Running LLMs Locally: From Cloud to Laptop

Discover how to harness the power of large language models on your own hardware. This practical guide covers the fundamentals of LLM architecture, demonstrates live model switching with Ollama, and shows how to migrate a production RAG application from cloud to local inference—all running on standard hardware.

FOUNDATIONS

LLMs: A 3-Minute Mental Model

At their core, large language models predict the next token in a sequence. Text is broken into tokens, converted to numerical embeddings, and processed through transformer layers. The attention mechanism determines which parts of the input matter most for generating each subsequent token.

Think of it as autocomplete trained on the internet—sophisticated pattern matching that generates human-like text one piece at a time.

01

Tokenisation

Text splits into discrete units

02

Embeddings

Tokens become numerical vectors

03

Attention

Model weighs relevance of context

04

Generation

Next token predicted and output

Why Run LLMs Locally?

Privacy

Sensitive data never leaves your infrastructure. Perfect for regulated industries or confidential work.

Cost Control

No per-token billing or API limits. Predictable infrastructure costs only.

Low Latency

Eliminate network hops and API rate limits for faster responses.

Full Control

Choose your models, prompts, and upgrade schedule without vendor constraints.

Offline Ready

Works in air-gapped environments or when cloud services are unavailable.

Key trade-off: Cloud delivers raw power, local delivers control.

REALITY CHECK

When Local LLMs Make Sense

Local LLMs excel in specific scenarios where control, privacy, or cost matters more than cutting-edge capabilities. Set realistic expectations: local models won't replace GPT-4 for complex reasoning, but they're brilliant for focused applications.

- **Internal Tools & Prototypes**

  Rapid experimentation without cloud dependencies

- **Knowledge Assistants**

  Search through documentation, policies, or personal notes

- **Regulated Data**

  Healthcare, finance, legal—industries with strict compliance

- **Developer Productivity**

  Code completion, documentation generation, debugging help

- **Edge Environments**

  Offline access or devices with limited connectivity

DEMO

Installing & Running Ollama

Ollama makes running local LLMs remarkably simple. It handles model downloads, caching, and exposes a REST API—all optimised for both CPU and GPU execution.

1

Install Ollama

```
ollama install
```

Single command installation for macOS, Linux, or Windows

2

Download a Model

```
ollama pull llama3
```

Models are cached locally and versioned automatically

3

Run Interactive Chat

```
ollama run llama3
```

Start chatting immediately—models load and respond in seconds

Live demo: Run `ollama run llama3` and ask a simple question to see instant local inference.

Demo Setup: Pre-Built Cloud RAG Chatbot

What We've Already Built

A production-ready document-specific chatbot powered by Retrieval Augmented Generation. This system grounds LLM responses in your specific knowledge base, reducing hallucinations and ensuring accurate, contextual answers.

1

Document Chunking

Content split into searchable segments

2

Vector Embeddings

Chunks converted to semantic vectors

3

Similarity Search

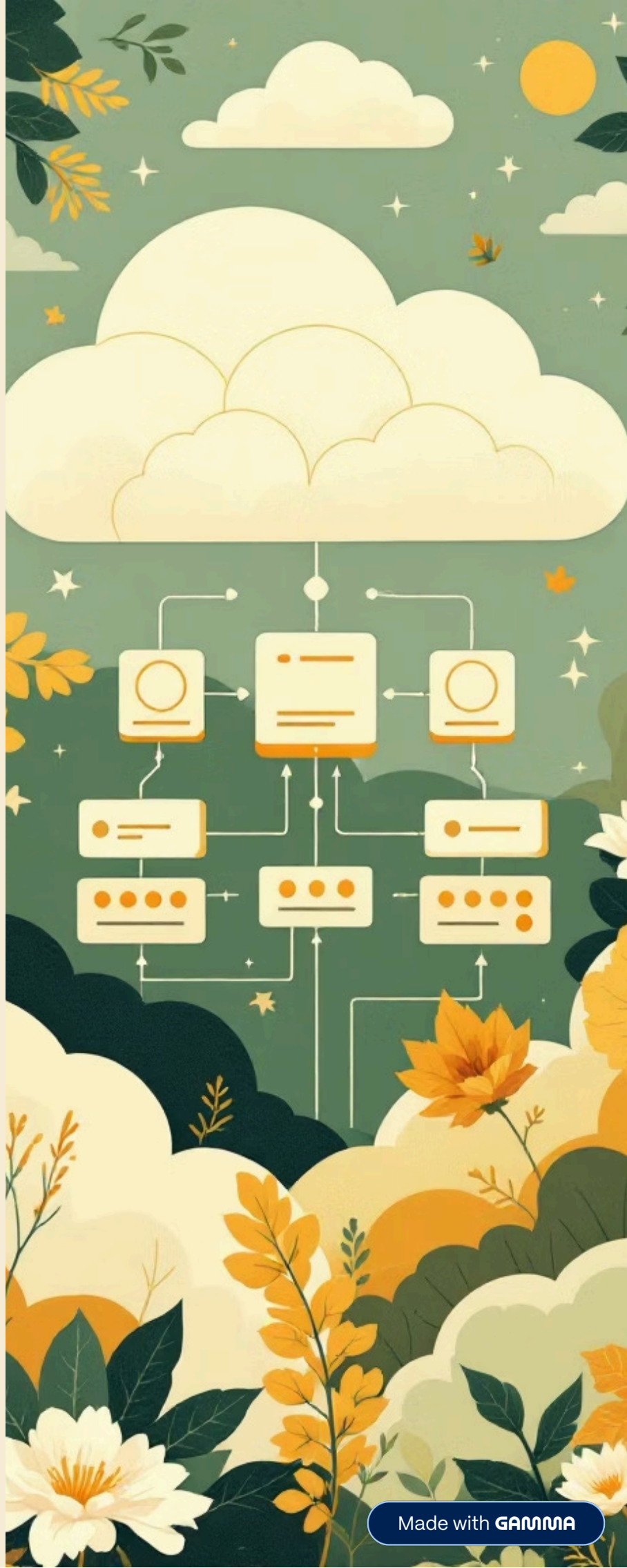
Retrieve relevant context from vector store

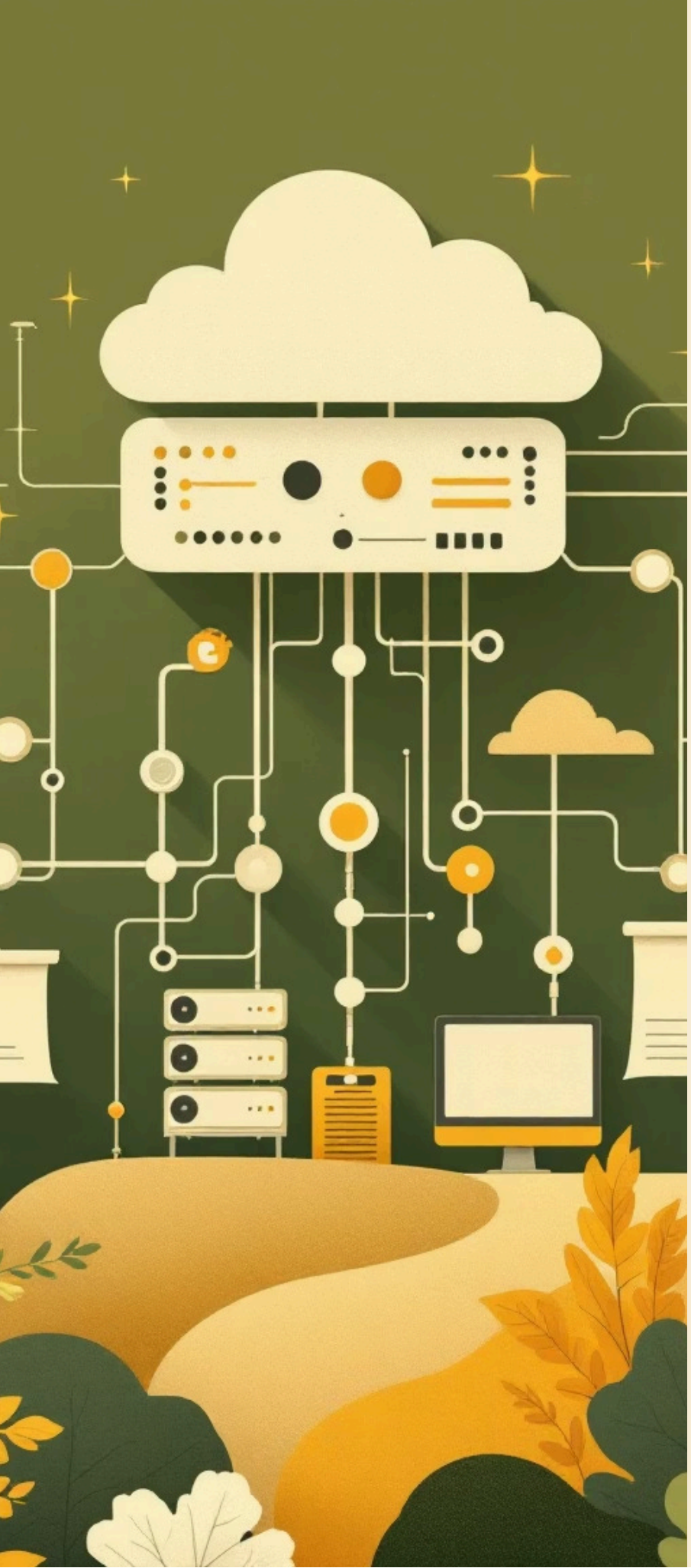
4

Cloud LLM Inference

OpenAI or Claude generates grounded responses

Demo pointer: Ask domain-specific questions and observe how retrieved context grounds the answers.





☆ CRITICAL INSIGHT

Switching from Cloud to Local LLM

What Stays the Same

- Document chunking logic
- Embedding generation
- Vector store queries
- RAG pipeline architecture
- Prompt templates

What Changes

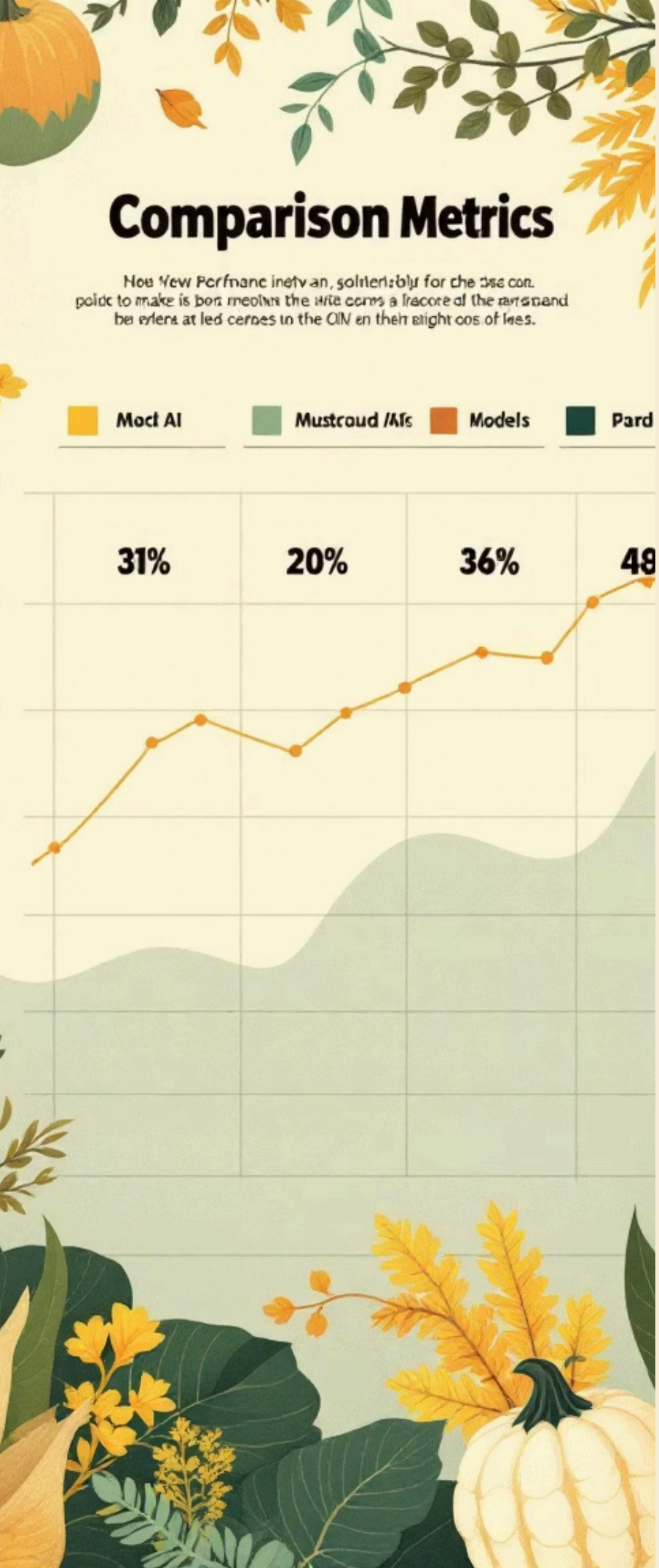
Only the inference call:

```
# Cloud
response = openai.chat.completions.create(...)

# Local
response = ollama.chat(model="llama3", ...)
```

Same application. Same data. Different inference engine.

The entire RAG pipeline remains unchanged—you're simply swapping where the final text generation happens.



LIVE DEMO

Switching Between Local Models

One of the most powerful aspects of local LLMs is the ability to instantly switch between different models. Compare speed, quality, and behaviour across architectures—all without changing your code or incurring API costs.

```
ollama run mistral
ollama run phi
ollama run llama3
```

Llama 3

Balanced performance and capability for general tasks

Mistral

Optimised for speed with strong reasoning abilities

Phi

Tiny model perfect for resource-constrained environments

Quantised Variants

Reduced precision models trading accuracy for efficiency

Demo approach: Ask the same question to each model and compare response speed, quality, and tendency to hallucinate.

Why Fine-Tuning Matters Locally

Cloud models like GPT-4 have enormous capacity and broad knowledge. Local models are smaller and less capable out of the box. Fine-tuning bridges this gap, transforming a general-purpose model into a specialist that understands your domain, matches your tone, and formats responses consistently.

Improved Accuracy

Models learn domain-specific terminology, concepts, and factual knowledge from your data.

Consistent Tone

Fine-tuning ensures responses match your organisation's voice and communication style.

Structured Formats

Train models to output JSON, markdown, or other structured formats reliably.

> For local LLMs, fine-tuning isn't a luxury—it's a **necessity** for production quality.

Fine-Tuning Strategies for Limited Hardware

What Works Locally

LoRA & QLoRA

Parameter-efficient methods that fine-tune small adapter layers instead of the entire model, drastically reducing memory requirements.

Small Base Models

Start with 7B–13B parameter models that fit comfortably in consumer GPU memory whilst still delivering strong performance.

Curated Datasets

Use narrow, high-quality datasets (500–5000 examples) focused on your specific use case rather than massive general datasets.

What to Avoid

Full Model Training

Updating all parameters requires enormous compute and memory—impractical for local hardware.

Huge Datasets

Millions of examples lead to prohibitively long training times on consumer hardware.

Long Context Tuning

Extending context windows requires substantial resources and often yields diminishing returns.

Best practice: Compare multiple base models first, then fine-tune your winner. Don't waste time tuning the wrong foundation.