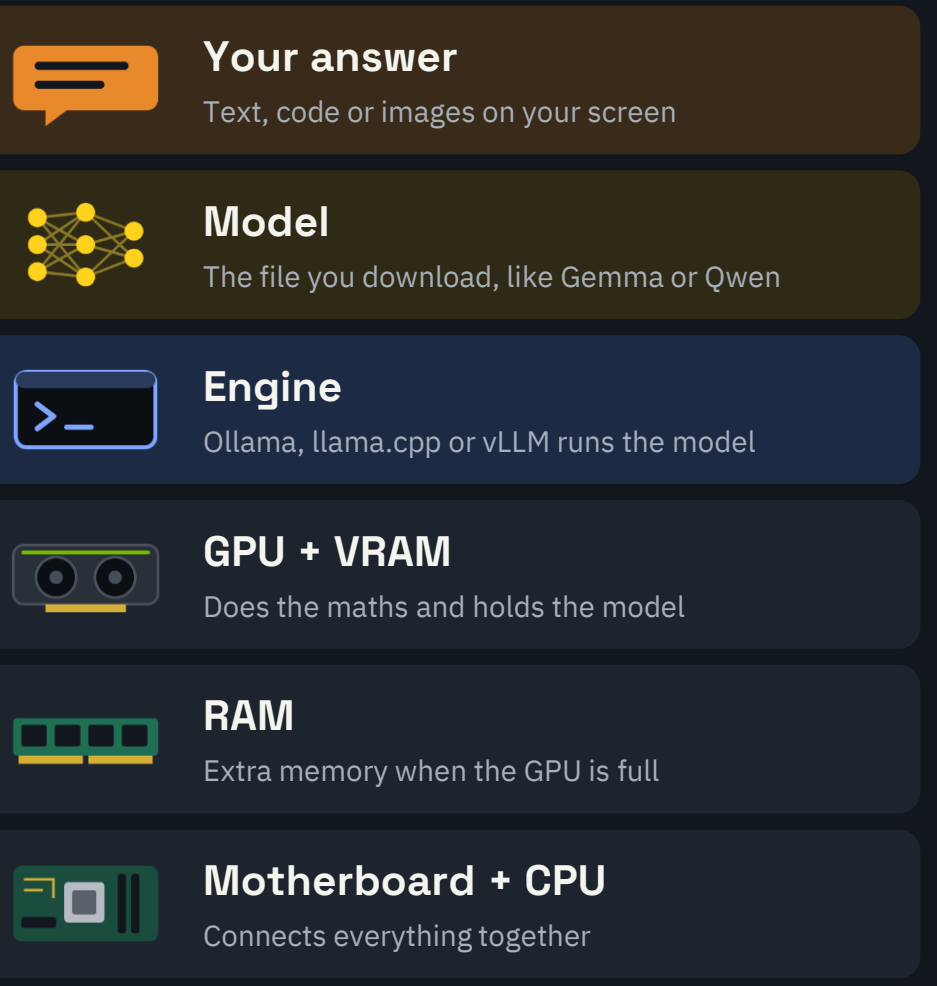
A BEGINNER'S GUIDE

# Running AI on your own computer

The hardware, the software, and how to pick the right model






Ibrahim Jaafar

TODAY'S GOAL

# What you will learn

- Tell if a model will **fit** on your computer, and how **fast** it will run
- Choose the right **engine** to run it: Ollama, llama.cpp or vLLM
- Understand what **compression** is, and spot it in a model's name (like **Q4_K_M**)
- Read a model's **name** on Hugging Face and pick the right **file**
- Run your **first model** on your own computer

WHY BOTHER?

# Why run AI on your own machine?

### Private

Your questions and files never leave your computer.

### Works offline

No internet connection needed once it’s set up.

### Free to use

No subscription and no fee per message.

### Full control

You choose the model, the settings and the software.

THE BIG PICTURE

# Five things to understand

1

### Hardware

What your computer needs

2

### Engines

The app that runs the model

3

### Compression

Making models smaller

4

### Model names

Decoding what you download

5

### Finding models

Where to get them safely

BEFORE WE START

# Key words you'll hear today

### Model / LLM

The AI itself: a program trained on huge amounts of text.

LLM = Large Language Model

### Token

A small piece of text the model reads and writes, about ¾ of a word.

"unbelievable" → un · believ · able

### Tokens / sec

How fast the model writes. Higher is faster.

Reading speed ≈ 5–10

### Weights

The numbers a model learned in training. Also called **parameters**.

8B = 8 billion weights

### Inference

Using a finished model to get answers (not training it).

### GPU & VRAM

The graphics card, and its own very fast memory.

### RAM

Your computer's normal working memory. Slower than VRAM.

### Context

How much of the chat the model can "see" at once, counted in tokens.

1

PART 1 OF 5

# Hardware

## What your computer needs to run AI

Memory size, memory speed, GPUs, Macs and other hardware.

1 · HARDWARE

# Two things decide everything

MEMORY SIZE

## Can the model fit?

A model has to be loaded into memory before it can answer. Too big = it won't run, or runs very slowly.

MEMORY SPEED

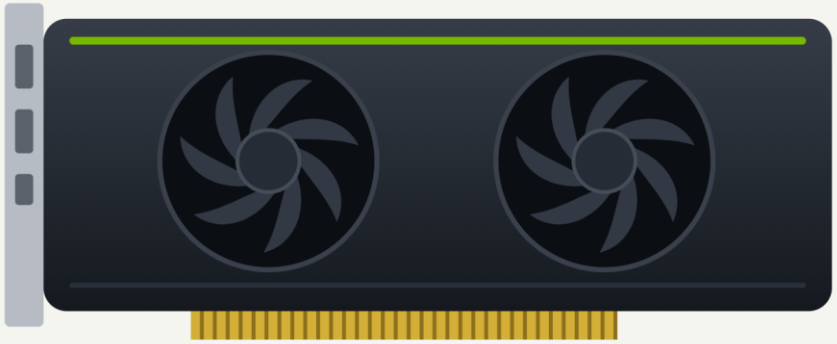
## How fast does it answer?

Speed is measured in **tokens per second**. A token is a small piece of a word.

*Think of a desk: its size is how many books fit on it; your reading speed is how fast you get through them.*

1 · HARDWARE

# VRAM: the graphics card’s own memory



e.g. NVIDIA RTX 3090 / 4090 / 5090

FITS IN VRAM

## Fast: 50–180+ tokens/sec

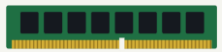
The whole model sits on the graphics card.

TOO BIG FOR VRAM

## 10–50× slower: 1–5 tokens/sec

Part of the model spills into normal system RAM, which is much slower. This is called **offloading to the CPU**.

# Faster memory = faster answers



**Normal PC memory (RAM)**

Memory speed: 60–100 GB/s

**2–8 tokens/sec**



**Apple Silicon Mac**

Memory speed: 150–800 GB/s

**15–60 tokens/sec**



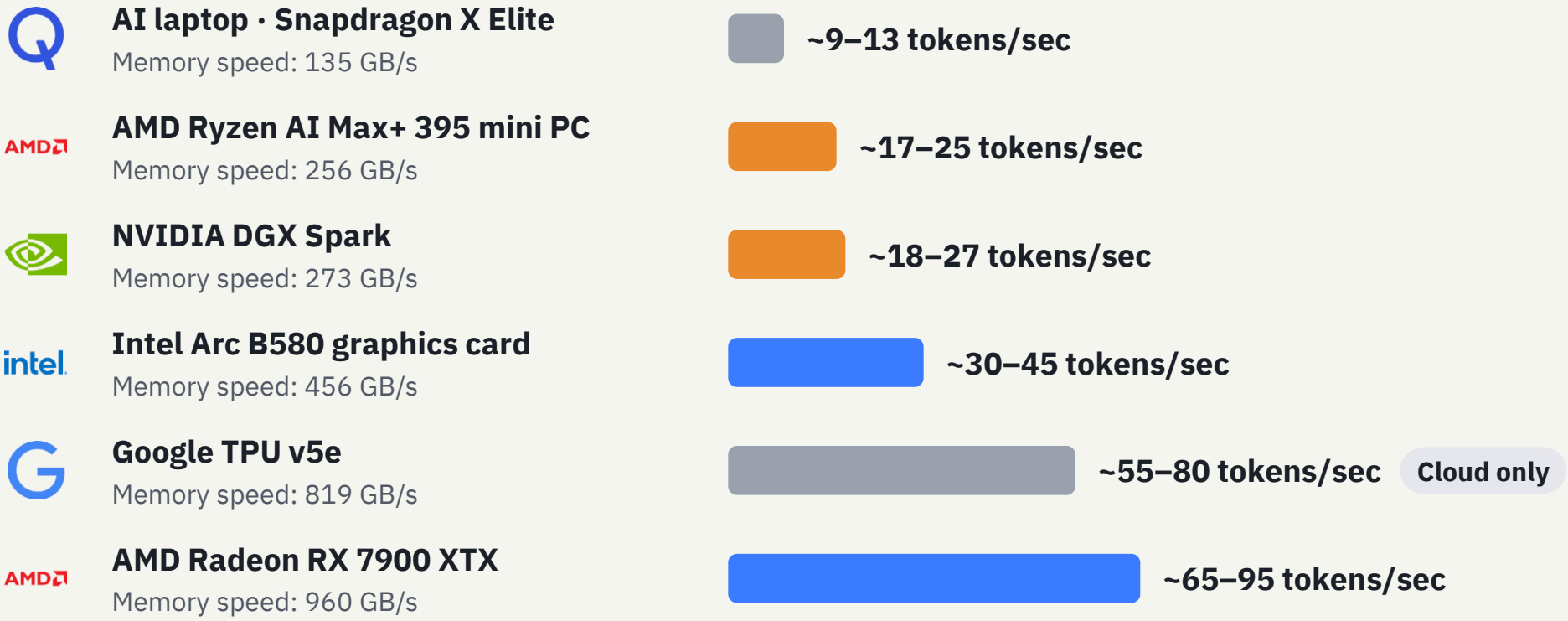
**Gaming graphics card**

Memory speed: 900–1,800 GB/s

**60–180+ tokens/sec**

Bar length = top speed. Reading speed for a person is roughly 5–10 tokens/sec.

# Other hardware you might hear about

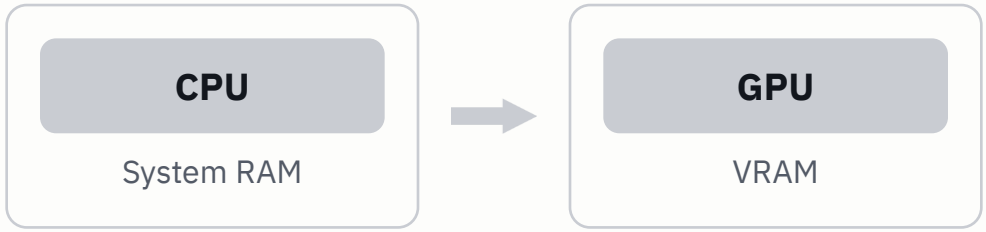



Rough estimates, same rule as the previous slide: about 1 token/sec for every 10–15 GB/s of memory speed. Bars use the same scale.

1 · HARDWARE

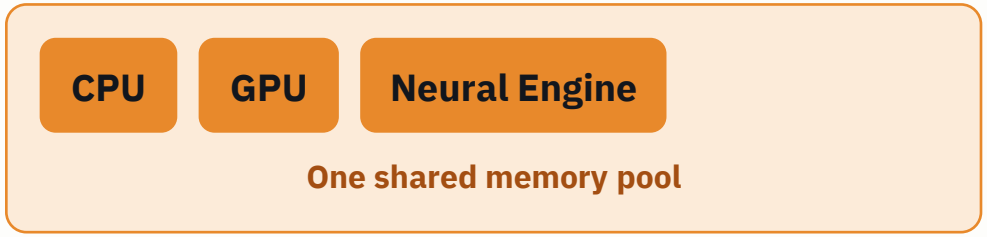
# Why Macs can run big models

### Typical PC




Two separate memory pools. Sharing between them is slow.

### Apple Silicon Mac (M1–M4)




Everything uses the same fast memory.

## A 64 GB memory Mac can give about 48 GB (~75%) to AI models.

Slower than a gaming GPU (slide 9), but it fits much bigger models.

# How much memory does a model need?

**Rule of thumb:** about 0.6 GB per billion parameters (“B”), plus a little extra for the conversation.

| Model size | File size | Minimum | Recommended |
|---|---|---|---|
| 1B – 3B | 1.5 – 3.5 GB | 8 GB RAM laptop (no GPU) | Any modern PC or Mac Mini |
| 7B – 8B | 4.5 – 5.2 GB | RTX 3060 (12 GB) / 16 GB Mac | RTX 4060 Ti 16 GB / 16 GB Mac |
| 14B – 32B | 9 – 20 GB | RTX 3090 (24 GB) / 32 GB Mac | RTX 4090 (24 GB) / 36 GB+ Mac |
| 70B | 40 – 43 GB | Two RTX 3090s (48 GB) | Mac Studio / Mac Pro, 64–128 GB |
| MoE (e.g. 8x7B) | 18 – 24 GB | 24 GB GPU / 36 GB Mac | 24 GB GPU / 64 GB Mac |

**Too big? Offload part to the CPU**

Some layers run from normal RAM instead of VRAM. It works, but gets 10–50× slower.

**Or use 2+ GPUs in one PC**

Their VRAM adds up: two RTX 3090s = 48 GB, enough for 70B. Ollama splits the model for you.

2

PART 2 OF 5

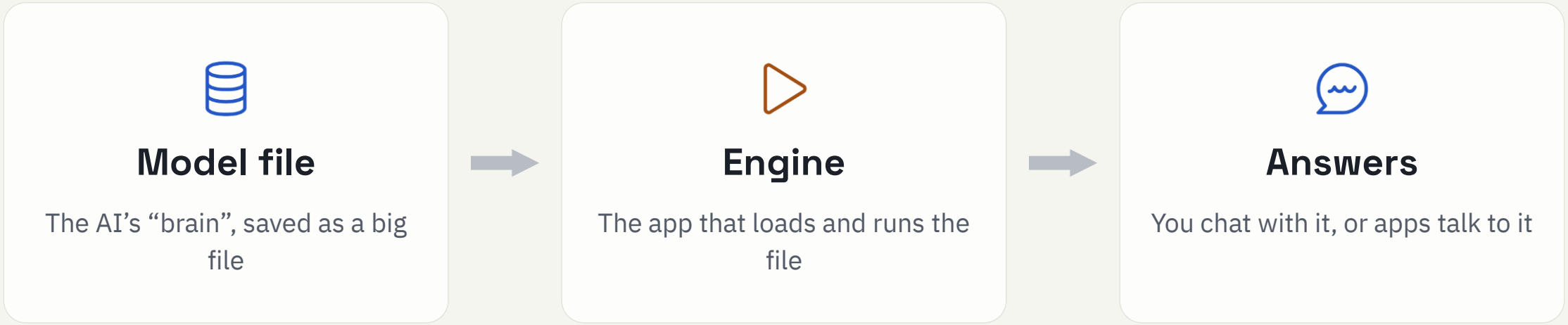
# Engines

## The software that runs the model

Ollama, llama.cpp and vLLM, and which one to pick.

2 · ENGINES

# What is an inference engine?




*Like a movie file and a video player: the file holds the content, the player makes it work.*

# Three popular engines

**How they relate:** Ollama runs llama.cpp inside it. vLLM is a separate engine, built for servers.



## Ollama

**The easy app**

**What it is:** A friendly app built on top of llama.cpp. One command and it works.

**Think of it as:** A car with automatic gears: just drive.

**Best for:** your own laptop, first steps

## llama.cpp

**The engine inside**

**What it is:** The core program that runs the model. Ollama uses it inside.

**Think of it as:** The car's engine: powerful, but you tune it yourself.

**Best for:** tinkerers, unusual hardware



## vLLM

**The server**

**What it is:** A separate engine for servers that answer many people at once.

**Think of it as:** A bus: built to carry many passengers at once.

**Best for:** a team or company, 10+ users

# Which engine should I use?

| | Ollama | llama.cpp | vLLM |
|---|---|---|---|
| How hard to set up? | Very easy | Easy to medium | Medium to hard |
| Runs on | Mac, Windows, Linux | Almost anything | Mainly NVIDIA / AMD GPUs |
| Model file type | GGUF | GGUF | Safetensors, AWQ, GPTQ |
| Several people asking at the same time? | Mostly one at a time, others wait in a queue | A few at once: memory split into fixed slots | Many (10+), no queue: memory shared on demand |

☆ **New to this?** Start with Ollama.

3

PART 3 OF 5

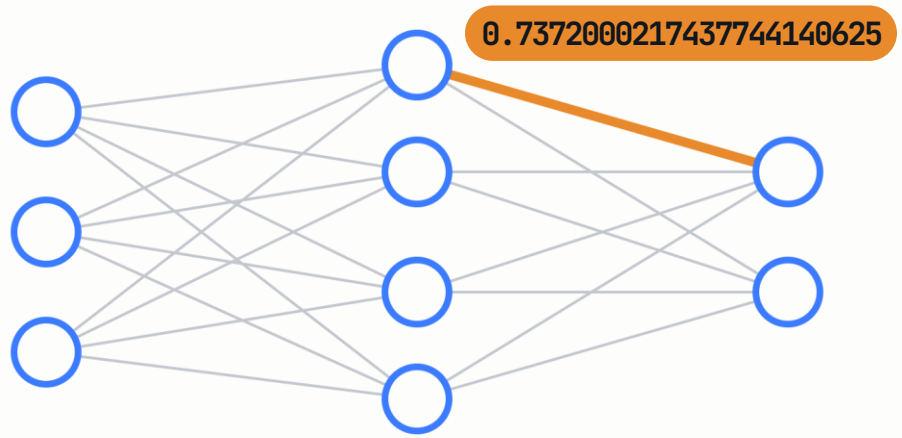
# Compression

## Making models smaller

Weights, bits, quantization levels and file types.

# What’s inside a model file

## A tiny piece of a model




Input Hidden layer Output

Each circle is a neuron. **Each line is one weight** which is a number learned in training. An 8B model has 8 billion of these lines (weights).

1. **A model = billions of weights**

   8B means 8 billion weights. Together they decide how the model turns your text into an answer.

2. **Each weight is a small number**

   Usually between **−1 and 1**, and often negative, like −0.42 or 0.7372.

3. **Each weight uses a fixed number of bits**

   The bits decide how many different numbers a weight can be: **4 bits → 16**, **16 bits → 65,536**, **32 bits → ~4.3 billion** (2 × 2 × … once per bit).

4. **File size ≈ weights × bits ÷ 8 + small extra**

   ÷ 8 turns bits into bytes. 8 billion × 16 bits ÷ 8 = **16 GB**. 8 billion × 4 bits ÷ 8 = **4 GB**, + extra for block scales and file info ≈ **5 GB**.

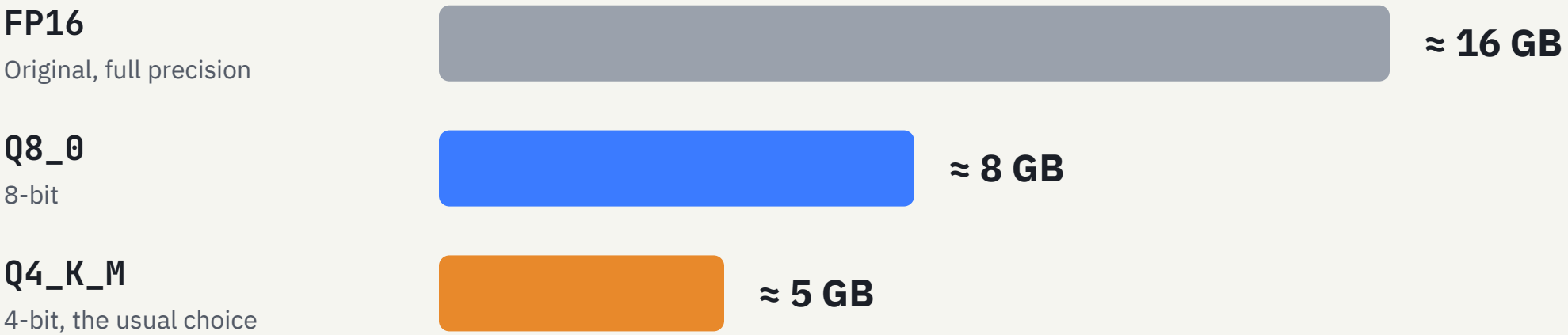
3 · COMPRESSION

# Quantization: making models smaller

**Why do it?** ✓ Fits in less memory ✓ Answers faster ⚠ Loses a little quality

Like saving a photo as a JPEG: the file gets much smaller, and you can barely see the difference.

**The same 8B model can be compressed (quantized) in multiple ways, for example:**



# How quantization works

## Rounding every weight number

A model is billions of weights, like **0.73720002**.

**32-bit:** 4.3 billion values → stored as 0.73720002



**16-bit:** 65,536 values → stored as 0.7373



**4-bit:** only 16 values → stored as 0.733



Like rounding prices to the nearest dollar: a tiny error on each number, but the file gets about 4× smaller.

## Q, INT or FP + a number = bits per weight

| Name | Bits | Possible numbers per weight |
|---|---|---|
| FP32 / F32 | 32 | ~4.3 billion |
| FP16 / BF16 | 16 | 65,536 (the original) |
| Q8 · INT8 · FP8 | 8 | 256 |
| Q6 | 6 | 64 |
| Q5 | 5 | 32 |
| Q4 · INT4 · FP4 | 4 | 16 (the usual choice) |
| Q3 | 3 | 8 |
| Q2 | 2 | 4 |

# Common compression levels

| Level | Size of an 8B model (grey = 100%) | Quality |
|---|---|---|
| FP16 | 100% · 16 GB | **Original quality** — No compression: the baseline we compare to |
| Q8_0 | ~50% · ~8 GB | **No visible loss** — Half the original size |
| Q5_K_M | ~35% · ~5.5 GB | **Almost perfect** — Use it if you have spare memory |
| Q4_K_M | ~30% · ~5 GB | **Tiny loss** — The community default — Recommended |
| Q3_K_M / Q2_K | ~20–25% · ~3.5 GB | **Noticeable loss** — Only to squeeze a big model onto small hardware |

3 · COMPRESSION

# The letters in Q4_K_M

## K = smart blocks

Weights are split into small blocks (e.g. 32 weights). Each block spreads its own 16 steps over its own range, which doesn't have to be centered on 0.

**Block A:** tiny, lopsided weights → steps ~0.005 apart




**Block B:** big weights → steps ~0.12 apart




Without blocks, Block A's tiny weights would all round to 0. The ranges cost a little extra: Q4_K_M ≈ 4.8 bits per weight.

`_0`

### Older, simple method

Stores only a scale, so the range is always centered on 0. Block A (–0.02 to 0.05) becomes **–0.05 to 0.05**:



The 5 grey steps (below –0.02) are never used. OK at 8 bits (256 steps), wasteful at 4 bits (16).

### Q4_K_M

= 4 bits · smart blocks · M (next slide)

# The letters in Q4_K_M

The S, M, L at the end of the name

## Some weights matter more

A model is made of different parts. A few of them hurt quality the most when rounded, so S, M and L give those parts **extra bits**:

| Part of the model | Q3_K_S | Q3_K_M | Q3_K_L |
|---|---|---|---|
| Most weights | 3 bits | 3 bits | 3 bits |
| Attention: value | 3 bits | 4 bits | 5 bits |
| Attention: output | 3 bits | 4 bits | 5 bits |
| Feed-forward: down | 3 bits | 4 bits | 5 bits |
| Final output layer | 6 bits | 6 bits | 6 bits |

Same base (3 bits) for most weights. Only the sensitive parts change: that is the whole difference.

## Same 8B model, real file sizes

| Name | File size | Quality |
|---|---|---|
| Q3_K_S | 3.7 GB | Lowest |
| Q3_K_M | 4.0 GB | Better |
| Q3_K_L | 4.3 GB | Best of Q3 |
| Q4_K_S | 4.7 GB | Good |
| Q4_K_M | 4.9 GB | Good+ (usual choice) |

**Rule of thumb**

Pick **M**. Pick **S** only if M doesn't fit in your memory.

# INT vs FP: two ways to store a number

## Same 8 bits, different steps

**INT8:** equal steps everywhere



**FP8:** tiny steps around 0, bigger steps toward both ends



–big **0** +big

A model’s numbers are mostly small, with a few big ones. FP fits that naturally; INT uses a scale to stretch its ruler.

## Same size, different labels

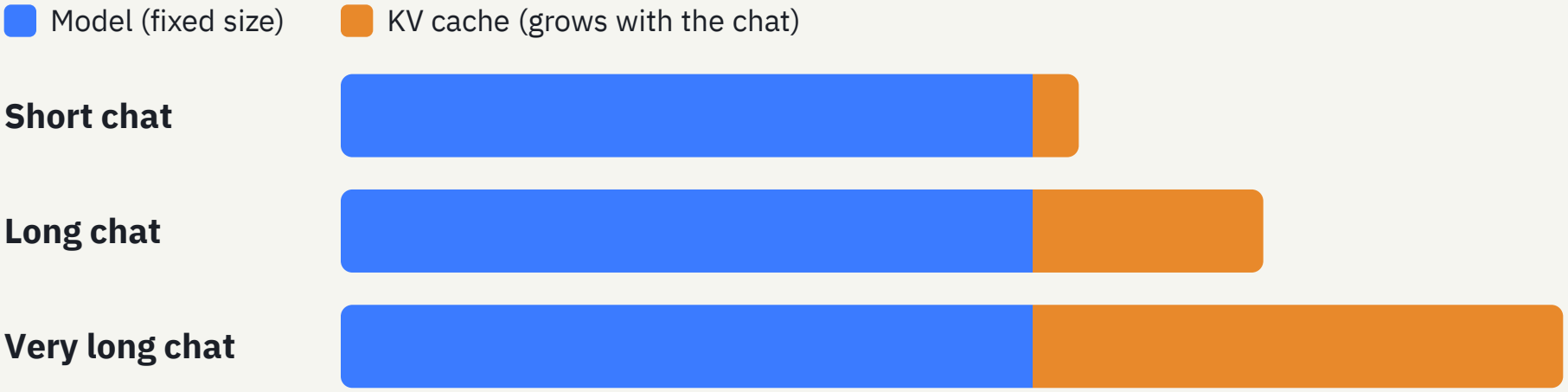
| Bits | Names you’ll see | Size |
|---|---|---|
| 16 | FP16 · BF16 | The original |
| 8 | Q8_0 · INT8 · FP8 | Half size |
| 4 | Q4_K_M · INT4 · FP4 | About ⅓ of the size |

Same bits = same size and similar quality. FP8 / FP4 need newer GPUs (e.g. RTX 40-series and up).

3 · COMPRESSION

# The KV cache: the model’s short-term memory

While you chat, the model remembers everything said so far. That memory also lives in VRAM.




⚠ Crashes in long chats? Shorten the context, or turn on KV cache compression (FP8 / INT4).

4

PART 4 OF 5

# Model names

## Decoding what you download

Reading a name on Hugging Face, MoE, and Base vs Instruct.

4 · MODEL NAMES

# How to read a model name

A typical name on Hugging Face, split into its parts below:

**bartowski/DeepSeek-R1-Distill-Qwen-32B-Instruct-GGUF-Q4_K_M**

| `bartowski/` | `DeepSeek-R1` | `Distill-Qwen` | `32B` | `Instruct` | `GGUF` | `Q4_K_M` |
|---|---|---|---|---|---|---|
| Who uploaded it | The teacher: gave its skills | The student: really a Qwen model | Size: 32 billion parameters | Tuned to chat | File type | Compression |

**DeepSeek on Qwen?** A Qwen model (the student) was trained on answers written by DeepSeek-R1 (the teacher). So it's a Qwen that learned to reason like DeepSeek.

Once you know the pattern, a long name tells you who made it, how big it is, and which engine can run it.

4 · MODEL NAMES

# Which file type is best for what?

## For your own computer

Start here

**GGUF · Q2 … Q8**

One file per level. Runs on almost anything and can split a model between GPU and CPU. → Ollama, llama.cpp, LM Studio

**MLX · 4bit · 8bit**

Apple's own format. Usually the fastest option on Apple Silicon Macs. → LM Studio, mlx-lm



## For GPU servers

**AWQ / GPTQ · Int4 · Int8**

4- or 8-bit formats for serving many users at once. → vLLM

**EXL2 · 2–8 bits**

Pick any size, like 4.2 bits, to fit one GPU's VRAM exactly. Very fast for one user. → ExLlamaV2

**FP8 · 8 bits**

Near-original quality at half the size. Needs RTX 40-series or newer. → vLLM

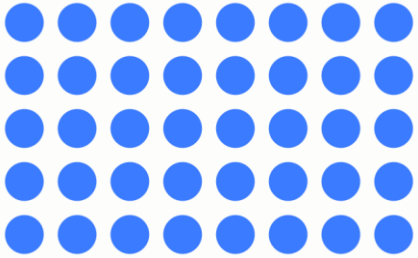


**Not compressed:** Safetensors (BF16) is the original model, used for fine-tuning and for making all of the versions above.

# Dense models vs. Mixture of Experts

## Dense

e.g. 8B, 32B, 70B



Every part of the model works on every word. Speed and memory both match its size.

## Mixture of Experts (MoE)

e.g. 35B-A3B, Mixtral 8x7B



Only a few “experts” work on each word (**A3B** = 3B active). Runs fast like a 3B, but needs memory like a 30B.

☆ **Quality:** 8B dense (~5 GB) < 30B-A3B MoE (~18 GB) < 32B dense (~19 GB). The MoE is the fastest of the three.

4 · MODEL NAMES

# Base, Instruct or Distill?

### Base

Raw autocomplete. It just continues your text instead of answering.

**Rarely what you want**

### Instruct / Chat / IT

Trained to follow instructions and hold a conversation. **IT** = Instruction-Tuned.

**Pick this for chat**

### Distill

A smaller “student” model trained on answers from a big “teacher” model.

**Big-model skills, small size**

</> **For coding:** look for “Coder” in the name, e.g. Qwen2.5-Coder-32B-Instruct.

5

PART 5 OF 5

# Finding models

## Where to get them, and your first run

Popular model families, licenses and getting started.

5 · FINDING MODELS

# Popular open-source model families

### Qwen

**Alibaba** · Qwen 3.x

Great at coding, maths and many languages. Lots of sizes.

### DeepSeek

**DeepSeek** · V4

Strong reasoning. Small distilled versions for home.

### Gemma

**Google** · Gemma 4

Small, efficient models (2B–31B). Laptop-friendly.

### Mistral

**Mistral AI** · Small 4

Efficient European models. Made MoE popular.

### Llama / Muse

**Meta** · Muse Glimmer

Llama 3 & 4 still widely used. Newer open models are called Muse.

### GLM

**Z.ai** · GLM-5.x

Popular for coding and agents. Very large; smaller Flash versions.

### Kimi

**Moonshot AI** · Kimi K3

One of the strongest open models. Very large.

### MiniMax

**MiniMax AI** · M3

Coding, agents and very long chats. Very large.

All of them are on **Hugging Face**. “Very large” = server hardware, not a home PC.

# Open-source models beyond text

## Images

**FLUX.2** · **Black Forest Labs**
Top image quality. The [dev] version is non-commercial.

**Qwen-Image** · **Alibaba**
Great at writing text inside images.

**Stable Diffusion** · **Stability AI**
The classic, with a huge community of add-ons.

## Video

**Wan 2.2** · **Alibaba**
Text or image → short video clips.

**LTX-2** · **Lightricks**
Fast, and can add sound to the clip.

**HunyuanVideo** · **Tencent**
High-quality clips; needs a big GPU.

## Audio

**Whisper** · **OpenAI**
Speech → text, in many languages.

**Kokoro** · **hexgrad**
Text → speech. Tiny, even runs on a CPU.

**Chatterbox** · **Resemble AI**
Text → speech that can copy a voice.

Images and video usually run in **ComfyUI**, a free visual app. Video needs much more VRAM than images.

5 · FINDING MODELS

# Check the license before you use it

### Permissive

`Apache 2.0 · MIT`

Use it for anything, including business. Change it and share it freely.

### Community

`e.g. Llama Community License`

Free to use, with conditions — like a limit of 700 million monthly users, and giving credit.

Find the license on the model's page, in its **Model Card**.

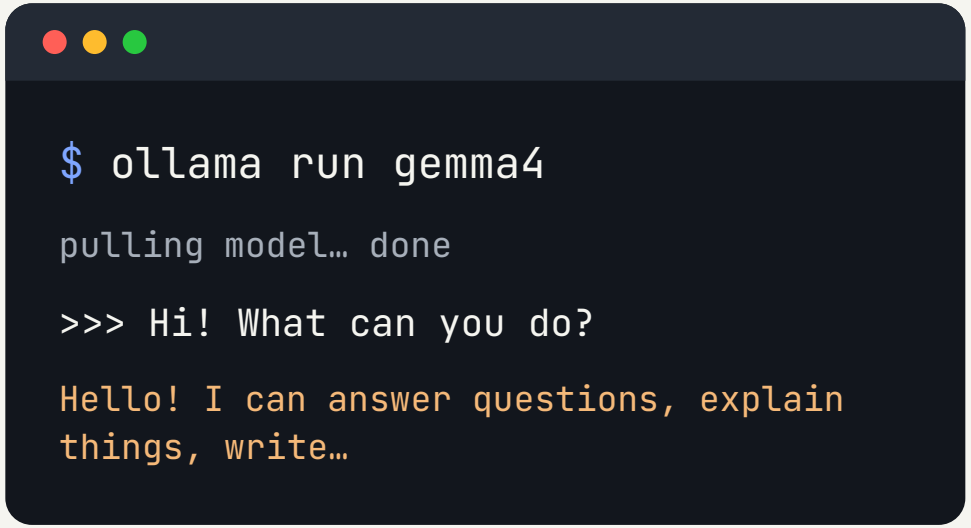
TRY IT YOURSELF

# Your first local model in 3 steps

1. **Install Ollama**
   Download the installer from ollama.com.

2. **Run one command**
   Open a terminal and type the command on the right.

3. **Chat**
   Ollama downloads the model (~10 GB), then you can talk to it.




Want more choice? On Hugging Face, filter by **GGUF** and pick trusted uploaders: **bartowski**, **unsloth**, **mradermacher**.

RECAP

# Five things to remember

1. Memory size decides **if** a model fits; memory speed decides **how fast** it answers.
2. Fits in GPU VRAM = fastest. Too big? Offload to RAM (slow), add a second GPU, or use a Mac.
3. A model is billions of **weights**. Quantization stores each one with fewer bits: smaller and faster, with a little less quality.
4. Start with **Ollama** and a **Q4_K_M**, **Instruct** model (for example **gemma4**).
5. Check the **license** in the Model Card before using a model.