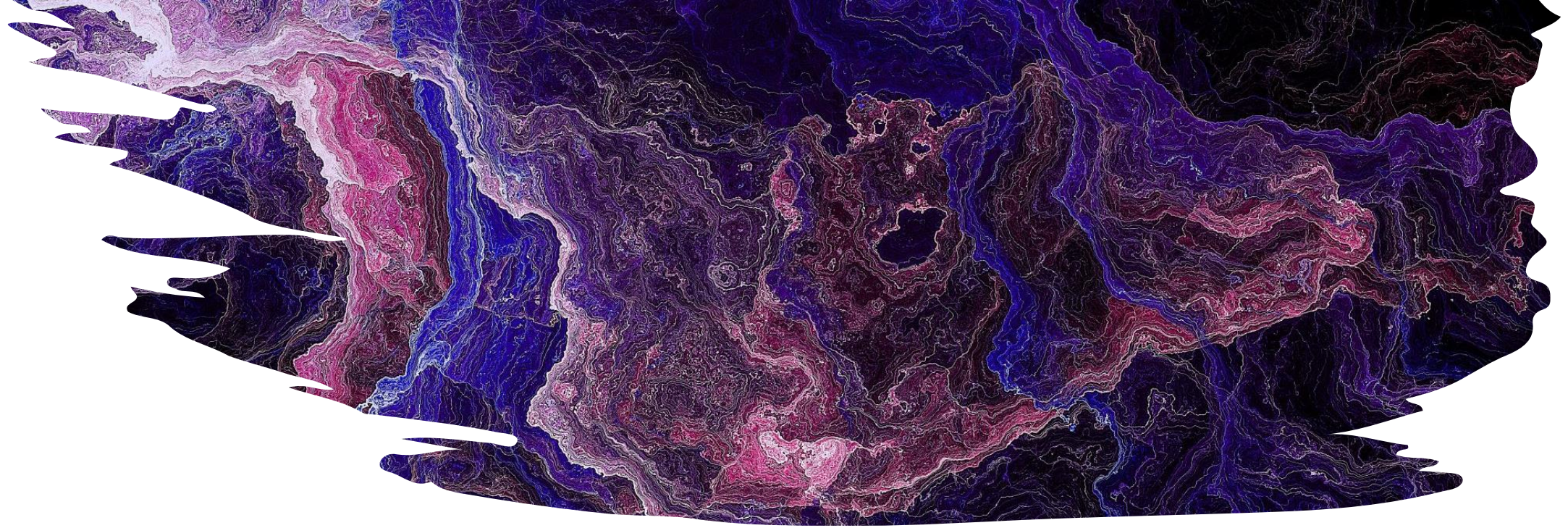


Fine-tuning de Large Language Models

Les diapos sont une version modifiée de :
https://www.slideshare.net/slideshow/finetuning-large-language-models-by-dmitry-balabka/266993288

Agenda

- Introduction au fine-tuning des LLMs
- LoRA
- QLoRA
- Formation pratique et mise en œuvre

Introduction au fine-tuning des LLMs

Apprentissage Zero/One/Few-shot

- Apprentissage Zero-shot
  - Signifie fournir un prompt qui ne fait pas partie des données d'entraînement
  - Exemple : poser des questions ouvertes au modèle
- Apprentissage One/Few-shot
  - Fournir un ou quelques exemples dans le prompt
  - Exemple : demander au modèle de formater le texte en fournissant quelques exemples
- Prompt engineering

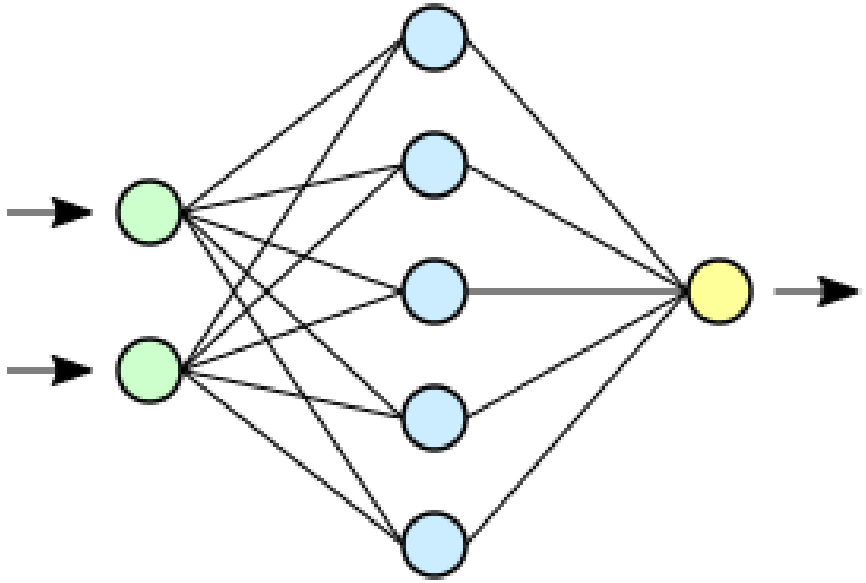
Quand devez-vous faire le fine-tuning?

- Prompt engineering n'a pas fonctionné.
- La génération augmentée par récupération (RAG) n'a pas fonctionné.
- Des données hautement **qualitatives** pour l'entraînement sont disponibles.
- Le coût n'est pas un problème.
- Il est clair comment évaluer le résultat.
- En savoir plus :
  - https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/fine-tuning-considerations
  - https://platform.openai.com/docs/guides/fine-tuning/when-to-use-fine-tuning

Qu'est-ce que le fine-tuning?

> En deep learning, le **fine-tuning** est une approche de l'apprentissage par transfert dans laquelle **les poids d'un modèle pré-entraîné** sont ajustés sur **de nouvelles données**.

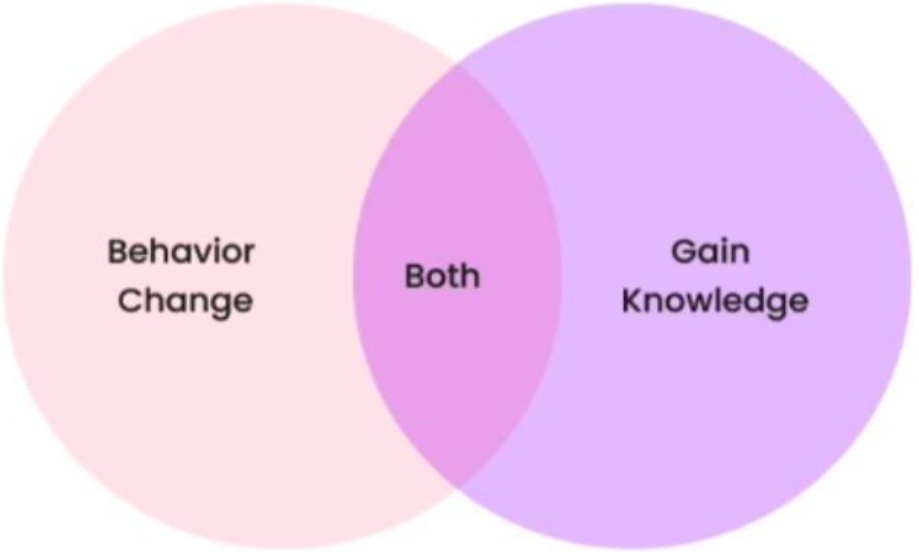
https://en.wikipedia.org/wiki/Fine-tuning_(deep_learning)



Impact du fine-tuning sur le modèle

Changement de comportement

- Répondre de manière plus **logique et cohérente.**
- Se spécialiser sur un sujet précis, comme la modération.
- étendre les compétences du modèle, comme mieux dialoguer.

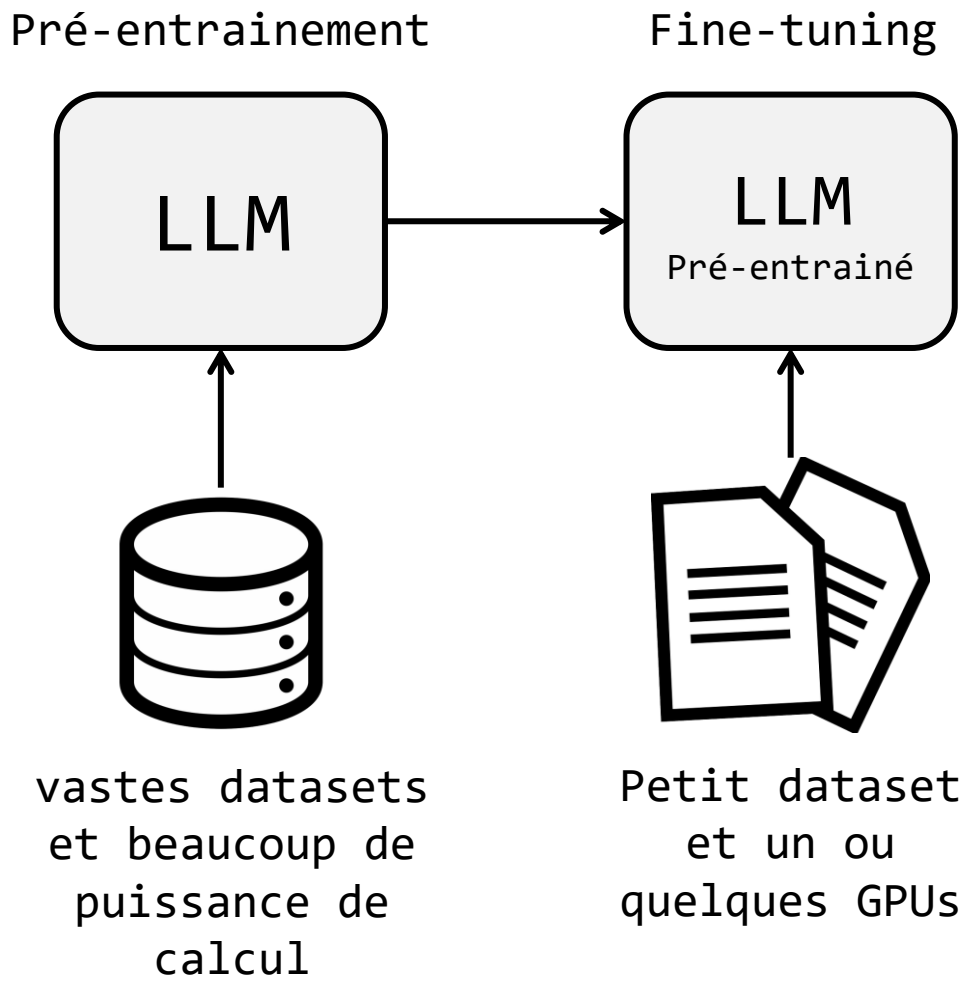


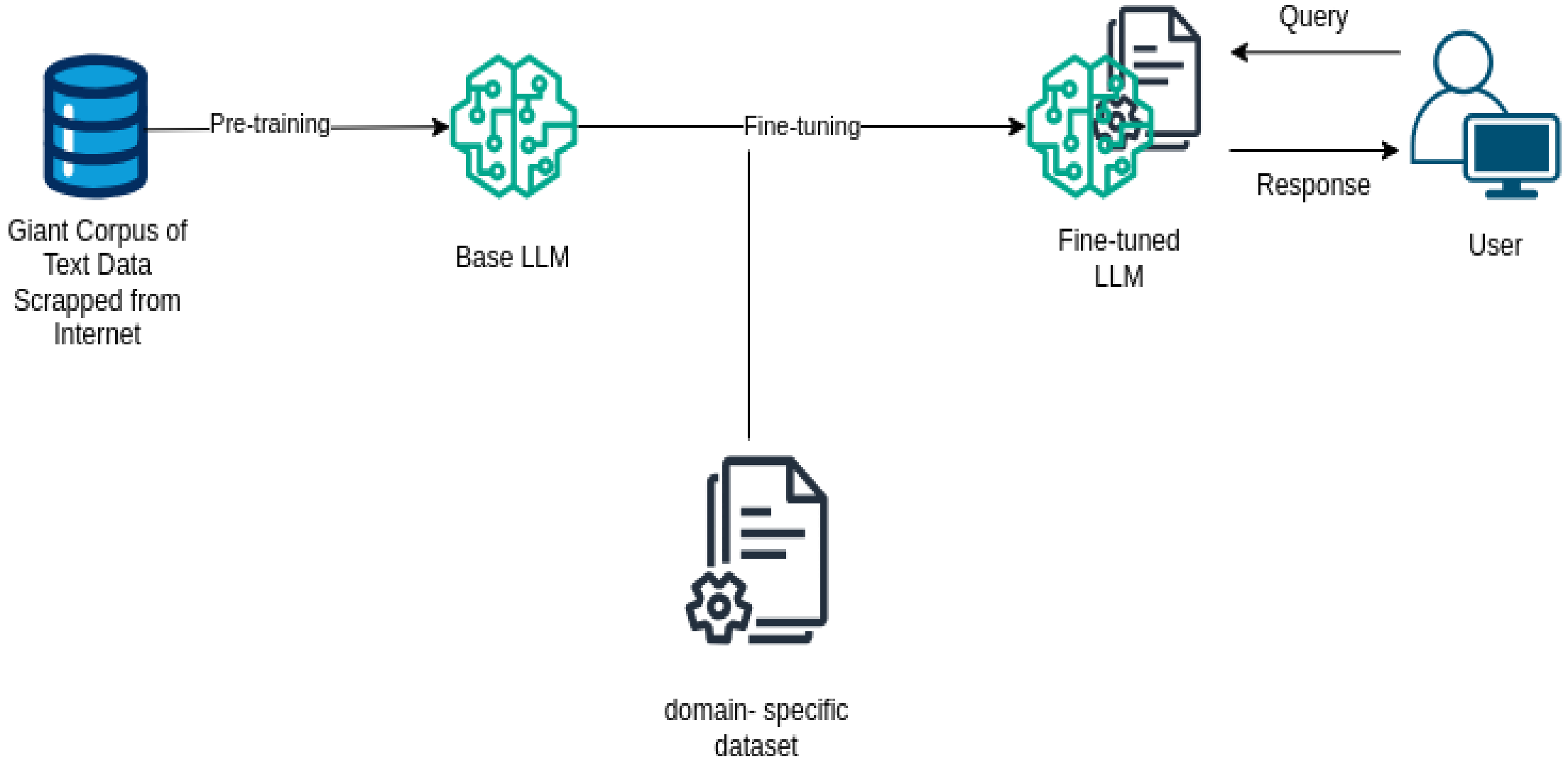
Acquisition de nouvelles connaissances

- Apprendre des informations spécifiques sur **un sujet donné**.
- Permet **de corriger des erreurs ou informations obsolètes** déjà présentes dans le modèle.
- **Réduction des hallucinations** : le modèle fait moins d'erreurs en inventant des faits.

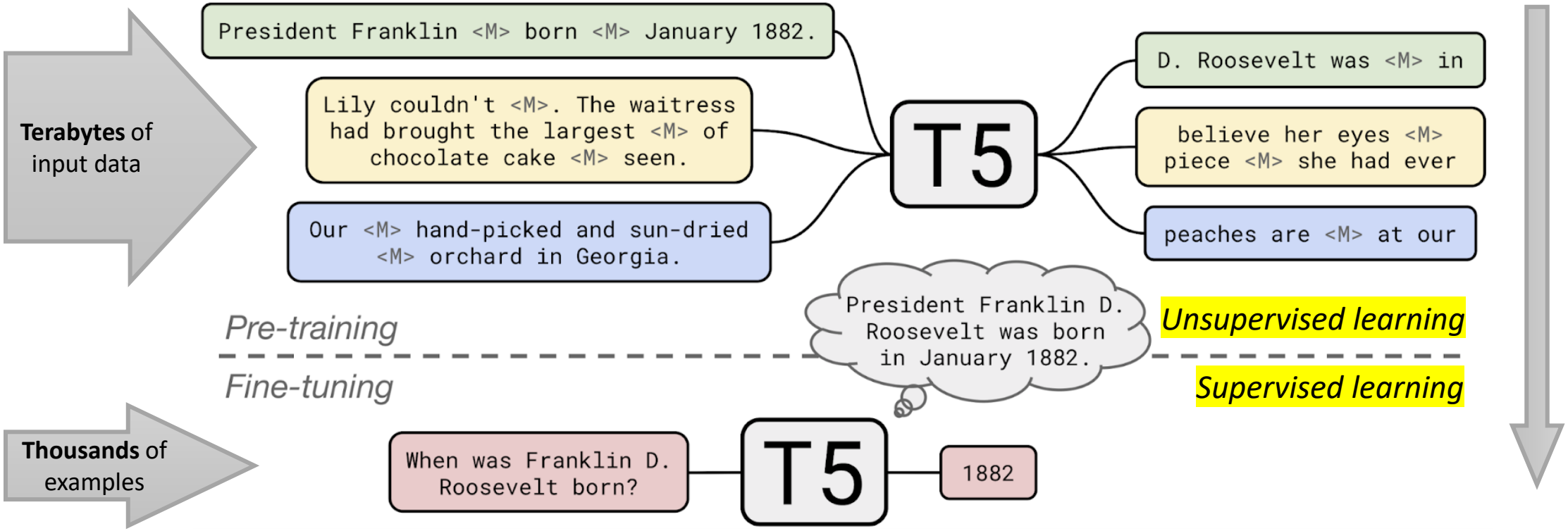
Différence entre le pré-entraînement et le fine-tuning 1/2

	Pré-entrainement	Fine-tuning
Temps d'entraînement	Semaines	Heures
Ressources	Milliers de GPUs	Un ou quelques GPUs
Dataset	Terabytes (e.g., C4, Pile)	100-1000 MB
Budget	$ Millions	$ Centaines




Pre-training
Fine-tuning
Query
Response
Giant Corpus of Text Data Scrapped from Internet
Base LLM
Fine-tuned LLM
User
domain- specific dataset

Différence entre le pré-entraînement et le fine-tuning 2/2



Source: https://blog.research.google/2020/02/exploring-transfer-learning-with-t5.html

LLMs open-source pré-entrainés

- Considérez les **licences** qui permettent des cas d'utilisation commerciaux.
- Un **LLM plus grand** possède de plus grandes capacités, mais il nécessite également plus de **ressources de calcul**.
- Une **fenêtre de contexte** plus grande permet d'ajouter plus d'informations dans le contexte.
- Les modèles attractifs :
  - Mistral avec 7B params, 4096 tokens et une fenêtre glissante de 16K, Licence Apache 2.0
  - Gemma avec 7B params, 8192 tokens, Conditions d'utilisation de Gemma de Google
  - Meta Llama2 7B.

En savoir plus: https://github.com/eugeneyan/open-llms

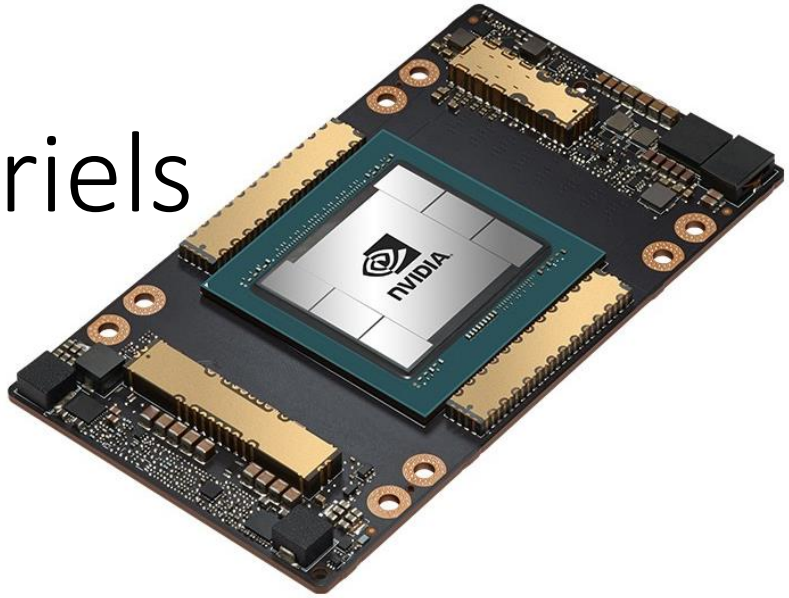
Bibliothèques pour le fine-tuning

Nom de la bibliothèque	Entreprise	Popularité ⭐	PEFT	DL Framework	Modèles LLM supportés	Liens
Deep Speed	Microsoft	31.5k	✅	PyTorch	A lot	docs, github
PEFT	HuggingFace 🤗	12.7k	✅	PyTorch	LLaMA, Mistral, T5, GPT, others	blog, github, docs
Accelerate	HuggingFace 🤗	6.6k	❌	PyTorch	A lot	github, docs
NeMo	Nvidia	9.4k	✅	PyTorch	LLaMA, Falcon, T5, GPT, others	docs, github
T5X	Google	2.3k	❔	JAX	T5 and some others, PaLM*	paper, github, docs
Paxml	Google	0.3k	❔	JAX	PaLM-2*	docs, github

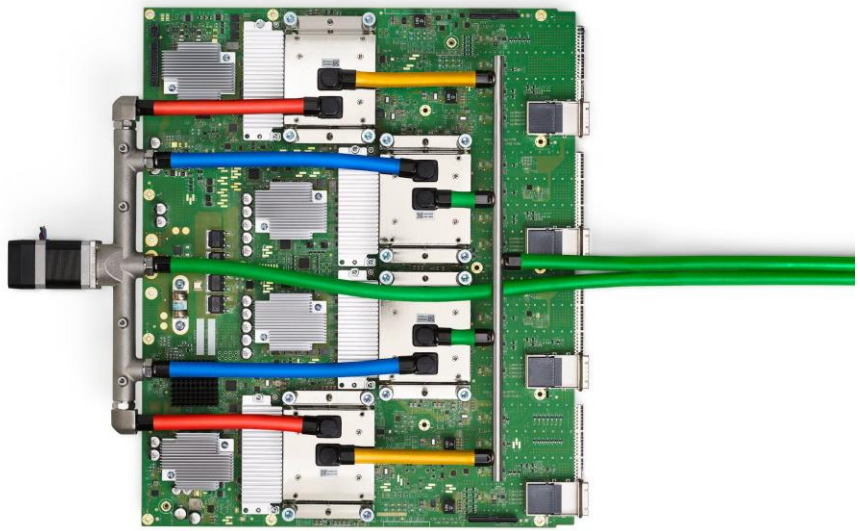
Fine-tuning supervisé dans le cloud

Cloud	LLM
Azure	GPT, Llama
AWS Bedrock	Amazon Titan, Anthropic Cloude, Cohere Command, Meta Llama [link]
GCP Vertex AI	PaLM 🌴, Gemma, T5, Gemini**, Llama
OpenAI Platform	GPT
Anthropic	Claude
Cohere	Command
MosaicML	MPT

Matériel pour les besoins industriels



- GPU Nvidia
  - H100 jusqu'à 80 Go de **RAM**
  - Prend en charge tous les **frameworks**
  - Disponible dans n'importe quel cloud
  - Nécessite NVLink/NVSwitch pour un **parallélisme** efficace des données/modèles
  - Possibilité sur site



- TPU Google
  - Plus rentable
  - V3-8 jusqu'à 128 Go de **RAM**
  - Supporte uniquement XLA : Jax, PyTorch/XLA, TF
  - Verrouillage GCP
  - Supporte le **parallélisme** des données/modèles prêt à l'emploi

En savoir plus: https://khairy2011.medium.com/tpu-vs-gpu-vs-cerebras-vs-graphcore-a-fair-comparison-between-ml-hardware-3f5a19d89e38

Méthodes de fine-tuning

- **Full fine-tuning :** met à jour tous les paramètres.
- **PEFT** (Parameter-Efficient Fine-Tuning) : met à jour un petit nombre de paramètres (**supplémentaires**) du modèle :
  - **Réduit les coûts computationnels et de stockage** de manière significative .
  - Donne des performances comparables à celles d'un modèle entièrement fine-tuné.
- **Méthodes populaires de PEFT** :
  - **LoRA** : insère une décomposition de bas rang de chaque matrice d'attention.
  - **QLoRA** : identique à LoRA mais avec un modèle quantifié."

En savoir plus : https://huggingface.co/docs/peft/index

Full Fine-Tuning est coûteux

Fine-Tuning de tous les paramètres

- Met à jour **tous les paramètres** → Nécessite une grande mémoire GPU
- Par exemple, coût du fine-tuning à 16 bits par paramètre
  - Poids : 16 bits (2 octets)
  - Gradient des poids : 16 bits (2 octets)
  - États de l'optimiseur : 65 bits (8 octets)
  - 96 bits (12 octets) par paramètre
  - Modèle de 65B -> 780 Go de mémoire GPU -> 17 GPU de centre de données (34 consumer GPU)

https://llmsystem.github.io/llmsystem2024spring/assets/files/Group2-Presentation-cf8028bc58193a5e6e6d7b05709ef1a9.pdf

Parameter Efficient Fine-tuning (PEFT)

- Mettre à jour **un petit sous-ensemble de paramètres**, sans dégrader la qualité du modèle
- Par exemple, coût du fine-tuning par paramètre avec **LoRA (Low Rank Adaptation)**
  - Poids : 16 bits
  - Gradient des poids : ~0,4 bits
  - État de l'optimiseur : ~0,8 bits
  - Poids de l'adaptateur : ~0,4 bits
  - 17,6 bits par paramètre
  - Modèle de 65B -> 143 Go de mémoire GPU -> 4 GPU de centre de données (8 consumer GPU)

https://llmsystem.github.io/llmsystem2024spring/assets/files/Group2-Presentation-cf8028bc58193a5e6e6d7b05709ef1a9.pdf

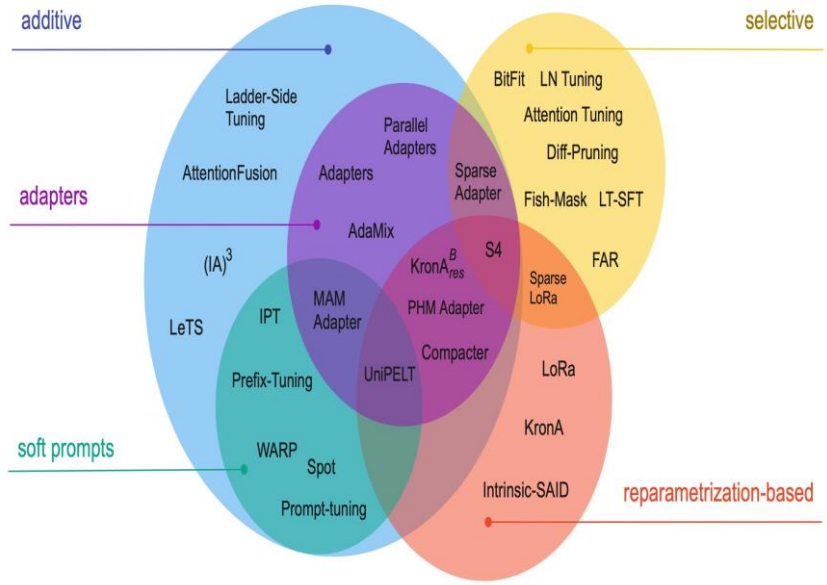
Parameter Efficient Fine-tuning (PEFT)

- Mettre à jour **un petit sous-ensemble de paramètres**, sans dégrader la qualité du modèle
- Par exemple, coût par paramètre avec **QLoRA (LoRA + Quantification du modèle)**
  - Poids : 4 bits
  - Gradient des poids : ~0,4 bit
  - État de l'optimiseur : ~0,8 bit
  - Poids de l'adaptateur : ~0,4 bit
  - 5,2 bits par paramètre
  - Modèle de 65B -> 42 Go de mémoire GPU -> 1 GPU de centre de données.

https://llmsystem.github.io/llmsystem2024spring/assets/files/Group2-Presentation-cf8028bc58193a5e6e6d7b05709ef1a9.pdf

Parameter Efficient Fine-tuning (PEFT)

- PEFT Trade-offs
  - Efficacité mémoire, efficacité des paramètres, performance du modèle, vitesse d'entraînement, coûts d'inférence
- PEFT Methods
  - **Méthodes sélectives :**
    - fine-tuner des sous-ensembles de paramètres
  - **Méthode de reparamétrisation :**
    - représentation de bas rang des poids du modèle
    - par exemple, LoRA
  - **Méthodes additives :**
    - ajouter des couches entraînables au modèle
    - par exemple, Adapters, Soft prompts (Prompt Tuning)



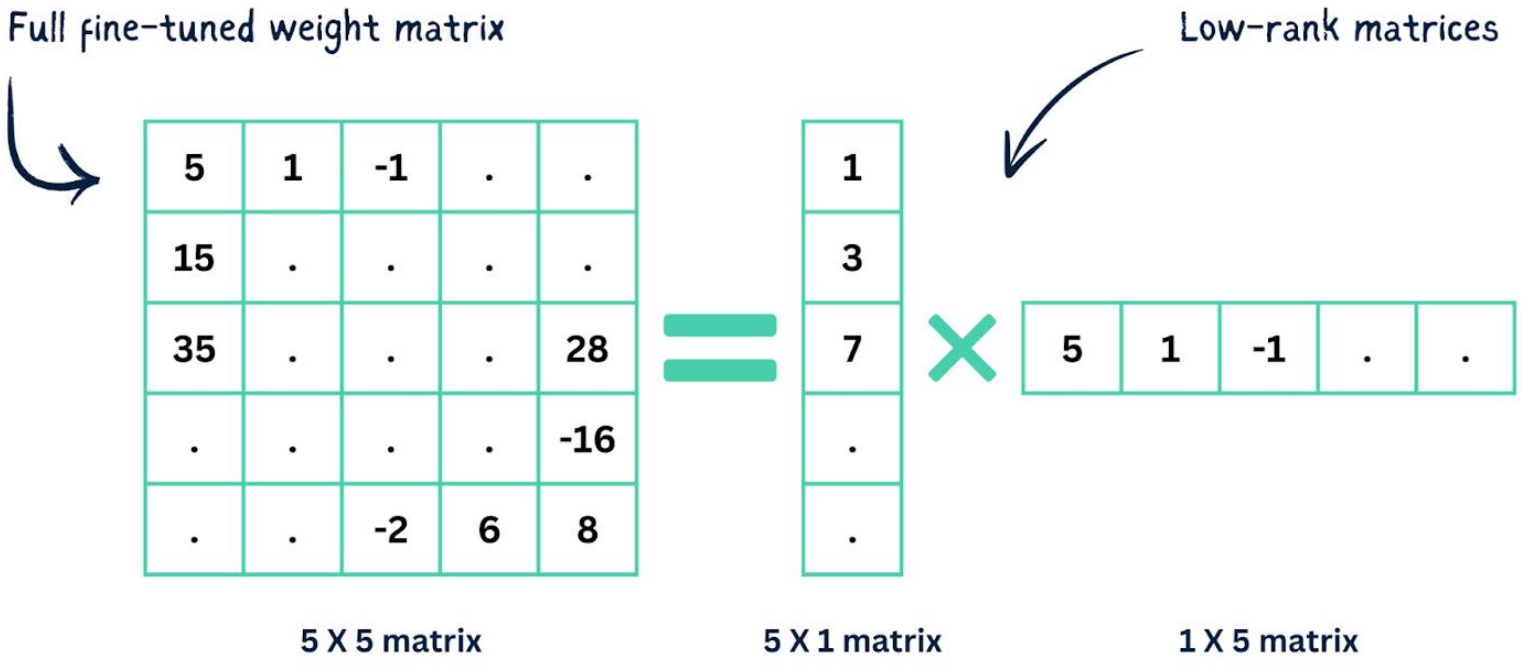
https://llmsystem.github.io/llmsystem2024spring/assets/files/Group2-Presentation-cf8028bc58193a5e6e6d7b05709ef1a9.pdf

LoRA et QLoRA pour la sélection des coefficients

- **LoRA (Low-Rank Adaptation) :**
  - Dans LoRA, la grande matrice de poids est divisée en deux matrices plus petites par factorisation.
  - Nous réduisons le nombre de coefficients qui nécessitent un ajustement, rendant le processus de fine-tuning plus efficace.
- **QLoRA (Quantification + Low-Rank Adoption) :**
  - La quantification consiste à convertir des coefficients à virgule flottante de haute précision en représentations de plus faible précision, comme une représentation sur 4 bits.
  - Exemple : un nombre à virgule flottante sur 32 bits peut être représenté comme un entier sur 4 bits dans une plage spécifique.
  - Cette conversion réduit considérablement l'empreinte mémoire.

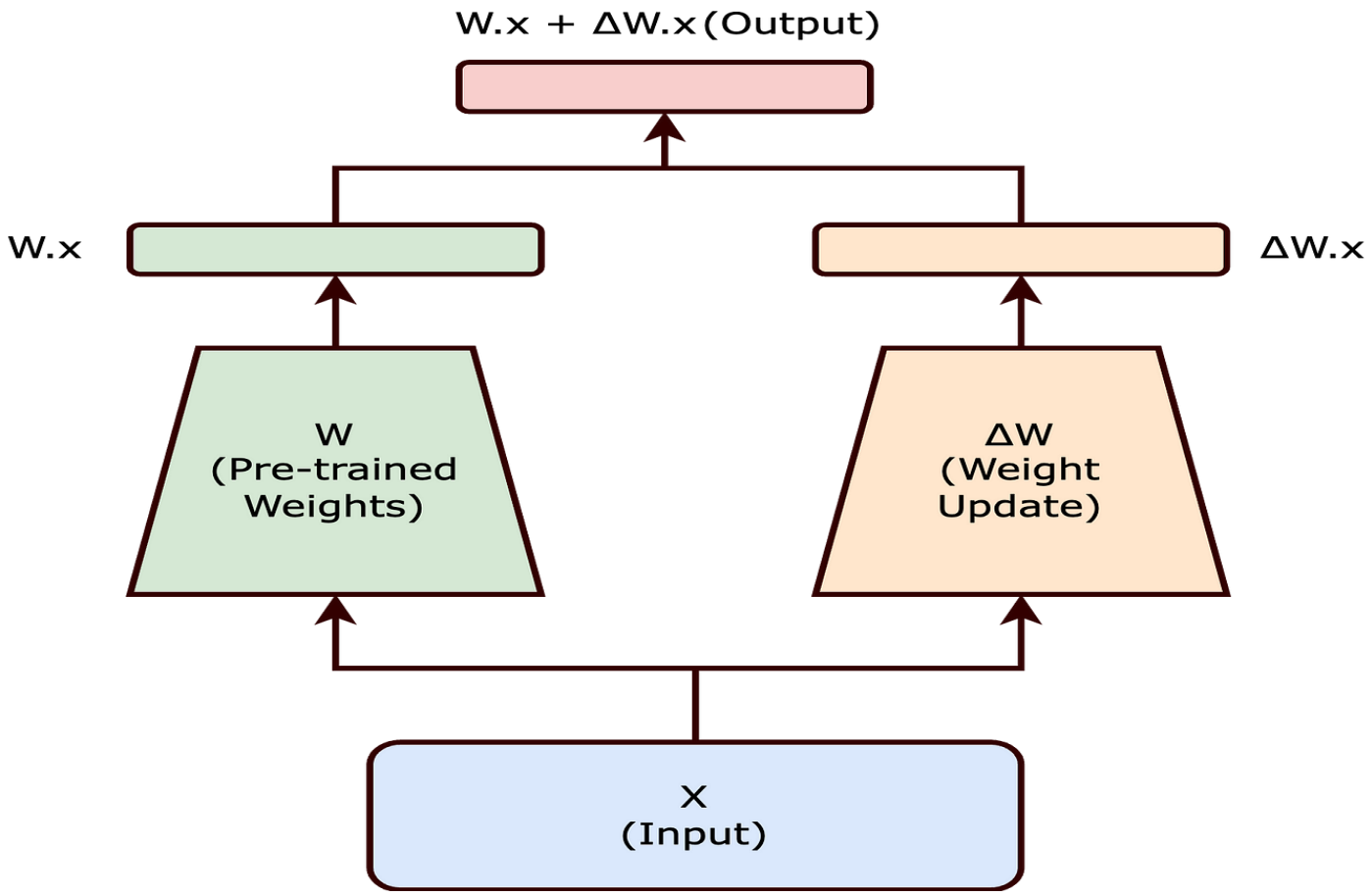
LoRA

Low-rank Matrix Decomposition



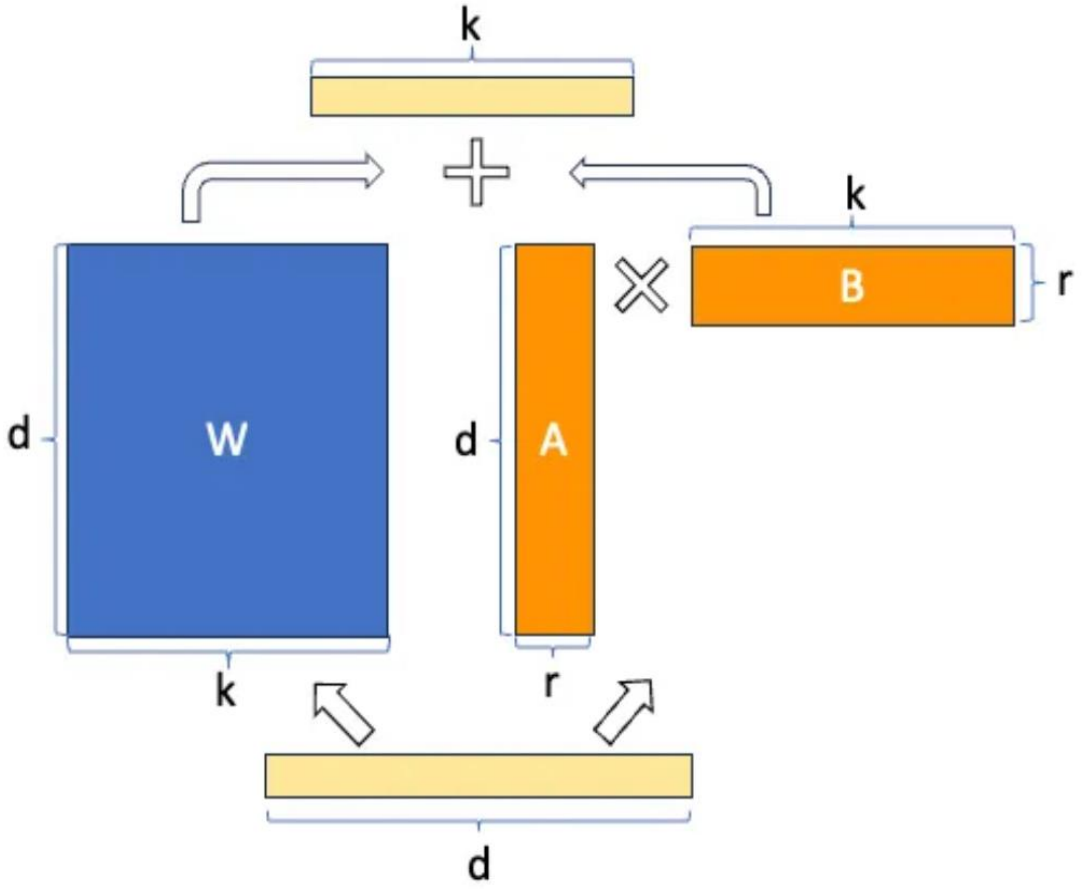
https://www.aporia.com/learn/low-rank-adaptation-lora/

LoRA: Low Rank Adaptation



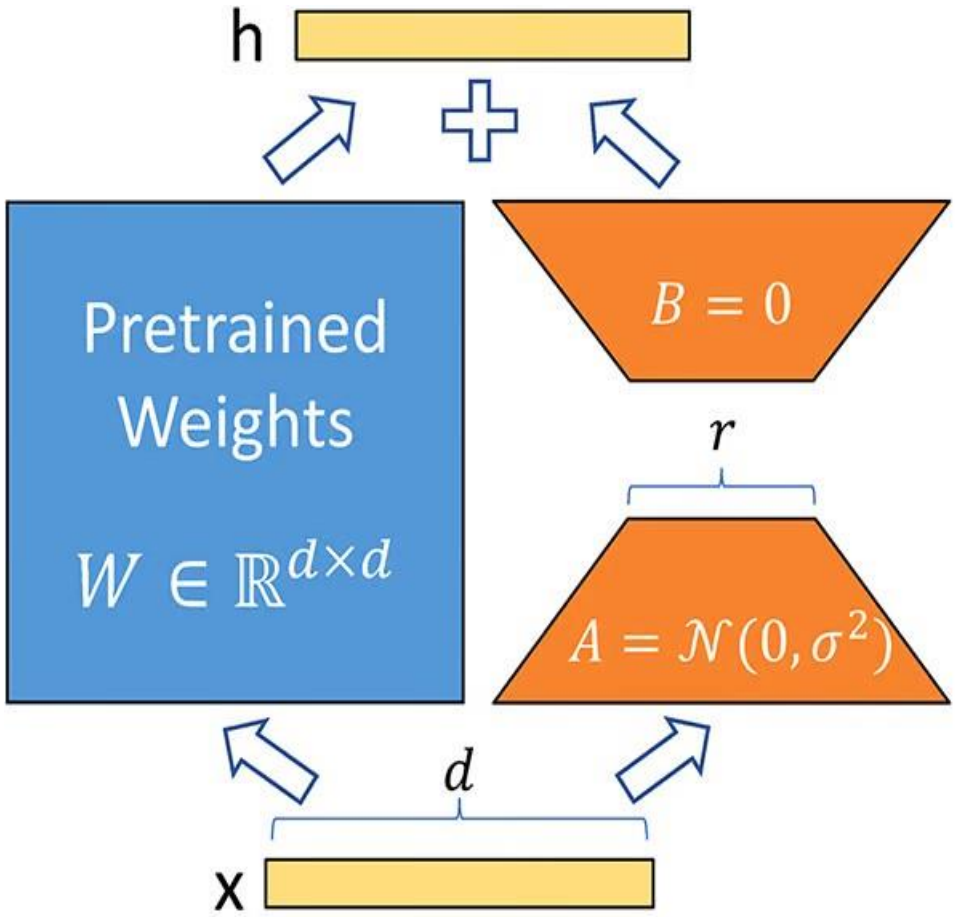
https://towardsdatascience.com/understanding-lora-low-rank-adaptation-for-finetuning-large-models-936bce1a07c6

LoRA: Low Rank Adaptation



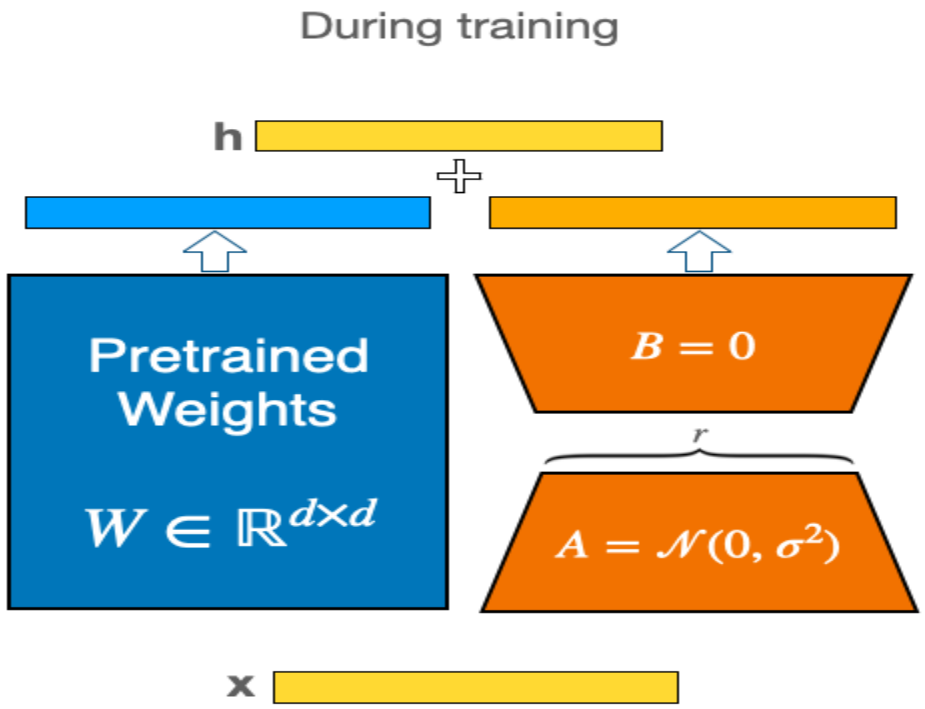
https://rocm.blogs.amd.com/artificial-intelligence/lora-fundamentals/README.html

LoRA: Low Rank Adaptation



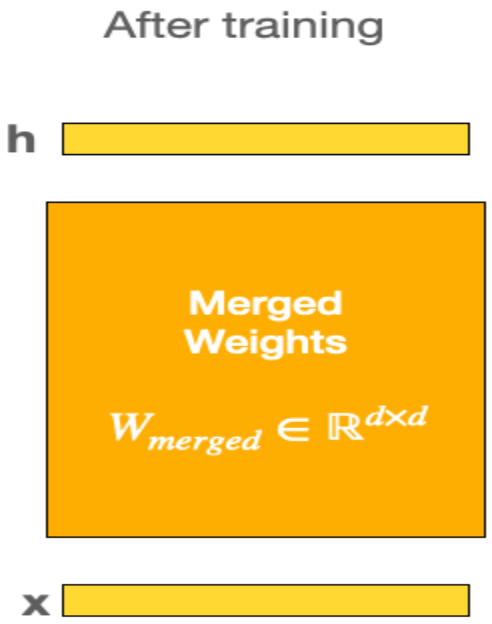
www.datacamp.com/tutorial/mastering-low-rank-adaptation-lora-enhancing-large-language-models-for-efficient-adaptation

LoRA: Low Rank Adaptation



$$h = Wx + BAx$$

$$h = \underbrace{(W + BA)}_{W_{merged}}x$$



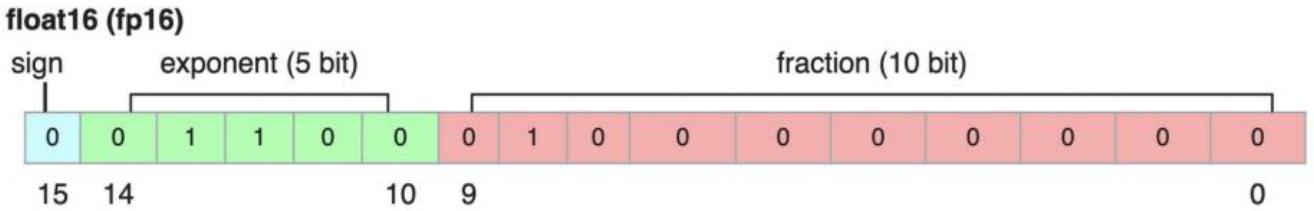
https://www.linkedin.com/pulse/lora-low-rank-adaptation-arjun-p-v-ljn4c/

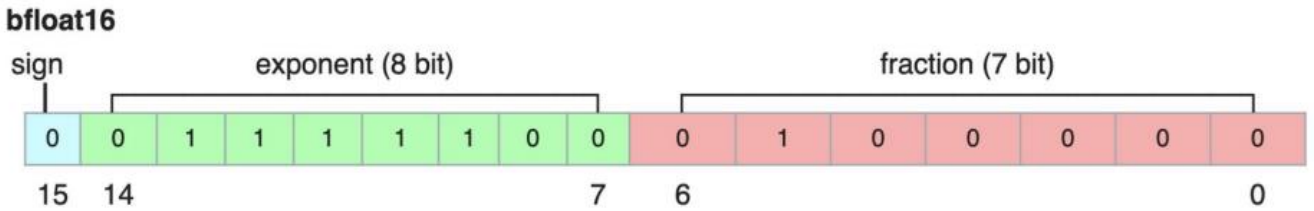
LoRA

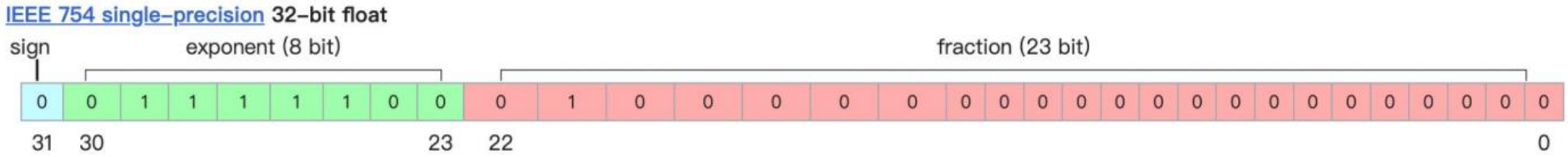
- **LoRA (Low-Rank Adaptation) :**
  - Dans LoRA, la grande matrice de poids est divisée en deux matrices plus petites par factorisation.
  - Nous réduisons le nombre de coefficients qui nécessitent un ajustement, rendant le processus de fine-tuning plus efficace.

QLoRA

Floating-point numbers:







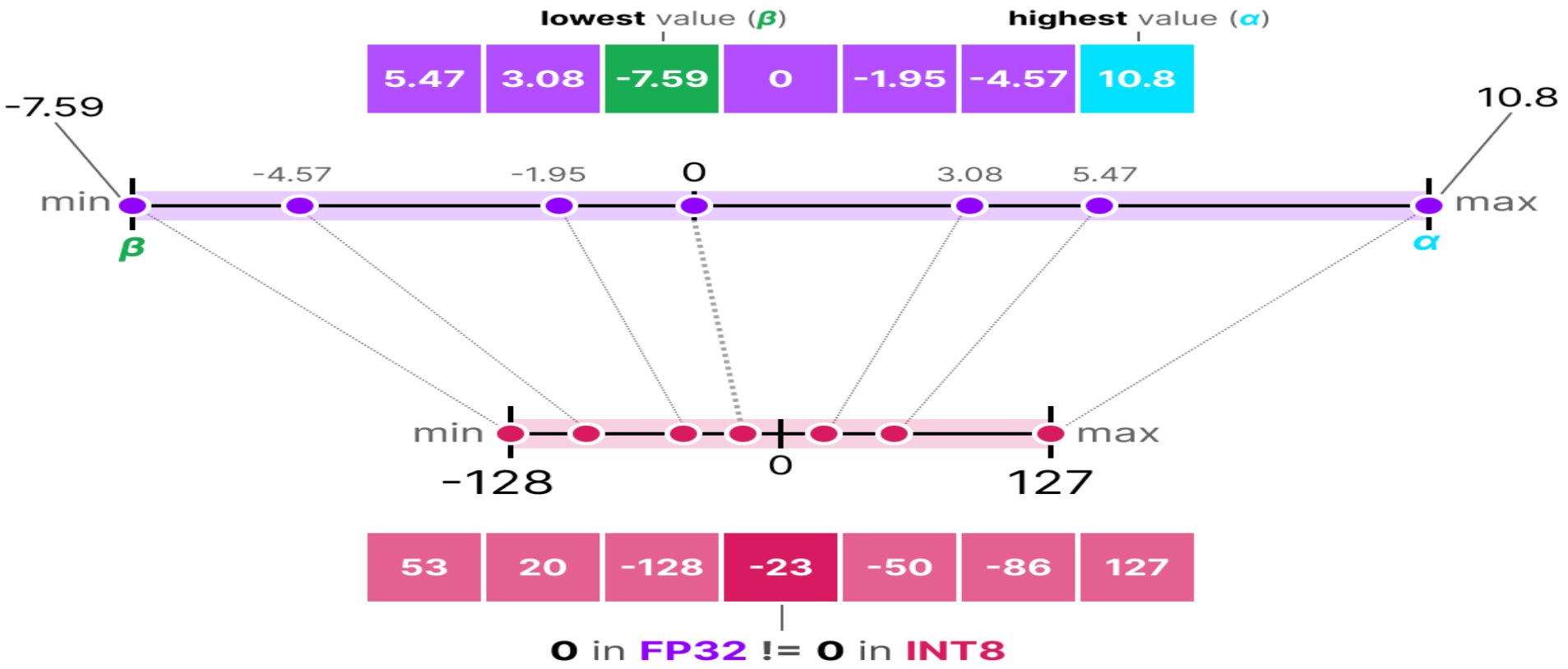
https://llmsystem.github.io/llmsystem2024spring/assets/files/Group2-Presentation-cf8028bc58193a5e6e6d7b05709ef1a9.pdf

QLoRA : LoRA + quantification du modèle

- Innovations majeures :
  - Float Normal 4 bits
  - Double quantification
  - Optimiseur de pages
- Type de données de stockage en 4 bits
- Type de données computationnel Bfloat16
- **Préservation des performances** comparées à un modèle entièrement fine-tunée en 16 bits.

Quantification du modèle

Exemple : utiliser le Maximum Absolu (absmax) pour quantifier un tenseur 32 bits Point Flottant(FP32) en un tenseur Int8 avec une plage de valeurs [-127,127] :



https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Quantification du modèle

Exemple : utiliser le Maximum Absolu (absmax) pour quantifier un tenseur 32 bits Point Flottant(FP32) en un tenseur Int8 avec une plage de valeurs [-127,127] :

$$\mathbf{X}_{\text{quant}} = \text{round}\left(\frac{127}{\max|\mathbf{X}|} \cdot \mathbf{X}\right)$$

$$\mathbf{X}_{\text{dequant}} = \frac{\max|\mathbf{X}|}{127} \cdot \mathbf{X}_{\text{quant}}$$

Par exemple, étant donné FP32 [1.2, -3.1, 0.8, 2.4, 5.4]

Facteur d'échelle = 127 / 5.4 = 23.5 **(constante de quantification)**

Nouveau Int8L [28, -73, 19, 56, 127]

https://llmsystem.github.io/llmsystem2024spring/assets/files/Group2-Presentation-cf8028bc58193a5e6e6d7b05709ef1a9.pdf

Formation pratique et mise en œuvre

Planning

- Fine-tuning de Llama2 sur un dataset personnalisé de HuggingFace en utilisant PEFT.
  - Code Python utilisant HuggingFace PEFT.
  - Sur Google Colab.
  - Exercice : refaire avec Mistral.
- Fine-tuning de Roberta sur des données personnalisées pour l'analyse de sentiments.
  - Pas de code.
  - HuggingFace AutoTrain.
- Fine-tuning de Mistral pour écrire des fichiers Docker sur Llama Factory.
  - Pas de code.
  - Exercice : refaire pour Llama2.