

Open in app ↗

Sign up

Sign In



Search Medium



# 用GGML和美洲駝量化美洲駝模型.cpp

GGML vs. GPTQ vs. NF4



馬克西姆·拉博納 · 跟隨

發表於 邁向數據科學

9 分鐘閱讀 · 4月&lt;&gt;



Listen



Share



圖片來源：作者

由於大型語言模型 ( LLM ) 的規模龐大，量化已成為有效運行它們的基本技術。通過降低其權重的精度，您可以節省記憶體並加快推理速度，同時保留模型的大部分性能。最近，8位和4位量化解鎖了**在消費類硬體上運行LLM**的可能性。再加上駱駝模型和參數高效技術的發佈來微調它們 ( LoRA , QLoRA ) ，這創造了一個豐富的本地LLM生態系統，現在正在與OpenAI的GPT-3.5和GPT-4競爭。

除了本文介紹的樸素方法外，還有三種主要的量化技術：NF4、GPTQ 和 GGML。NF4 是 QLoRA 使用的一種靜態方法，用於以 4 位精度載入模型以執行微調。在上一篇文章中，我們探討了 GPTQ 方法，並量化了我們自己的模型以在消費者 GPU 上運行它。在本文中，我們將介紹 GGML 技術，瞭解如何量化駱駝模型，並提供實現最佳結果的提示和技巧。

您可以在[Google Colab](#)和[GitHub](#)上找到代碼。

## 什麼是 GGML？

GGML是一個專注於機器學習的C庫。它是由Georgi Gerganov創建的，這就是首字母“GG”所代表的。該庫不僅提供機器學習的基礎元素，例如張量，而且還提供分發LLM的獨特二進位格式。

這種格式最近改為GGUF。這種新格式設計為可擴展，因此新功能不應破壞與現有模型的相容性。它還將所有元數據集中在一個檔中，例如特殊令牌、RoPE 縮放參數等。簡而言之，它回答了一些歷史痛點，應該是面向未來的。有關更多資訊，您可以閱讀[此地址](#)的規範。在本文的其餘部分，我們將“GGML 模型”稱為所有使用 GGUF 或以前格式的模型。

GGML被設計為與同樣由Georgi Gerganov創建的[llama.cpp](#)庫一起使用。該庫是用C/C++ 編寫的，用於高效推理 Llama 模型。它可以載入GGML模型並在CPU上運行它們。最初，這是與GPTQ模型的主要區別，GPTQ模型在GPU上載入和運行。但是，您現在可以使用llama.cpp將LLM的某些層卸載到GPU。舉個例子，35b 參數模型有 7 個層。這大大加快了推理速度，並允許您運行不適合 VRAM 的 LLM。

```
F:/Desktop/llama.cpp-master $ ./main.exe -m ./models/ggml-model-q5_k_m.gguf -n 128 -ngl 125 -p "Write a Python function to print the nth Fibonacci numbers"
```

圖片來源：作者

如果你喜歡命令行工具，llama.cpp和GGUF支持已經集成到許多GUI中，比如oobabooga的文本生成web-ui，koboldcpp，LM Studio或ctransformers。您可以簡單地使用這些工具載入您的 GGML 模型，並以類似 ChatGPT 的方式與它們進行交互。幸運的是，許多量化模型可以直接在擁抱面部中心獲得。您很快就會注意到，其中大多數都是由LLM社區中的流行人物TheBloke量化的。

在下一節中，我們將瞭解如何量化我們自己的模型並在消費者 GPU 上運行它們。

## 如何使用GGML量化LLM？

讓我們看看 [TheBloke/Llama-2-13B-chat-GGML](#) 儲存庫中的檔。我們可以看到 14 種不同的 GGML 模型，對應於不同類型的量化。它們遵循特定的命名約定：「q」+用於儲存權重（精度）的位數+特定變體。以下是所有可能的定量方法及其相應用例的清單，基於TheBloke製作的模型卡：

- q2\_k：將 Q4\_K 用於 attention.vw 和 feed\_forward.w2 張量，Q2\_K用於其他張量。
- q3\_k\_l：將 Q5\_K 用於 attention.wv、attention.wo 和 feed\_forward.w2 張量，否則 Q3\_K
- q3\_k\_m：將 Q4\_K 用於 attention.wv、attention.wo 和 feed\_forward.w2 張量，否則 Q3\_K

- q3\_k\_s : 對所有張量使用Q3\_K
- q4\_0 : 原始定量方法，4 位。
- q4\_1 : 精度高於q4\_0，但不如q5\_0高。但是，推理速度比 q5 模型更快。
- q4\_k\_m : 使用 Q6\_K 表示一半的 attention.wv 和 feed\_forward.w2 張量，否則Q4\_K
- q4\_k\_s : 對所有張量使用Q4\_K
- q5\_0 : 更高的準確性、更高的資源使用率和更慢的推理速度。
- q5\_1 : 更高的準確性、資源使用率和更慢的推理速度。
- q5\_k\_m : 使用 Q6\_K 表示一半的 attention.wv 和 feed\_forward.w2 張量，否則Q5\_K
- q5\_k\_s : 對所有張量使用Q5\_K
- q6\_k : 對所有張量使用Q8\_K
- q8\_0 : 與浮點數16幾乎無法區分。資源使用率高，速度慢。不建議大多數使用者使用。

根據經驗，**我建議使用 Q5\_K\_M**，因為它可以保留模型的大部分性能。或者，如果要節省一些記憶體，可以使用Q4\_K\_M。通常，K\_M版本比K\_S版本更好。我不能推薦Q2或Q3版本，因為它們會大大降低模型性能。

現在我們對可用的量化類型有了更多的瞭解，讓我們看看如何在真實模型上使用它們。您可以在[Google Colab](#)上的**免費T4 GPU**上執行以下代碼。第一步包括編譯 llama.cpp並在我們的 Python 環境中安裝所需的庫。

```
# Install llama.cpp
!git clone https://github.com/ggerganov/llama.cpp
!cd llama.cpp && git pull && make clean && LLAMA_CUBLAS=1 make
!pip install -r llama.cpp/requirements.txt
```

現在我們可以下載我們的模型了。我們將使用我們在上一篇文章中微調的模型。

[mlabonne/EvolCodeLlama-7b](#)

```
MODEL_ID = "mlabonne/EvolCodeLlama-7b"

# Download model
!git lfs install
!git clone https://huggingface.co/{MODEL_ID}
```

此步驟可能需要一段時間。完成後，我們需要將權重轉換為 GGML FP16 格式。

```
MODEL_NAME = MODEL_ID.split('/')[1]
GGML_VERSION = "gguf"

# Convert to fp16
fp16 = f"{MODEL_NAME}/{MODEL_NAME.lower()}.{GGML_VERSION}.fp16.bin"
!python llama.cpp/convert.py {MODEL_NAME} --outtype f16 --outfile {fp16}
```

最後，我們可以使用一種或多種方法量化模型。在這種情況下，我們將使用我之前推薦的 Q4\_K\_M 和 Q5\_K\_M 方法。這是實際需要 GPU 的唯一步驟。

```
QUANTIZATION_METHODS = ["q4_k_m", "q5_k_m"]

for method in QUANTIZATION_METHODS:
    qtype = f"{MODEL_NAME}/{MODEL_NAME.lower()}.{GGML_VERSION}.{method}.bin"
    !./llama.cpp/quantize {fp16} {qtype} {method}
```

我們的兩個量化模型現在已經**準備好進行推理**了。我們可以檢查 bin 檔的大小，看看我們壓縮了多少。FP16 型號佔用 13.5 GB，而 Q4\_K\_M 型號佔用 4.08 GB（小 3.3 倍），Q5\_K\_M 型號佔用 4.78 GB（小 2.8 倍）。

讓我們使用駱駝.cpp 來有效地運行它們。由於我們使用的是具有 16 GB VRAM 的 GPU，因此我們可以將每一層卸載到 GPU。在本例中，它表示 35 層（7b 參數模型），因此我們將使用該參數。在下面的代碼塊中，我們還將輸入提示和要使用的量化方法。-ngl

```
import os

model_list = [file for file in os.listdir(MODEL_NAME) if GGML_VERSION in file]
prompt = input("Enter your prompt: ")
chosen_method = input("Please specify the quantization method to run the model (or")

# Verify the chosen method is in the list
if chosen_method not in model_list:
    print("Invalid method chosen!")
else:
    qtype = f"{MODEL_NAME}/{MODEL_NAME.lower()}.{GGML_VERSION}.{method}.bin"
    !./llama.cpp/main -m {qtype} -n 128 --color -ngl 35 -p "{prompt}"
```

讓我們使用 Q5\_K\_M 方法詢問模型「編寫一個 Python 函數來列印第 n 個斐波那契數」。如果我們查看日誌，我們可以確認我們成功地卸載了我們的層，這要歸功於“llm\_load\_tensors：將 35/35 層卸載到 GPU”這一行。下面是模型產生的代碼：

```
def fib(n):
    if n == 0 or n == 1:
        return n
    return fib(n - 2) + fib(n - 1)

for i in range(1, 10):
    print(fib(i))
```

這不是一個非常複雜的提示，但它很快就成功地生成了一段工作代碼。使用此 GGML，您可以使用互動模式（標誌）將本地 LLM 用作終端中的助手。請注意，這也適用於帶有 Apple 的 Metal Performance Shaders（MPS）的 Macbook，這是運行 LLM 的絕佳選擇。-i

最後，我們可以將量化模型推送到擁抱面部集線器上帶有“-GGUF”後綴的新存儲庫。首先，讓我們登錄並修改以下代碼塊以匹配您的使用者名。

```
!pip install -q huggingface_hub
```

```
username = "mlabonne"

from huggingface_hub import notebook_login, create_repo, HfApi
notebook_login()
```

現在，我們可以創建存儲庫並上傳模型。我們使用參數來過濾要上傳的檔，因此我們不會推送整個目錄。allow\_patterns

```
api = HfApi()

# Create repo
create_repo(
    repo_id=f"{username}/{MODEL_NAME}-GGML",
    repo_type="model",
    exist_ok=True
)

# Upload bin models
api.upload_folder(
    folder_path=MODEL_NAME,
    repo_id=f"{username}/{MODEL_NAME}-GGML",
    allow_patterns=f"*{GGML_VERSION}*",
)
```

我們已經成功地量化、運行了 GGML 模型，並將其推送到擁抱面部中心！在下一節中，我們將探討GGML如何實際量化這些模型。

## 使用 GGML 進行定量

GGML量化權重的方式並不像GPTQ那樣複雜。基本上，它對值塊進行分組並將它們捨入到較低的精度。某些技術（如Q4\_K\_M和Q5\_K\_M）為關鍵層實現了更高的精度。在這種情況下，每個權重都以 4 位精度存儲，除了一半的 attention.wv 和 feed\_forward.w2 張量。實驗證明，這種混合精度是準確性和資源使用之間的良好權衡。

如果我們查看 ggml.c 文件，我們可以看到塊是如何定義的。例如，結構定義為：

```
block_q4_0
```

```
#define QK4_0 32
typedef struct {
    ggml_fp16_t d;           // delta
    uint8_t qs[QK4_0 / 2];  // nibbles / quants
} block_q4_0;
```

在 GGML 中，權重以塊的形式處理，每個塊由 32 個值組成。對於每個塊，比例因數（delta）是從最大權重值派生的。然後，對塊中的所有重量進行縮放、量化和高效包裝以進行存儲（半位元節）。這種方法大大降低了存儲要求，同時允許在原始權重和量化權重之間進行相對簡單和確定的轉換。

現在我們對量化過程有了更多的瞭解，我們可以將結果與 NF4 和 GPTQ 進行比較。

## NF4 vs. GGML vs. GPTQ

哪種技術更適合 4 位量化？為了回答這個問題，我們需要介紹運行這些量化LLM的不同後端。對於GGML模型，美洲駝.cpp Q4\_K\_M模型是要走的路。對於 GPTQ 模型，我們有兩個選擇：[AutoGPTQ](#) 或 [ExLlama](#)。最後，NF4型號可以直接在帶有標誌的變壓器中運行。 --load-in-4bit

Oobabooga 在一篇出色的[博客文章中](#)進行了多個實驗，比較了不同模型的困惑度（越低越好）：





## Llama-13b Perplexity



## Llama-30b Perplexity

NF4	5.73047	5.24609
GPTQ (AutoGPTQ)	5.72656	5.30078
GPTQ (ExLlama)	5.72581	5.25923
GGML (Q4_K_M)	5.71705	5.21557

基於這些結果，我們可以說GGML模型在困惑度方面略有優勢。差異不是特別顯著，這就是為什麼最好以代幣/秒為單位關注生成速度的原因。最好的技術取決於您的GPU：如果您有足夠的VRAM來擬合整個量化模型，那麼帶有ExLlama的GPTQ將是最快的。如果不是這種情況，您可以卸載一些層並將 **GGML 模型與 llama 一起使用.cpp**來運行您的LLM。

### 結論

在本文中，我們介紹了GGML庫和新的GGUF格式，以有效地存儲這些量化模型。我們用它來**量化我們自己的不同格式**（Q4\_K\_M和Q5\_K\_M）的駱駝模型。然後，我們運行GGML模型並將我們的bin檔推送到Hugging Face Hub。最後，我們深入研究了GGML的代碼，以瞭解它如何實際量化權重，並將其與NF4和GPTQ進行比較。

量化是通過降低運行LLM的成本來實現LLM民主化的強大載體。未來，混合精度和其他技術將繼續提高我們用量化權重實現的性能。在那之前，我希望你喜歡閱讀這篇文章並學到一些新東西。

如果您對有關LLM的更多技術內容感興趣，請在[Medium](#)上關注我。

### 有關量化的文章

## 第 1 部分：重量量化簡介

使用8位量化減小大型語言模型的大小

[towardsdatascience.com](https://towardsdatascience.com)

## 第 2 部分：使用 GPTQ 進行 4 位量化

使用 AutoGPTQ 量化您自己的 LLM

[towardsdatascience.com](https://towardsdatascience.com)

瞭解有關機器學習的更多資訊，只需按兩下即可支援我的工作 - 在此處成為 Medium 會員：

## 使用我的推薦連結加入 Medium — 馬克西姆·拉博納

作為 Medium 會員，您的部分會員費將用於您閱讀的作家，您可以完全訪問每個故事.....

[medium.com](https://medium.com)

程式設計

大型語言模型

數據科學

量化

機器學習



Follow

**作者：Maxime Labonne**

2.8K 粉絲 · 作家邁向數據科學

博士，高級機器學習科學家@摩根大通•“動手圖神經網路”的作者•[twitter.com/maximelabonne](https://twitter.com/maximelabonne)更多來自 **Maxime Labonne** 和 **Towards Data Science**

Maxime Labonne in Towards Data Science

## Fine-Tune Your Own Llama 2 Model in a Colab Notebook

A practical introduction to LLM fine-tuning

★ · 12 min read · Jul 26

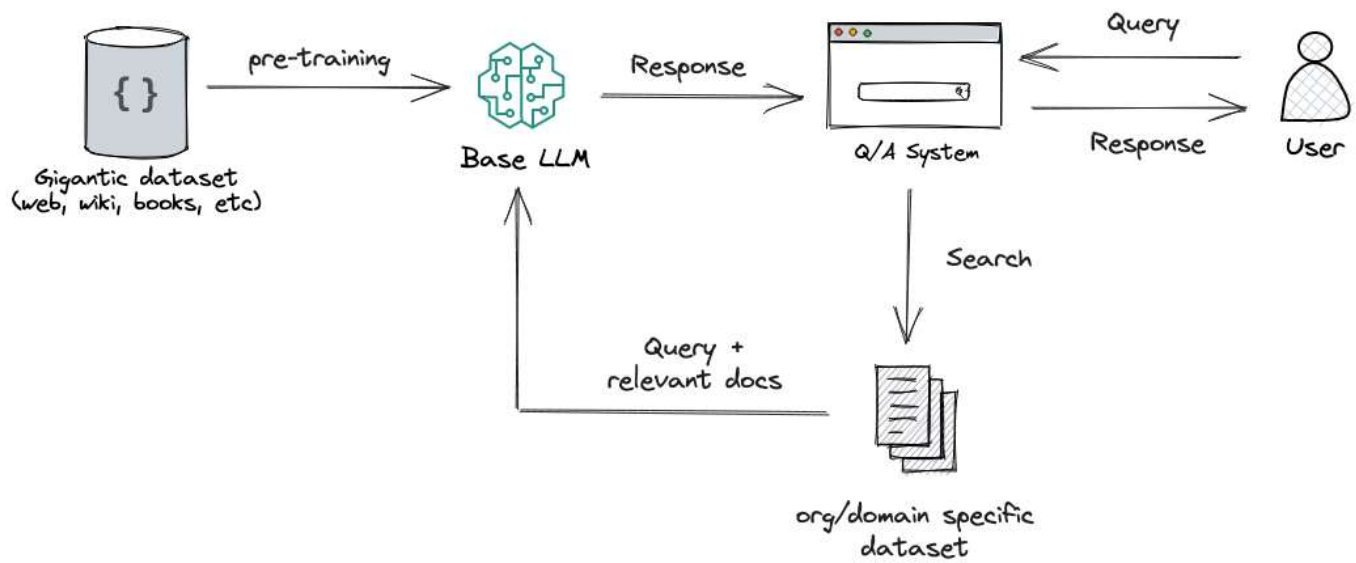


2.2K



37





Heiko Hotz in Towards Data Science

## RAG vs Finetuning—Which Is the Best Tool to Boost Your LLM Application?

The definitive guide for choosing the right method for your use case

★ • 19 min read • Aug 25



1.8K



16







Cameron R. Wolfe, Ph.D. in Towards Data Science

## Advanced Prompt Engineering

What to do when few-shot learning isn't enough...

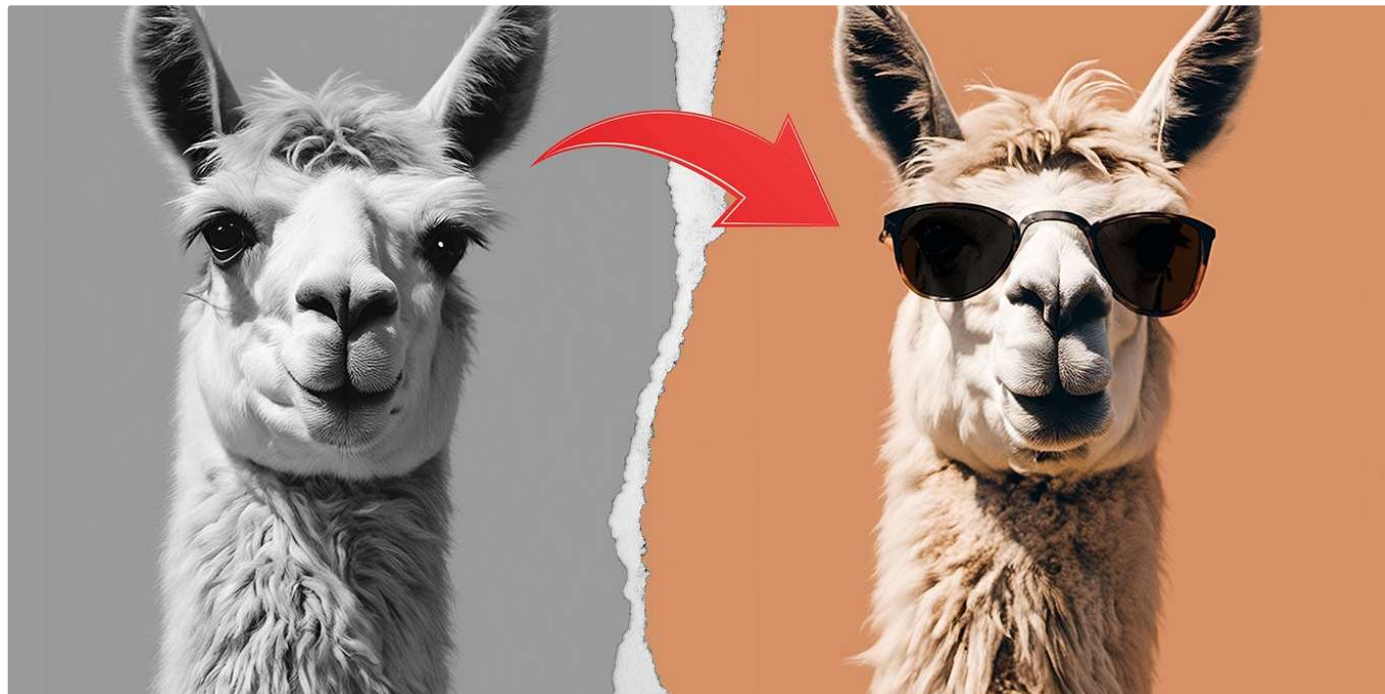
★ • 17 min read • Aug 7



1.1K



8



Maxime Labonne in Towards Data Science

## A Beginner's Guide to LLM Fine-Tuning

How to fine-tune Llama and other LLMs with one tool

★ • 8 min read • Aug 30



400



5

[See all from Maxime Labonne](#)[See all from Towards Data Science](#)

Best access to the next version of Llama

ama Chat

Michael Humor in GoPenAI

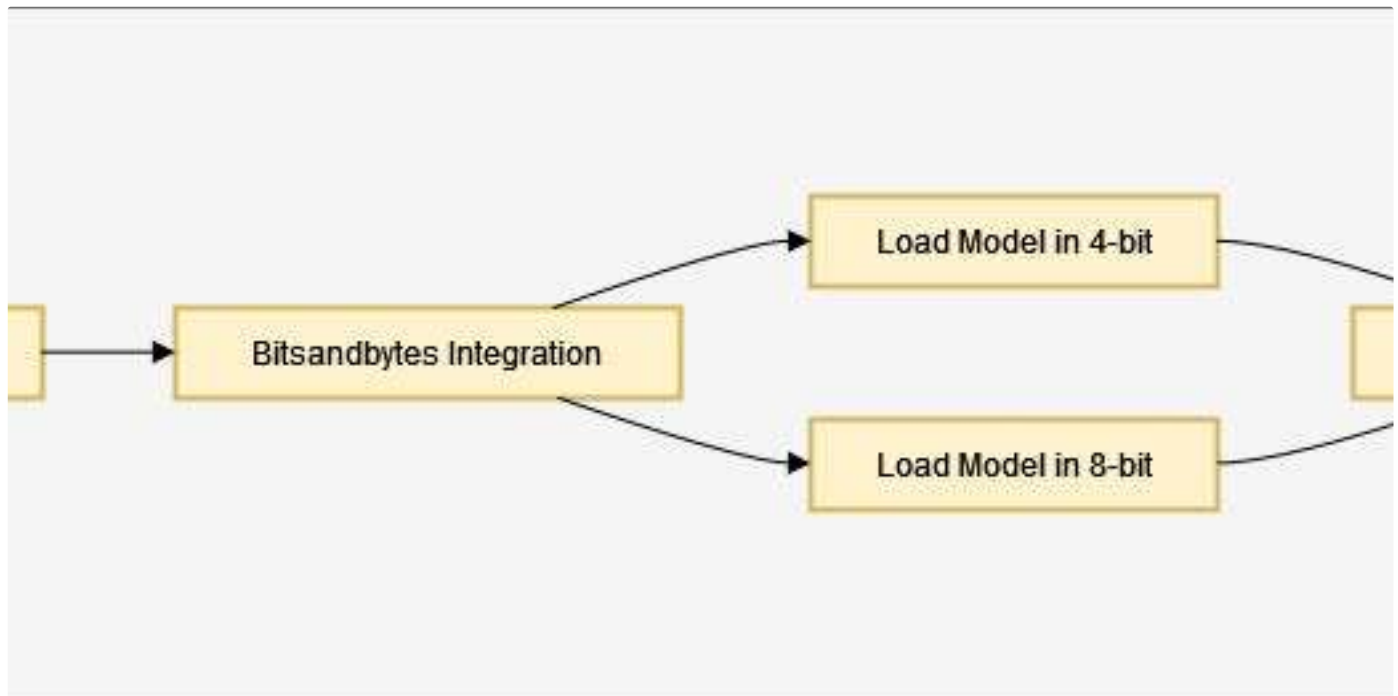
## Can you run Llama 2 on a standard laptop, such as a Mac Pro?

4 min read · Aug 28



7





Rakesh Rajpurohit

## Model Quantization with 🤗 Hugging Face Transformers and Bitsandbytes Integration

Introduction:

4 min read · Aug 21



6



### Lists



#### Predictive Modeling w/ Python

20 stories378 saves ·



#### Practical Guides to Machine Learning

10 stories422 saves ·



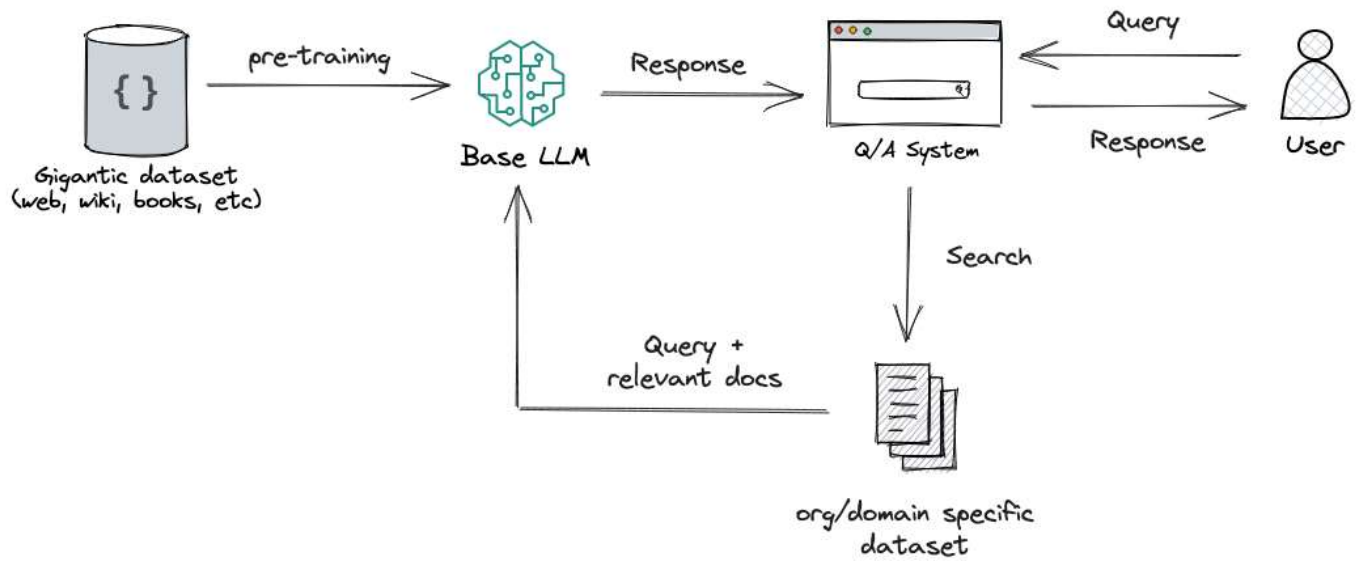
#### Natural Language Processing

599 stories215 saves ·



#### It's never too late or early to start something

15 stories116 saves ·



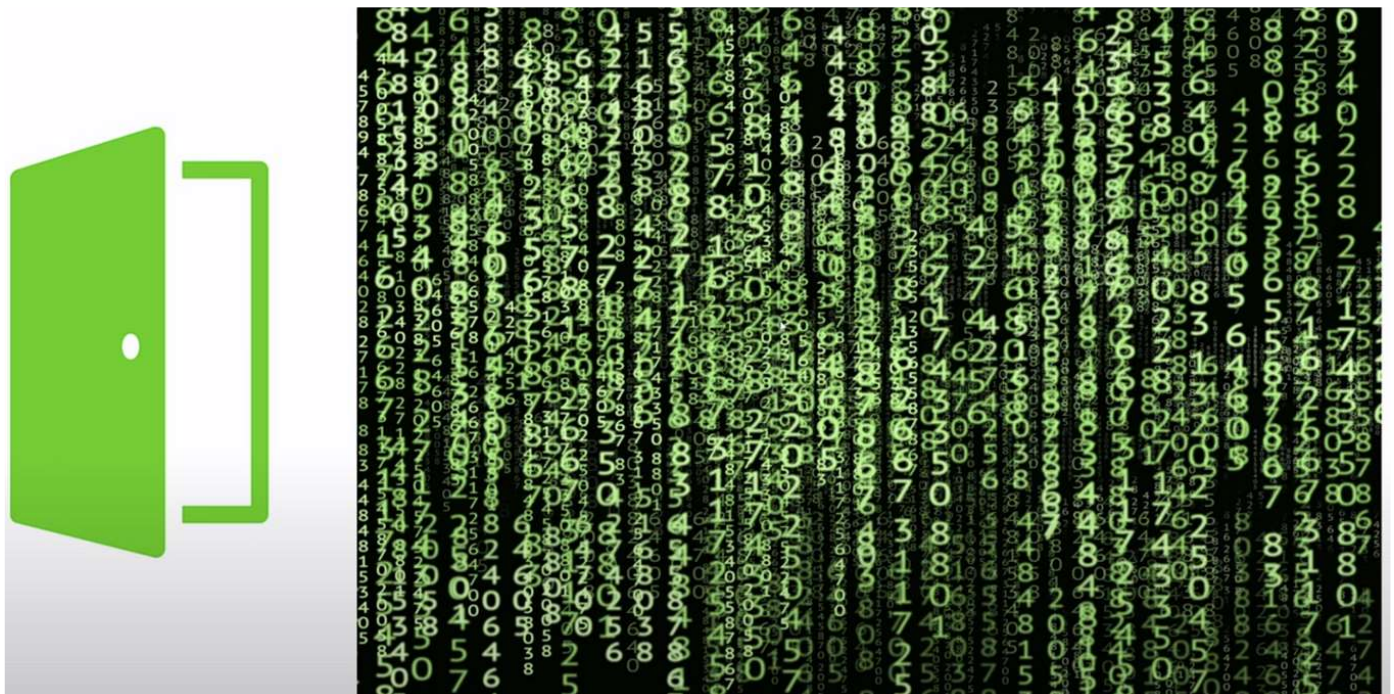
 Heiko Hotz in Towards Data Science

## RAG vs Finetuning—Which Is the Best Tool to Boost Your LLM Application?

The definitive guide for choosing the right method for your use case

★ • 19 min read • Aug 25

 1.8K  16







Murali Manohar

## Understanding LoRA and QLoRA — The Powerhouses of Efficient Finetuning in Large Language Models

Background

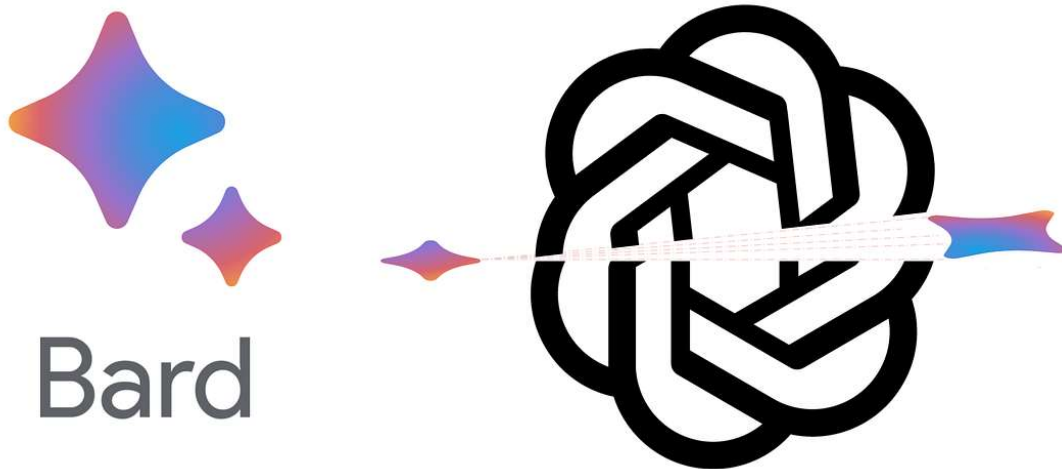
8 min read · Aug 8



37



1



AL Anany

## The ChatGPT Hype Is Over — Now Watch How Google Will Kill ChatGPT.

It never happens instantly. The business game is longer than you know.



· 6 min read · Sep 2



5.8K

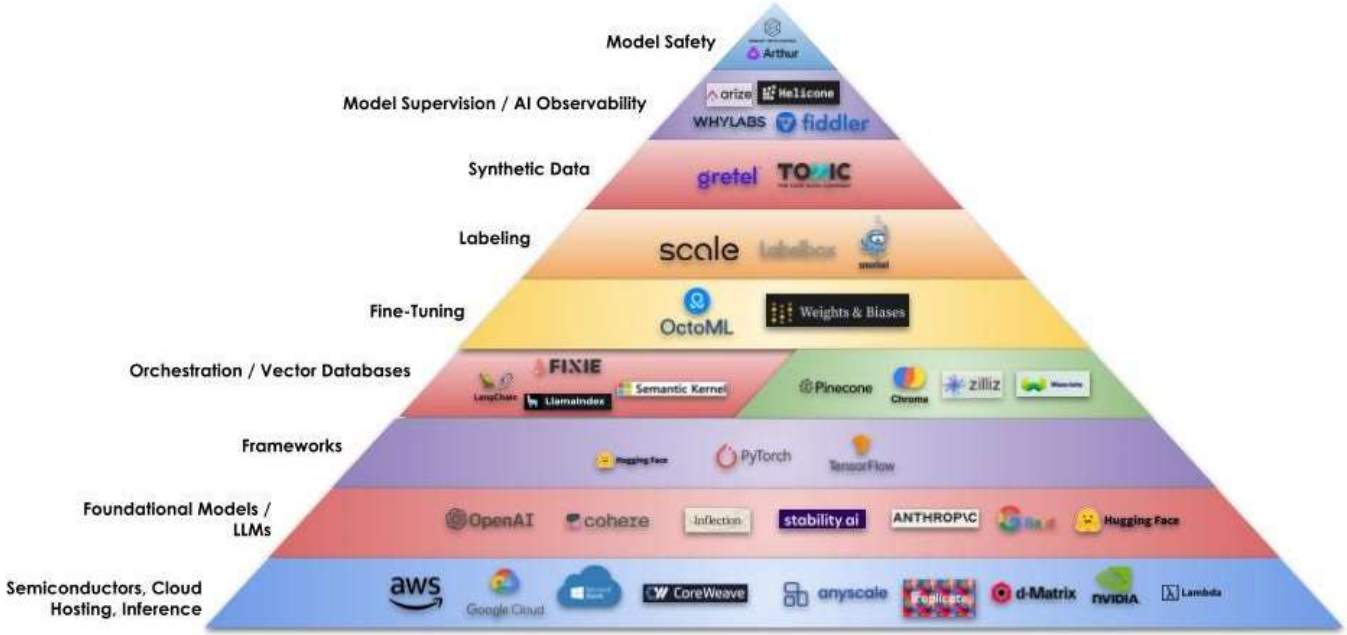



198



The Building Blocks of Generative AI

10 min read



 Jonathan Shriftman

# The Building Blocks of Generative AI

A Beginners Guide to The Generative AI Infrastructure Stack

22 min read · Jul 10

 490  5



See more recommendations