn lợi ích của LoRA trong thiết lập này đến từ việc can thiệp vào MLP; việc "thêm" LoRA vào attention (khi đã có MLP) không tạo ra cải thiện lớn, nên `all-linear` chỉ hoà hoặc nhỉnh hơn rất nhẹ so với `mlp-only`.



Hình 12: Kết quả so sánh vị trí đặt LoRA trong mô hình (`all-linear`, `attention-only`, `mlp-only`) dưới LR sweep, kèm nhánh `match-params` để cân bằng quy mô tham số.

V. Câu hỏi trắc nghiệm

1. Mục tiêu chính của Parameter-Efficient Fine-Tuning (PEFT) so với fine-tune toàn tham số là gì?
   (a) Cập nhật toàn bộ trọng số để đạt loss thấp nhất trong mọi trường hợp.
   (b) Chỉ cập nhật một phần nhỏ tham số để giảm bộ nhớ huấn luyện và chi phí lưu trữ, đồng thời dễ triển khai nhiều biến thể theo tác vụ.
   (c) Chỉ dùng dữ liệu pretraining lớn để tránh overfitting khi post-training.
   (d) Tăng kích thước mô hình để cải thiện chất lượng suy luận.

2. Trong LoRA, ma trận trọng số của một lớp tuyến tính được sử dụng như thế nào khi tinh chỉnh?
   (a) $\mathbf{W}$ được cập nhật trực tiếp bằng gradient descent.
   (b) $\mathbf{W}$ được giữ cố định, chỉ học $\Delta\mathbf{W}$ dưới dạng hạng thấp rồi cộng vào khi chạy mô hình.
   (c) Chỉ học lại embedding, không học lớp tuyến tính.
   (d) Chỉ học $\mathbf{W}$ còn $\Delta\mathbf{W}$ bị đặt bằng 0 trong suốt quá trình train.

3. Ý nghĩa của rank r trong LoRA là gì?
   (a) Là số lượng lớp Transformer được thay đổi trọng số.
   (b) Là số bước huấn luyện tối đa.
   (c) Là số chiều của không gian con hạng thấp dùng để biểu diễn cập nhật $\Delta\mathbf{W} = \mathbf{BA}$.
   (d) Là số token tối đa của chuỗi đầu vào.

4. Công thức LoRA có hệ số scale thường dùng là gì?
   (a) $\mathbf{W}' = \mathbf{W} + \alpha\mathbf{BA}$
   (b) $\mathbf{W}' = \mathbf{W} + \left(\frac{\alpha}{r}\right)\mathbf{BA}$
   (c) $\mathbf{W}' = \mathbf{W} + \left(\frac{r}{\alpha}\right)\mathbf{BA}$
   (d) $\mathbf{W}' = \mathbf{W} + \alpha r\,\mathbf{BA}$

5. Thực hành khởi tạo nào giúp mô hình ban đầu hành xử gần giống mô hình gốc trong LoRA?
   (a) Khởi tạo $\mathbf{W}$ ngẫu nhiên lại từ đầu, giữ $\mathbf{A}, \mathbf{B}$ bằng 0.
   (b) Đặt một trong hai ma trận sao cho $\Delta\mathbf{W}$ ban đầu xấp xỉ 0 (ví dụ $\mathbf{B} = \mathbf{0}$, $\mathbf{A}$ ngẫu nhiên nhỏ).
   (c) Khởi tạo cả $\mathbf{A}$ và $\mathbf{B}$ bằng 1 để tăng tốc hội tụ.
   (d) Đặt $\Delta\mathbf{W} = \mathbf{W}$ để mô hình học nhanh hơn.

6. Trong LoRA Without Regret (SFT), tác giả dùng thước đo nào để theo dõi động lực học huấn luyện thay vì đánh giá bằng sampling?

(a) BLEU score trên tập test.

(b) Accuracy sau khi decode greedy.

(c) Test negative log-likelihood (test NLL).

(d) ROUGE-L trên câu trả lời sinh ra.

7. "Best-of-sweep theo step" trong cách vẽ đường cong của bài có nghĩa là gì?

   (a) Chọn một learning rate cố định duy nhất cho tất cả cấu hình và mọi step.

   (b) Ở mỗi step, với một cấu hình (ví dụ một rank), chọn learning rate cho NLL nhỏ nhất tại step đó.

   (c) Chọn learning rate lớn nhất để đường cong giảm nhanh nhất.

   (d) Chọn learning rate nhỏ nhất để tránh dao động loss.

8. Theo Finding 2 trong SFT, kết luận nào đúng về learning rate tối ưu của LoRA so với FullFT?

   (a) LoRA tối ưu ở learning rate nhỏ hơn FullFT khoảng 10 lần.

   (b) LoRA và FullFT có learning rate tối ưu giống hệt nhau trong mọi trường hợp.

   (c) LoRA thường tối ưu ở learning rate lớn hơn FullFT khoảng một bậc độ lớn (xấp xỉ 10 lần).

   (d) Learning rate không ảnh hưởng đến LoRA nếu rank đủ lớn.

9. Theo Finding 3, phát biểu nào mô tả đúng ảnh hưởng của batch size trong một số bối cảnh SFT?

   (a) Tăng batch size luôn làm LoRA tốt hơn FullFT.

   (b) Batch size không ảnh hưởng đến khoảng cách giữa LoRA và FullFT.

   (c) Ở batch lớn, khoảng cách NLL giữa LoRA và FullFT có thể rõ rệt và kéo dài theo step; LoRA chịu "mức phạt" lớn hơn khi tăng batch.

   (d) Tăng batch size luôn làm khoảng cách biến mất nếu tăng rank.

10. Theo Finding 4 về vị trí gắn LoRA, nhận định nào phù hợp nhất với kết quả thực nghiệm trong SFT?

    (a) Attention-only thường tốt nhất vì attention là nơi mô hình "tập trung" thông tin.

    (b) MLP-only thường kém nhất vì MLP không ảnh hưởng tới biểu diễn.

    (c) Attention-only thường kém hơn MLP-only và all-layers; vị trí đặt LoRA quan trọng không kém (thậm chí hơn) việc chỉ so số tham số trainable.

    (d) Chỉ cần làm match số tham số trainable thì attention-only luôn tương đương MLP-only.

VI. Tài liệu tham khảo

[1] K. V. Nguyen et al., "Uit-vsfc: Vietnamese students' feedback corpus for sentiment analysis", in *2018 10th International Conference on Knowledge and Systems Engineering (KSE)*, 2018, pp. 19–24. DOI: 10.1109/KSE.2018.8573337.

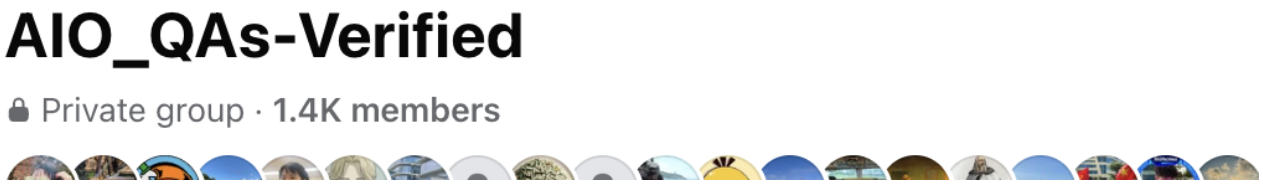
[2] S. T. Truong et al., *Crossing linguistic horizons: Finetuning and comprehensive evaluation of vietnamese large language models*, 2024. arXiv: 2403.02715 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2403.02715.

[3] T. Dettmers et al., *Qlora: Efficient finetuning of quantized llms*, 2023. arXiv: 2305.14314 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2305.14314.

[4] J. Schulman et al., "Lora without regret", *Thinking Machines Lab: Connectionism*, 2025, https://thinkingmachines.ai/blog/lora/. DOI: 10.64434/tml.20250929.

Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại đây.
2. **Code:** Các file code cài đặt hoàn chỉnh có thể được tải tại đây.



Hình 13: Hình ảnh group facebook AIO Q&A.