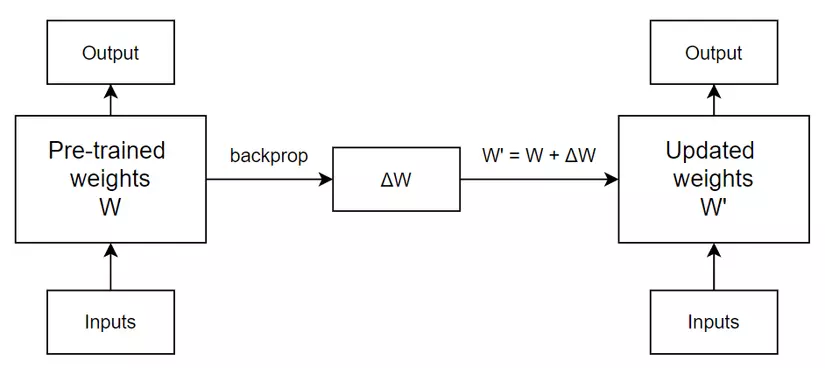
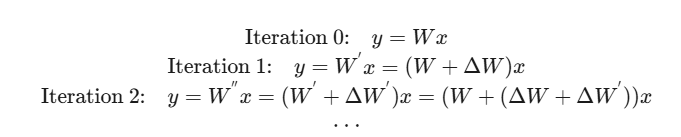
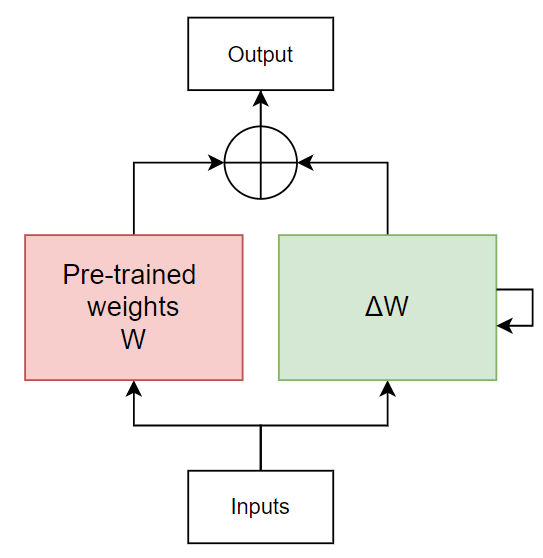
1. Mô tả đề tài:
   1. Trong thời đại mà AI dần trở thành một phần của cuộc sống như Internet ngày xưa, việc fine-tune các mô hình lớn như **Stable Diffusion**, **LLM** trở nên ngày càng phổ biến, đặc biệt là với những người dùng có nhu cầu điều chỉnh những model này cho các tác vụ cụ thể. Tuy nhiên, việc fine-tune toàn bộ model với hàng trăm triệu đến hàng tỷ tham số là một thách thức rất lớn, do yêu cầu về phần cứng, bộ nhớ và tài nguyên tính toán. Việc này đã khiến fine-tune trở thành một thứ gì đó xa xỉ đối với người dùng phổ thông, những người chỉ có tài nguyên tính toán không quá mạnh.
   2. **LoRA** xuất hiện như một giải pháp hiệu quả để giải quyết các vấn đề này. LoRA cho phép giảm đáng kể tài nguyên cần thiết để fine-tune các model lớn mà vẫn giữ được chất lượng cao về kết quả thu được. Thay vì cập nhật toàn bộ trọng số của mô hình, LoRA chỉ điều chỉnh các biến đổi hạng thấp (low-rank transformations), giúp giảm thiểu bộ nhớ sử dụng và thời gian lẫn về độ khó tính toán.
   3. Công dụng: LoRA dùng để fine-tune các mô hình có tham số lớn, từ vài trăm triệu đến hàng tỷ tham số lớn, mà tài nguyên tính toán cần thiết dành cho fine-tune lại giảm đi rất nhiều, giúp tiết kiệm tài nguyên tính toán cho người dùng, từ đó fine-tune dễ dàng tiếp cận và phổ biến với người dùng phổ thông.
2. Lý thuyết: finetune là gì?
   1. Machine Learning (tạm dịch: Máy học): **Học máy** hay **máy học** (tiếng Anh: machine learning) là một lĩnh vực của trí tuệ nhân tạo liên quan đến việc nghiên cứu và xây dựng các kĩ thuật cho phép các hệ thống "học" tự động từ dữ liệu để giải quyết những vấn đề cụ thể [1]. Có thể hiểu đơn giản là máy tính sẽ lưu trữ dữ liệu đã phân tích được bằng một thuật toán nào đó dưới dạng số, matrix, nhị phân,...để sau này có thể lấy dữ liệu đã phân tích trước đó ra và áp dụng vào trường hợp, tác vụ cụ thể.
   2. Transfer Learning (tạm dịch: học chuyển giao): là một kỹ thuật máy học (Machine Learning), trong đó **mô hình đã đào tạo trước** (pre-trained model) về một tác vụ nào đó (ví dụ như một model nhận diện xe) sẽ được tinh chỉnh cho một tác vụ mới, có liên quan (như nhận diện cây chẳng hạn). Đào tạo lại từ đầu một mô hình mới là một quá trình tiêu tốn rất nhiều thời gian và cần các kiến thức chuyên sâu, đòi hỏi một lượng lớn dữ liệu, sức mạnh điện toán, tính toán từ thiết bị điện tử và một số lần lặp đi lặp lại trước khi cho ra kết quả tốt nhất, được đem ‘xuất bản’(publish) (Note là mấy model hiện tại điều vậy, và nếu ai đã từng thử cũng vậy, phải train đi train lại nhiều thì model mới ‘ngon’ được, trừ ăn hên ra, 1 phát xong luôn model). Thay vào đó, các tổ chức lớn hiện nay sử dụng transfer learning để đào tạo lại các mô hình hiện có về các nhiệm vụ liên quan với dữ liệu mới, để đỡ tốn thời gian và chi phí. Ví dụ: nếu một mô hình máy học có thể xác định hình ảnh của chó, nó có thể được đào tạo để nhận diện mèo bằng cách sử dụng một bộ dataset ít hình ảnh hơn về mèo, vậy thì mô hình đó sau khi train transfer learning thì sẽ nhận diện được mèo mà không cần tốn quá nhiều thời gian cho việc train cũng như điện toán.
   3. Fine-tuning:

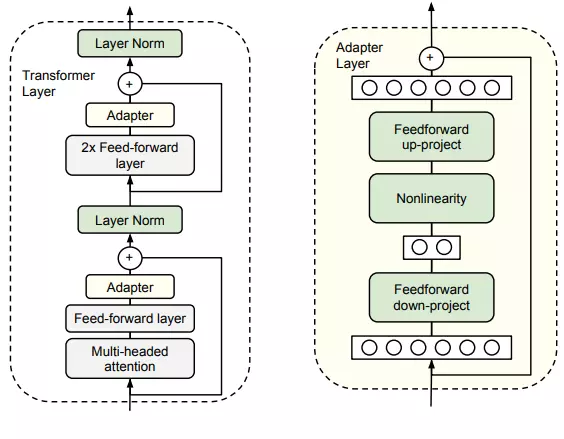
* **Fine-tuning** là một kỹ thuật trong **transfer learning** (học chuyển giao), nơi bạn sử dụng trọng số (weights) của một **pre-trained model** (mô hình đã được huấn luyện sẵn) để tiếp tục huấn luyện trên một bộ dữ liệu (dataset) mới, thường có kích thước nhỏ hơn và phục vụ cho một mục tiêu cụ thể hơn. Pre-trained model thường sẽ là model được train trên dataset lớn, với độ chính xác cao sẵn. Khi fine-tune, thay vì khởi tạo model từ đầu, ta chỉ cần điều chỉnh (tune) các trọng số của pre-trained model, từ đó giúp tăng độ chính xác cho task cụ thể đó mà không cần phải thu thập và train trên dataset mới từ đầu nữa.
* Thông thường, khi thực hiện fine-tuning, ta sẽ phải train một số hoặc toàn bộ layers của model, và cũng phải lưu lại toàn bộ tham số hoặc một số layers của model được fine-tune luôn. Quá trình fine-tuning chỉ ảnh hưởng đến một số layers của model. Thường thì các layers ở gần đầu của model học các đặc trưng tổng quát (như nhận diện hình khối, cạnh) và ít khi cần thay đổi. Trong khi đó, các layers gần cuối thường học các đặc trưng cụ thể hơn cho tác vụ cụ thể và cần được huấn luyện lại.
* Trước khi đến ví dụ, ta cùng tìm hiểu sơ bộ về downstream tasks: **Downstream tasks** là các nhiệm vụ hoặc tác vụ cụ thể mà một mô hình đã được huấn luyện sẵn (pre-trained model) sẽ đảm nhận sau khi được điều chỉnh (fine-tune) cho mục tiêu cụ thể đó.
  + Nói dễ hiểu: **Upstream**: Là quá trình huấn luyện ban đầu trên một bộ dữ liệu lớn và đa dạng để tạo ra 1 pre-trained model. **Downstream**: Là các tác vụ nhỏ hơn, cụ thể hơn mà mô hình sẽ được fine-tune để sử dụng, chẳng hạn như phân loại hình ảnh hoa, dịch ngôn ngữ, nhận diện khuôn mặt, hoặc các nhiệm vụ đặc thù khác.
  + Ví dụ: Giả sử bạn có một mô hình AI đã được huấn luyện trước trên một bộ dữ liệu hình ảnh lớn để phân biệt các loại vật thể nói chung như xe hơi, mèo, chó,... Sau đó, bạn muốn dùng model này phân loại các loài chim thì việc phân loại chim là một **downstream task**. Bạn sẽ fine-tune mô hình tổng quát ban đầu để nó có thể tập trung làm tốt nhiệm vụ này.
  + -> Mỗi downstream task sẽ cần mô hình được điều chỉnh (fine-tune) một cách riêng biệt, vì dữ liệu và yêu cầu cụ thể cho mỗi tác vụ không giống nhau.
* Ví dụ: Nếu bạn có 10 downstream tasks khác nhau, bạn sẽ phải fine-tune model cho mỗi task, và điều này đồng nghĩa với việc train 10 lần và lưu lại weights cho từng model đã fine-tuned. Tuy nhiên, với các mô hình nhỏ, điều này không gây ra vấn đề lớn, vì đơn giản là nó nhỏ, nó nhẹ. Nhưng khi finetune các mô hình lớn như Stable Diffusion hay LLaMA – vốn chứa từ hàng trăm triệu đến vài tỷ tham số – việc train lại toàn bộ mô hình và lưu lại mỗi phiên bản là một thách thức lớn về mặt phần cứng và lưu trữ.
* Một số thông tin: SDXL: 2.6B, Llama-3.1: 8B, 70B; MistralNemo: 12B (Hình ảnh hoặc thông tin thêm về nó sẽ thêm vô sau.)
* Giả sử bạn có một mô hình đã được huấn luyện sẵn để nhận diện các đối tượng như mèo, chó, xe hơi,... Bạn muốn sử dụng mô hình này để phân loại các loài hoa thì thay vì huấn luyện mô hình từ đầu bằng dataset hoa, ta chỉ cần fine-tune mô hình hiện có bằng cách điều chỉnh các layers gần cuối để nó có thể phân biệt các loài hoa chính xác hơn. Những layers đầu tiên vẫn giữ nguyên vì chúng đã học được các đặc trưng tổng quát chính xác, ‘tốt’ rồi.
* Lưu ý nhỏ là: không có model nào có độ chính xác 100% hết, dù 90 mấy % thôi cũng có thể là overfitting (hiện tượng kết quả nó quá khớp (khớp meme) với tập dữ liệu đưa ra)
* Lưu ý 2: Khi fine tuning model, bạn sẽ phải gặp trường hợp Forgetting Catastrophe (tạm dịch: Lãng quên thảm họa), là trường hợp là model sẽ quên luôn task cũ, chỉ nhớ mình task mới. Giống như ví dụ trên, bạn finetune xong từ model nhận dạng chó sang nhận dạng mèo chẳng hạn, thì khi bạn đưa hình ảnh con chó vô cho model nhận dạng, chưa chắc nó đã nhận dạng được, vì mấy layers mà quá trình fine-tune đã thay đổi giờ chỉ dành cho mèo.
* Quá trình fine-tuning:Pre-trained weights W của model sẽ biến thành updated weights W′ dựa trên giá trị weight cần thay đổi ΔW thu được từ quá trình backprop. Và ở iteration tiếp theo, W′ lại được update với một ΔW khác. (W,W’, ΔW là ma trận)



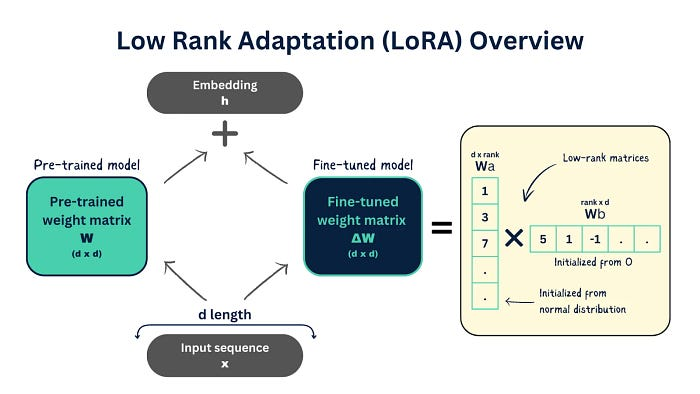


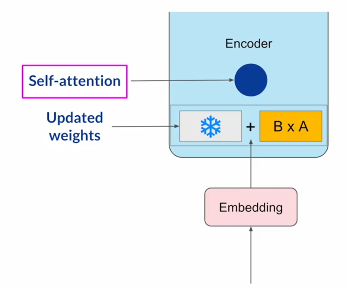


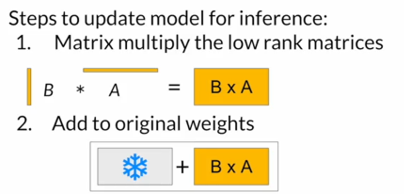
* Thay vì cập nhật toàn bộ trọng số W sau mỗi iteration, ta giữ nguyên trọng số đã được huấn luyện trước (freeze W) và chỉ cập nhật giá trị thay đổi ΔW. Thay vì thay đổi trực tiếp các trọng số của mô hình, ta chỉ lưu lại các thay đổi ΔW từ quá trình backpropagation. Trọng số mới sau mỗi lần lặp được tính bằng công thức: Wnew​=Wpretrained​+ΔWtotal​, trong đó ΔWtotal là tổng của tất cả các giá trị delta weight qua từng lần lặp.
  1. PEFT:
* Năm 2019, Neil Housby và các bạn của anh đã nghĩ ra một cách fine-tuning một cách hiệu quả, đó là Parameter-efficient Fine-tuning (PEFT): Ở mỗi khối Transformer trong model, ta chèn thêm vào 2 lớp Adapters trước khi thực hiện fine-tuning.[4]

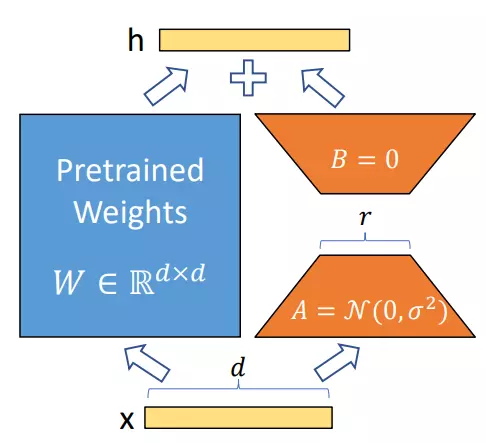


* Rồi sau đó chỉ cần train phần Adapter đã chèn vô model, phần còn lại của model thì giữ nguyên => giảm đáng kể số parameter cần train => giảm tài nguyên cần khi finetune về VRAM. Ngoài ra, việc finetune adapter như vầy còn giúp bạn **tiết kiệm dung lượng ổ cứng** khi chỉ cần lưu **weights của Adapters.** Và thêm 1 điều nữa là Adapter, như tên gọi của nó, chỉ cần tháo ra, lắp vô khi cần thiết, nên nó không gây ra hiện tượng Forgetting Castatrophe của Fine-tuning đã được nhắc ở trên.[5]
  + Lưu ý: Không phải Adapter được train từ model này cũng có thể dùng cho nhiều model khác và ngược lại, không phải Model nào cũng có thể dùng Adapter được train từ model khác.
  + Lưu ý 2: có thể dùng nhiều Adapter khác nhau cho cùng 1 model, cùng lúc.
  + Adapter giúp giảm thiểu được VRam cần thiết cho fine-tuning, nhưng độ ‘nặng’ về tính toán vẫn còn, dù đã thấp hơn fine-tuning truyền thống, nên tính ra thì fine-tuning vẫn còn ‘kén máy’. Năm 2021, một team của Microsoft, dẫn đầu là Edward Hu and Yelong Shen, đã tạo ra một phương pháp PEFT hiệu quả hơn, Low Rank Adapter.
  1. LoRA:
* Vừa nãy bạn đã ‘chứng kiến’ cách mà fine-tune hoạt động, LoRA nhắm đến việc biểu diễn ΔW thành một dạng có số chiều ít hơn, đơn giản hơn, nhẹ hơn. LoRA chọn sử dụng Matrix decomposition để biểu diễn ma trận ΔW bằng tích của các ma trận con với độ nặng tính toán thấp hơn việc tính trên ma trận gốc. Có rất nhiều phương pháp Matrix decomposition (LU decomposition, Singular Value Decomposition, ...), và LoRA chọn sử dụng phương pháp Neural Network

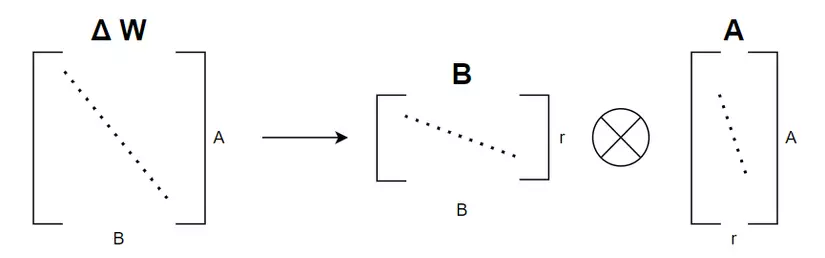








* Lora tách ma trận ΔW ban đầu thành 2 ma trận con A và B với số rank thấp hơn so với ban đầu. Sau khi tính toán, output của nó sẽ như vầy:
* *y*=*W*0​*x*+Δ*Wx*=*W*0​*x*+*BAx*
* Nhức đầu chưa? Nói về bước khởi tạo ma trận đi: Ma trận A thì khởi tạo bằng random Gaussian init (tra google thêm đi), B thì toàn 0. Nên ban đầu tính ra là toàn 0 cho ΔW, rồi cứ tính đi tính lại cho đến khi tìm ra ΔW tối ưu.
* Đến đây thì mình sẽ giải thích sao mà ma trận ΔW nó nhỏ đi sau khi tách ra rồi ghép lại: ΔW ban đầu là ma trận AxB là 512x64 (ví dụ vậy), thì sẽ có 32768 parameter có thể train, chỉ mới 1 layer, thấy ngán rồi. Giờ tách ra với rank là 8 thì sẽ có ma trận mới là A là 512x8= 4096 và B là 64x8=512, cộng lại chỉ 4096+512=4608, đâu đó chỉ còn 4096\*100/32768=12.5% so với ban đầu, yep, ngon rồi đó.



* Vậy là bạn đã hiểu ‘sơ sơ’ cách nó hoạt động, nói thêm tại sao nó ngon hơn Adapter, không bị chậm: LoRA áp dụng thêm Re-parameterize (search google tiếp nhe, cái này viết nữa dài lắm). Nói chung trong quá trình fine-tune thì vẫn như nãy nói, cũng ‘khá’ chậm, nhưng train xong thì mấy anh tác giả nghĩ ra cách Re-parameterize đơn giản nhưng hiệu quả:
* Trước Re-parameterize:*y*=*Wx*+*BAx*
* Sau Re-parametereize:*y*=(*W*+*BA*)*x*

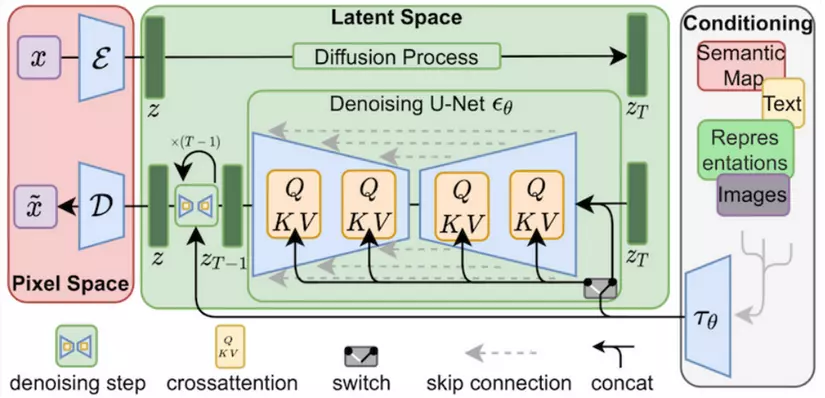
Done, giờ còn nói sơ stable diffusion với LLM.

1. Stable Diffusion, LoRA trong Stable Diffusion
2. Giới thiệu sơ lược: Stable Diffusion là một mô hình học sâu (deep learning) dùng thuật toán diffusion (khuyết tán) để tạo hình ảnh từ văn bản (text2img, đơn giản là txt2img). Mình sẽ không nói hết về diffusion, vì nếu làm vậy thì lạc đề, dài dòng. Mình sẽ chỉ tập trung vô LoRA trong SD ra sao, giờ chỉ nói sơ cho các bạn nắm cách hoạt động thôi, các bạn tra google thêm nhe.
3. Kiến trúc: Sơ bộ thì SD sẽ có 5 thành phần chính trong model:

* Unet (Noise predictor): Công cụ dự đoán nhiễu, dùng Mạng nơ ron dư (ResNet) để ước tính lượng nhiễu trong không gian ngầm (latent space) và loại bỏ nó khỏi hình ảnh. Unet sẽ lặp lại quá trình này theo một số lần nhất định (tí nữa demo bạn sẽ hiểu thêm về nó), giảm nhiễu theo các bước do người dùng chỉ định. Unet thì khá nhạy cảm với prompt, thứ giúp xác định output cuối cùng mà người dùng mong muốn.
* Text encoder: SD dùng CLIP tokenizer cho text encoder, sẽ phân tích từng từ trong prompt và nhúng nó vô vector 768 giá trị, mỗi đoạn prompt dài 75 token, rồi dùng vector này đưa vô Unet để tạo hình.
* Variational autoencoder (VAE): cộng đồng SD hay gọi chơi cái này là model màu, vì thiếu nó model của bạn sẽ output ra hình với màu sắc…không ổn tí nào, nhưng tác dụng thiệt của nó rất khác. Với SD 1.5, train từ dataset hình 512x512, thì VAE sẽ có 2 công dụng: encoder sẽ nén hình 512x512 pixels lại thành model nhỏ 64x64 để dễ lưu trữ, thao tác, decoder thì giải nén từ model 64x64 đó sang hình 512x512 pixel để bắt đầu tạo sinh.
* Forward Diffusion (Khuếch tán thuận): Từng bước thêm nhiễu vào hình ảnh một cách ngẫu nhiên
* Reverse Diffusion (Khuếch tán nghịch): Sử dụng UNet để loại bỏ nhiễu từ quá trình Forward Diffusion và dần tái tạo lại hình ảnh.

1. LoRA trong SD:

* Vào 1 ngày đẹp trời 2022, khi mà mọi người trong cộng đồng vẫn còn đang loay hoay với hypernetwork và text embedding để tạo ‘style’, ‘concept’,...nhưng chất lượng không tốt lắm, thì từ đâu nổi lên một cách finetune adapter mới cho SD, giúp cho ra chất lượng hình ảnh ‘ngon’ hơn 10 lần so với mấy cái kia, mà thời gian lại ngắn hơn. Ừ, đó là LoRA, nó khiến cộng đồng chơi generate hình từ nhỏ nhỏ trở nên bùng nổ chưa từng có. (và cũng khiến AI trở nên xấu hơn trong mắt nhiều người, theo cái hướng mà khó kiểm soát được.)
* Mình sẽ nói về cách LoRA hoạt động ra sao trong SD:
  + LoRA sẽ được ‘chèn’ vô trong Text Encoder và Unet, có nghĩa là, bạn sẽ có thể fine tune 1 trong hai hoặc fine-tune đồng thời 2 thứ: ký tự và hình ảnh, tùy theo mục đích sử dụng.
  + Như đã nói trước đó: Text Encoder sẽ đóng vai trò phân tích và lưu trữ prompt, trước khi gửi vô trong Unet để bắt đầu tạo ảnh theo prompt mà người dùng nhập. Việc fine-tune Text Encoder của model giúp LoRA adapter sẽ lưu trữ được những gì bạn ghi trong caption của từng hình trong dataset, từ đó khi bạn gọi 1 prompt có ghi trong caption đó, LoRA sẽ kích hoạt và chèn weight vô trong Text Encoder của model và tiếp tục quá trình tiếp theo.



* Tiếp theo là Unet, thì LoRA sẽ được chèn vô các model Unet trong SD. UNet sử dụng cơ chế self-attention, gồm 3 ma trận ‘nhỏ’: WQ (Query), WK (Key), WV (Value), và LoRA sẽ được chèn vô trong đây và tính toán lại các ma trận này rồi lưu trữ lại giá trị cho hợp lý.
* Cuối cùng, adapter LoRA được lưu lại với dạng .safetensors (Một định dạng an toàn của HuggingFace, không thể modify cấu trúc để thực hiện mục đích xấu) và chứa thông tin về weight Unet và Text Encoder đã train. File khá nhỏ, chỉ từ vài MB đến hơn vài trăm MB, so với model SD gốc (SD 1.5 sẽ là 4gb cho model fp32, còn SDXL sẽ là ~12GB cho model fp32)

Tham khảo:

[1] [Học máy – Wikipedia tiếng Việt](https://vi.wikipedia.org/wiki/H%E1%BB%8Dc_m%C3%A1y)

[2] [Học chuyển tiếp là gì? – Giải thích về Học chuyển tiếp trong Máy học – AWS](https://aws.amazon.com/vi/what-is/transfer-learning/)

[3] [Fine-tuning một cách hiệu quả và thân thiện với phần cứng: Adapters và LoRA](https://viblo.asia/p/fine-tuning-mot-cach-hieu-qua-va-than-thien-voi-phan-cung-adapters-va-lora-5pPLkj3eJRZ)

[4] [Parameter-Efficient Fine Tuning with Low Rank Adaptation (LoRA) for Large Language Models | by Sayan Das | Medium](https://medium.com/@das.sayan1025/parameter-efficient-fine-tuning-with-low-rank-adaptation-lora-for-large-language-models-3a3b8e15d516)

[5] [What is parameter-efficient fine-tuning (PEFT)?](https://www.ibm.com/think/topics/parameter-efficient-fine-tuning)

[6] [Fine Tuning LLM: Parameter Efficient Fine Tuning (PEFT) — LoRA & QLoRA — Part 1 | by A B Vijay Kumar - Freedium](https://freedium.cfd/https://abvijaykumar.medium.com/fine-tuning-llm-parameter-efficient-fine-tuning-peft-lora-qlora-part-1-571a472612c4)

[7] [Efficient Model Tuning: Exploring PEFT, LoRA, and Other Techniques | by Kshitij Darwhekar - Freedium](https://freedium.cfd/https://medium.com/@kshitijdarwhekar/parameter-efficient-fine-tuning-peft-268dd8c99692)

[8] [Finetuning Stable Diffusion hiệu quả với LoRA](https://viblo.asia/p/finetuning-stable-diffusion-hieu-qua-voi-lora-aNj4vxeqL6r)

[9] [What is Stable Diffusion? - Stable Diffusion AI Explained - AWS](https://aws.amazon.com/what-is/stable-diffusion/?nc1=h_ls)