

# LLM Basics - A DevOps Engineer's Guide

---

## What is an LLM?

### Simple Explanation

An **LLM (Large Language Model)** is an AI program trained on billions of text examples that can:

- Understand questions and requests written in natural language
- Generate human-like text responses
- Perform various language tasks (summarization, translation, coding, etc.)

Think of it like a very smart autocomplete system that understands context and can have intelligent conversations.

## Real-World Examples

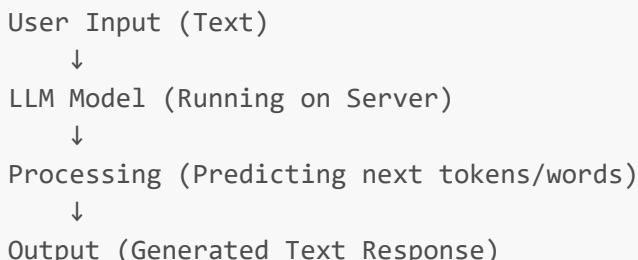
Model	Company	Use Case
GPT-4	OpenAI	General-purpose AI assistant
Claude	Anthropic	Customer service, content creation
Gemini	Google	Multi-modal AI (text + images)
Llama 2	Meta	Open-source general-purpose model
Mistral	Mistral AI	Efficient, cost-effective model

## How LLMs Work (High-Level Overview)

### Training Phase (Don't worry about this)

- Model learns patterns from massive amounts of text data
- This happens BEFORE deployment, not your responsibility
- Training is done by ML specialists

### Inference Phase (This is what you deploy)



**Token:** A token is a small piece of text (word, punctuation, or number). Models process text in tokens.

# Key Characteristics That Affect Deployment

## 1. Model Size

- **Small models** (7B parameters): Cheaper, faster, less GPU needed
- **Large models** (70B+ parameters): More accurate, expensive, needs powerful GPUs
- Parameters = learned weights in the model (more = smarter but heavier)

## 2. Latency Requirements

- **Real-time** (< 1 second): Chat applications, instant search
- **Near real-time** (1-5 seconds): Content generation, summarization
- **Batch** (minutes to hours): Large document processing, report generation

## 3. Throughput Requirements

- **Concurrent users:** How many people using it simultaneously?
- **Requests per second:** How many API calls per second?
- This determines how many replicas/instances you need

## 4. Hardware Requirements

Different models need different compute:

- **CPU-only deployment:** Small models, non-time-critical
- **Single GPU (NVIDIA T4/L4):** Medium models, light workloads
- **Multi-GPU (A100/H100):** Large models, high throughput
- **TPU:** Google's custom chips for even more performance

# LLM Deployment Patterns

## Pattern 1: Using Pre-Built Managed Services

```
User Request → AWS Bedrock / Azure OpenAI / Google Vertex AI  
→ Model already running in cloud → Response
```

**Pros:** Easy, no infrastructure to manage

**Cons:** More expensive, limited customization

## Pattern 2: Self-Hosted on Compute

```
User Request → Your API (Flask/FastAPI) → Model on GPU Instance  
→ Response
```

**Pros:** Cost-effective for high volume, full control

**Cons:** Need to manage infrastructure, scaling, updates

## Pattern 3: Containerized Deployment

```
User Request → Kubernetes Cluster → Container with Model  
→ Response
```

**Pros:** Auto-scaling, easy updates, multi-region

**Cons:** More complex to manage

## Pattern 4: Batch Processing

```
Documents → Queue (SQS/Pub-Sub) → Batch Processing Job  
→ Database/Storage → Results retrieved later
```

**Pros:** Cost-effective for non-real-time work

**Cons:** Not suitable for interactive use

# Common Deployment Scenarios

## Scenario 1: Internal Chatbot

- **Users:** 50-200 concurrent employees
- **Pattern:** Self-hosted on single GPU
- **Cost:** ~\$1,000-3,000/month
- **Example:** HR Q&A, internal documentation search

## Scenario 2: Customer-Facing Chat

- **Users:** 1,000+ concurrent
- **Pattern:** Managed service or large compute cluster
- **Cost:** \$10,000-50,000+/month
- **Example:** Customer support chatbot

## Scenario 3: Content Generation API

- **Users:** Application backend
- **Pattern:** Batch processing or async queue
- **Cost:** Pay per use (cheaper than real-time)
- **Example:** Generate product descriptions, blog posts

## Scenario 4: Document Analysis

- **Processing:** Large document batches
- **Pattern:** Batch jobs, scheduled tasks
- **Cost:** Very economical (~\$100-500/month)
- **Example:** Resume screening, contract analysis

# Key Metrics You'll Encounter

Metric	Meaning	Why It Matters
<b>Tokens/sec</b>	Words generated per second	Determines user experience latency
<b>Throughput</b>	Requests processed per hour	How many users can be served
<b>P95 Latency</b>	95% of requests finish within this time	Performance SLA
<b>VRAM/Memory</b>	GPU memory needed	Determines what hardware you need
<b>Context Window</b>	How many tokens the model can "remember"	Max document length it can process

## What You DON'T Need to Know

- **How the model learns:** Not your concern as DevOps
- **Mathematical details:** Neural networks, transformers, etc.
- **Training process:** That's ML engineers' job
- **Model architecture internals:** You deploy it, not build it

## What You DO Need to Know

- How much compute/GPU it needs
- How long requests take (latency)
- How many concurrent users it supports
- Cost per hour/month
- How to scale it up/down
- How to monitor its performance
- How to update model versions

## Simple Analogy

Think of LLM deployment like deploying a database server:

- **Database:** Stores data, responds to queries
- **LLM:** Generates responses based on learned patterns

Both need:

- Enough compute resources
- Fast response times
- Ability to scale
- Monitoring and alerting
- Cost optimization
- Security measures

The main difference: LLMs need GPUs while databases might only need CPUs.

## Next Steps

Now that you understand what LLMs are, learn about:

1. **Cloud Architecture Overview** → See [02-Architecture-Overview.md](#)
  2. **Your specific platform:** AWS, Azure, or GCP
- 

**Key Takeaway:** LLMs are sophisticated AI models, but from a DevOps perspective, they're just applications that need compute, scaling, monitoring, and cost management like any other service.