

# 1.0 LLM CONCEPTS & ARCHITECTURE

Welcome to Day 1 of the watsonx Workshop! Today we'll explore Large Language Models (LLMs), their architecture, and how to work with them effectively.

# LEARNING OBJECTIVES

By the end of this module, you will:

- Understand core LLM terminology and constraints
- Compare local vs managed LLM deployment models
- Know how LLMs fit into production architectures
- Understand key parameters that control model behavior

# WHAT IS A LARGE LANGUAGE MODEL?

A Large Language Model (LLM) is a neural network trained on vast amounts of text data to understand and generate human-like text. Think of it as a powerful pattern-matching engine that has learned the statistical relationships between words, phrases, and concepts.

## KEY CHARACTERISTICS

**Scale:** LLMs contain billions of parameters (weights) that encode knowledge learned from training data. For example: - GPT-3: 175 billion parameters - Llama 3.2: 1-3 billion parameters (smaller variants) - Granite 13B: 13 billion parameters

**Training Data:** Models are trained on diverse text sources including: - Books, articles, and documentation - Web pages and forums - Code repositories - Scientific papers

**Capabilities:** Modern LLMs can: - Answer questions - Summarize documents - Write code - Translate languages - Extract structured information - Reason through problems (with varying degrees of success)

# KEY CONCEPTS

## TOKENS & TOKENIZATION

LLMs don't work with words—they work with **tokens**. A token is a sub-unit of text that the model processes.

**Examples:** - "Hello" → 1 token - "watsonx.ai" → might be 2-3 tokens (depends on the tokenizer) - "AI" → 1 token  
- A space or punctuation can be its own token

**Why this matters:** - Models have **token limits** (context windows) - API costs are often calculated per token -  
Long documents need to be chunked to fit within token limits

**Rule of thumb:** - English: ~4 characters per token on average - Code: Often more tokens per line than natural language

## CONTEXT WINDOW AND TRUNCATION

The **context window** is the maximum number of tokens a model can process at once. This includes both: -  
Your input (prompt) - The model's output (completion)

**Common context window sizes:** - Llama 3.2 (1B): 128K tokens - Granite 13B: 8K tokens (some variants) - GPT-4: 8K-32K tokens (depending on version)

**Truncation:** If your input exceeds the context window, it gets truncated (cut off), which can lead to: - Missing important context - Incomplete responses - Errors

# LOCAL VS MANAGED LLMS

You have two main deployment options for LLMs in production:

## LOCAL LLMS (E.G., OLLAMA)

**What it is:** Running models on your own infrastructure (laptop, on-prem servers, private cloud).

**Pros:** - **Privacy:** Data never leaves your environment - **Control:** Full control over model versions and updates - **Cost:** No per-token API charges after initial setup - **Customization:** Can fine-tune models for specific use cases - **Offline:** Works without internet connectivity

**Cons:** - **Hardware requirements:** Need GPUs for acceptable performance - **Maintenance:** You manage infrastructure, updates, scaling - **Limited scale:** Constrained by your hardware resources - **Model selection:** Limited to models that fit in your hardware

**Best for:** - Prototyping and development - Privacy-sensitive applications - Organizations with existing GPU infrastructure - Small to medium workloads

**Example tools:** - **Ollama:** Easy local LLM management - **LM Studio:** GUI for local models - **vLLM:** High-performance inference server

## MANAGED LLMS (E.G., WATSONX.AI)

**What it is:** Using LLMs via cloud APIs where the provider handles infrastructure.

# COST & RESOURCE CONSIDERATIONS

## GPU VS CPU

**GPU (Graphics Processing Unit):** - Designed for parallel computations - Essential for training LLMs - Greatly accelerates inference (10-100x faster than CPU) - **Cost:** \$1,000 - \$10,000+ per card (consumer to enterprise)

**CPU (Central Processing Unit):** - General-purpose computing - Can run small models (1-3B parameters) acceptably - Struggles with larger models (13B+) - **Cost:** Cheaper, already available in most systems

**Memory requirements:** - Rough estimate: Model needs ~2 bytes per parameter (for FP16) - Example: 13B model needs ~26 GB GPU memory

## CLOUD COST DIMENSIONS

When using managed services like watsonx.ai:

**Token-based pricing:** - Input tokens: Text you send - Output tokens: Text generated - Typically: \$0.0001 - \$0.001 per token (varies by model)

**Example calculation:**

Prompt: 1,000 tokens

Response: 500 tokens

Cost:  $(1000 + 500) \times \$0.0002 = \$0.30$  per request

**Cost optimization strategies:** - Use smaller models when appropriate - Cache common responses -

# WHERE THE ACCELERATOR FITS ARCHITECTURALLY

Throughout this workshop, we'll reference the **RAG Accelerator**—a production-ready skeleton for building LLM applications. Here's how it's structured:

## CORE ARCHITECTURE

```
accelerator/
├── rag/                      # RAG core logic
│   ├── pipeline.py            # Orchestrates retrieval + LLM
│   ├── retriever.py          # Vector DB queries (Elasticsearch/Chroma)
│   ├── prompt.py              # Shared prompt templates
│   └── embedder.py            # Text embedding logic
├── service/                  # Production API
│   ├── api.py                 # FastAPI microservice (POST /ask)
│   ├── deps.py                # Configuration & dependencies
│   └── models.py              # Request/response schemas
└── tools/                    # CLI utilities
    ├── chunk.py               # Document chunking
    ├── extract.py              # Text extraction from PDFs, docs
    ├── embed_index.py          # Embedding and indexing pipeline
    └── eval_small_nv           # Evaluation harness
```

## HOW LLMS FIT IN

On Day 1, we're focusing on **pure LLM behavior** (no retrieval). This maps to:

**Current state (Day 1):**

```
# pipeline.py (simplified)
def answer_question(question: str) -> str:
    # Direct LLM call
```

# REFERENCE NOTEBOOKS

The workshop includes several reference notebooks that show LLMs in production contexts:

## RAG EXAMPLES (`labs-src/`)

- `use-watsonx-elasticsearch-and-langchain-to-answer-questions-rag.ipynb`
  - Full RAG pipeline with Elasticsearch
  - Shows prompt structure with context
- `use-watsonx-chroma-and-langchain-to-answer-questions-rag.ipynb`
  - Alternative vector DB (Chroma)
  - LangChain integration patterns

## ACCELERATOR NOTEBOOKS (`accelerator/assets/notebook/`)

- `QnA_with_RAG.ipynb`
  - End-to-end Q&A with retrieval
  - Prompt engineering for RAG
- `Create_and_Deploy_QnA_AI_Service.ipynb`
  - Deploy RAG service to production
  - API endpoint creation

**How to use these:** - Don't run them line-by-line on Day 1 - Do open them to see: - How prompts are structured

How LLM calls are instrumented - How outputs are validated

# HOW THIS CONNECTS TO THE LABS

## DAY 1 LABS (TODAY)

- **Lab 1.1:** Quick start with both Ollama and watsonx
  - Focus: Basic LLM calls, parameter tuning
  - No retrieval, just prompts → responses
- **Lab 1.2:** Prompt templates
  - Build reusable prompt patterns
  - Compare behavior across backends
- **Lab 1.3:** Micro-evaluation
  - Rate LLM outputs
  - Build a simple evaluation framework

## DAY 2-3 LABS (UPCOMING)

- Add retrieval (RAG)
- Integrate with the accelerator
- Build production-ready pipelines
- Add orchestration and agents

Mental model: - Day 1 = Understanding the LLM building block - Day 2 = LLM + retrieval (RAG) - Day 3 = LLM + retrieval + orchestration (agents)

# FURTHER READING

## OFFICIAL DOCUMENTATION

- IBM Granite Models
- watsonx.ai Documentation
- Ollama Documentation

## PROMPT ENGINEERING

- OpenAI Prompt Engineering Guide
- Anthropic Prompt Engineering
- Granite Prompting Guide

## LLM CONCEPTS

- Hugging Face NLP Course
- LLM Training & Inference
- Understanding Tokenization

## RESPONSIBLE AI

- IBM AI Ethics
- Guardrails for LLMs

# SUMMARY

You now understand:

-  What LLMs are and how they work at a high level
-  Key concepts: tokens, context windows, temperature, top-k/top-p
-  Trade-offs between local and managed deployments
-  Cost considerations for LLM applications
-  How LLMs fit into the accelerator architecture

Next: Let's get hands-on with Lab 1.1 and actually run some prompts!