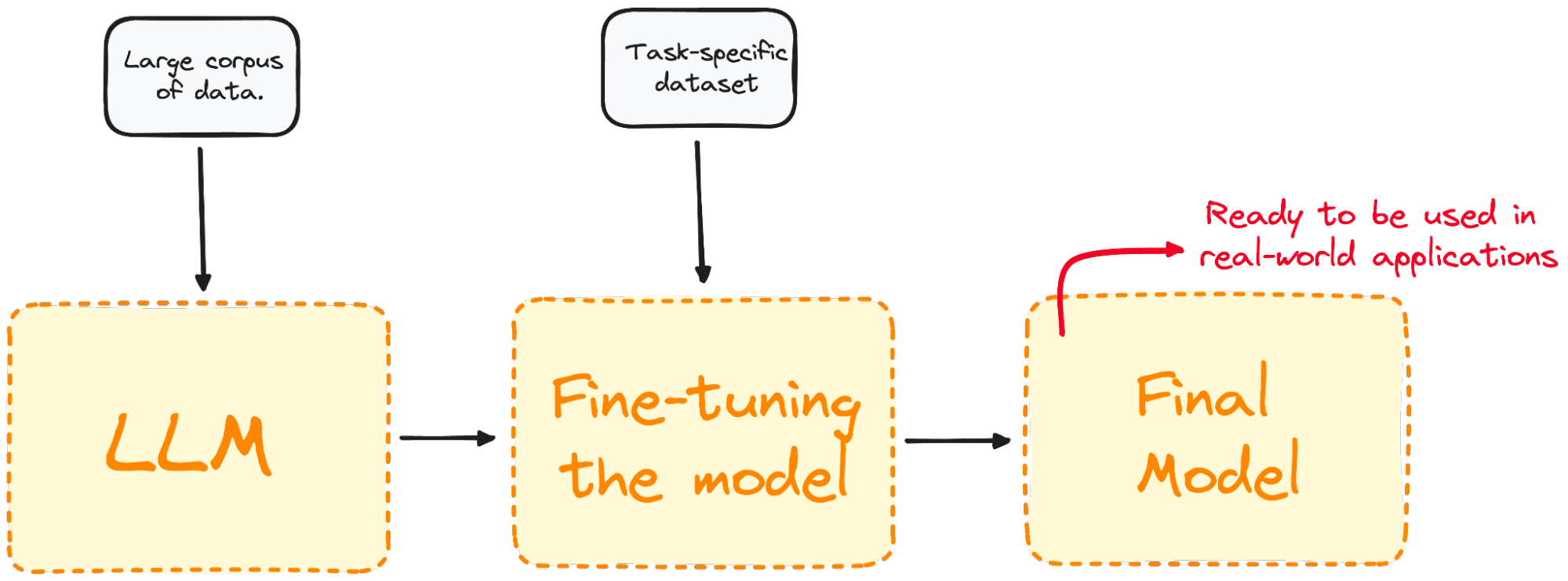
## **Fine-tuning Gemma 2 Using LoRA**

Fine-tuning is the process of taking a pre-trained model and further training it on a domain-specific dataset.

Most LLM models today have a very good global performance but fail in specific task-oriented problems. The fine-tuning process offers considerable advantages, including lowered computation expenses and the ability to leverage cutting-edge models without the necessity of building one from the ground up.

Transformers grant access to an extensive collection of pre-trained models suited for various tasks. Fine-tuning these models is a crucial step for improving the model's ability to perform specific tasks, such as sentiment analysis, question answering, or document summarization, with higher accuracy.



*Image by Author. Visualizing the Fine-Tuning process.*

Fine-tuning tailors the model to have a better performance for specific tasks, making it more effective and versatile in real-world applications. This process is essential for tailoring an existing model to a particular task or domain.

Whether to engage in fine-tuning hinges on your goals, which typically vary based on the specific domain or task at hand.

## **The Different Types of Fine-tuning**

Fine-tuning can be approached in several ways, depending mainly on its main focus and specific goals.

### **Supervised fine-tuning**

The most straightforward and common fine-tuning approach. The model is further trained on a labeled dataset specific to the target task to perform, such as text classification or named entity recognition.

For instance, we would train our model on a dataset containing text samples labeled with their corresponding sentiment for sentiment analysis.

### **Few-shot learning**

There are some cases where collecting a large labeled dataset is impractical. Few-shot learning tries to address this by providing a few examples (or shots) of the required task at the beginning of the input prompts. This helps the model have a better context of the task without an extensive fine-tuning process.

### **Transfer learning**

Even though all fine-tuning techniques are a form of transfer learning, this category is specifically aimed to allow a model to perform a task different from the task it was initially trained on. The main idea is to leverage the knowledge the model has gained from a large, general dataset and apply it to a more specific or related task.

### **Domain-specific fine-tuning**

This type of fine-tuning tries to adapt the model to understand and generate text that is specific to a particular domain or industry. The model is fine-tuned on a dataset composed of text from the target domain to improve its context and knowledge of domain-specific tasks.

For instance, to generate a chatbot for a medical app, the model would be trained with medical records, to adapt its language understanding capabilities to the health field.

**Fine-tuning truyền thống (Transformers)**

**Ưu điểm:**

1. **Toàn diện:**

- Bạn fine-tune toàn bộ mô hình, do đó có thể tối ưu hóa cho dữ liệu và tác vụ cụ thể.

- Thích hợp khi bạn cần thay đổi toàn bộ trọng số của mô hình.

1. **Hiệu quả trên tập dữ liệu lớn:**

- Nếu có dữ liệu phong phú và phần cứng mạnh, fine-tuning truyền thống có thể tạo ra mô hình mạnh mẽ nhất.

1. **Kiểm soát hoàn toàn:**

- Cho phép bạn thay đổi toàn bộ kiến trúc và các thành phần của mô hình nếu cần.

**Nhược điểm:**

1. **Tốn tài nguyên:**

- Yêu cầu GPU/TPU mạnh với bộ nhớ lớn (như A100 với 40GB VRAM).

- Quá trình fine-tune rất chậm nếu mô hình lớn (như Gemma 2).

1. **Khó lưu trữ và chia sẻ:**

- Mô hình sau fine-tuning thường lớn, khó lưu trữ và tải lên.

Khi nào nên chọn Transformers Fine-tuning?

- Khi bạn có đủ tài nguyên phần cứng.

- Khi tập dữ liệu lớn và chất lượng cao.

- Khi bạn cần thay đổi toàn bộ cấu trúc hoặc điều chỉnh sâu mô hình.

**LoRA (Low-Rank Adaptation)**

LoRA (Low-Rank Adaptation) là một phương pháp tối ưu hóa tham số hiệu quả (Parameter-Efficient Fine-Tuning) được sử dụng trong lĩnh vực học sâu, đặc biệt với các mô hình ngôn ngữ lớn (LLMs) như GPT, BERT, hay các mô hình lớn khác.

LoRA được thiết kế để giúp fine-tune các mô hình lớn với tài nguyên hạn chế, bằng cách giảm số lượng tham số cần cập nhật trong quá trình huấn luyện.

**Ưu điểm:**

1. **Nhẹ và nhanh:**

- Chỉ cập nhật một số lượng nhỏ tham số trong mô hình, giảm tải tài nguyên đáng kể.

- Dễ thực hiện trên máy có GPU trung bình (như RTX 3090 với 24GB VRAM).

1. **Hiệu quả với dữ liệu nhỏ:**

- Tốt hơn cho fine-tuning trên các tập dữ liệu nhỏ hoặc khi không muốn thay đổi toàn bộ mô hình gốc.

1. **Tính linh hoạt:**

- Giữ nguyên mô hình gốc, chỉ lưu các tham số đã tinh chỉnh (thường rất nhỏ, khoảng vài MB).

1. **Dễ chia sẻ:**

- Thay vì lưu cả mô hình, bạn chỉ cần lưu phần adapter đã tinh chỉnh, dễ dàng sử dụng lại hoặc chia sẻ.

**Nhược điểm:**

1. **Hạn chế thay đổi:**

- Không thể điều chỉnh toàn bộ mô hình. Nếu tác vụ cần thay đổi lớn, LoRA không phải là lựa chọn tốt.

1. **Hiệu suất giới hạn:**

- Hiệu suất có thể không tốt bằng fine-tuning truyền thống trong các tác vụ phức tạp hoặc yêu cầu nhiều thay đổi trong mô hình.

Khi nào nên chọn LoRA?

- Khi bạn có tài nguyên phần cứng hạn chế.

- Khi tập dữ liệu nhỏ và tác vụ chỉ yêu cầu tinh chỉnh nhẹ mô hình.

- Khi bạn cần triển khai hoặc chia sẻ mô hình nhanh chóng.

Accessing Gemma 2 and Fine-tuning Gemma 2 using LoRA: <https://www.datacamp.com/tutorial/fine-tuning-gemma-2>

**Sử dụng mô hình trong Ollama**

**Chuyển đổi mô hình sang GGML:**

1. **Cài đặt llama.cpp:**

git clone https://github.com/ggerganov/llama.cpp

cd llama.cpp

make

1. **Chuyển đổi mô hình:**

python3 ./convert\_hf\_to\_gguf.py ../Gemma-2-9b-it-chat-doctor --outfile gemma-2-2b.gguf --outtype f32

f32: Floating-point 32-bit (chính xác nhất, nhưng sử dụng nhiều tài nguyên nhất).

f16: Floating-point 16-bit (nhẹ hơn, hiệu suất tốt).

bf16: Brain Floating-point 16-bit (phiên bản hiệu quả của f16).

q8\_0: Quantized 8-bit (nhẹ hơn, phù hợp cho inference).

tq1\_0 và tq2\_0: Các tùy chọn quantized khác (nếu được hỗ trợ bởi mô hình).

**Thêm mô hình vào Ollama:**

<https://github.com/ollama/ollama/blob/main/docs/import.md>

<https://www.gpu-mart.com/blog/import-models-from-huggingface-to-ollama>

1. **Create a Modelfile for Ollama**: A Modelfile specifies the model's configuration for Ollama.

Example Modelfile:

# Modelfile

FROM "./Starling-LM-7B-beta-Q6\_K.gguf"

Replace ./Starling-LM-7B-beta-Q6\_K.gguf with the path to the GGUF file you downloaded. The TEMPLATE line defines the prompt format using system, user, and assistant roles. You can customize this based on your use case.

1. **Build the Model in Ollama**: Now, build the Ollama model using the ollama create command:

ollama create "Starling-LM-7B-beta-Q6\_K" -f Modelfile

Replace Starling-LM-7B-beta-Q6\_K with the name you want to give your model, and Modelfile with the path to your Modelfile.

1. **Run and Test the Model:** Finally, you can run and try your model using the ollama run command:

ollama run Starling-LM-7B-beta-Q6\_K:latest

The :latest tag runs the most recent version of your model. That's it! You have successfully imported a Hugging Face model and created a custom Ollama model.

API: <http://localhost:11434/api/generate>

Body:

{

"model": "gemma-2-2b",

"prompt": "Hello, how are you?",

"stream": false

}