COURSE MATERIAL

Foundations of LLM Applications & Agentic AI Systems

A Structured 10-Module Learning Programme

MODULES COVERED

1. Artificial Intelligence Foundations
2. Machine Learning & Deep Learning
3. From Sequence Models to Transformers
4. Pre-training, Transfer Learning & Fine-Tuning
5. Retrieval-Augmented Generation (RAG)
6. What Is an AI Agent?
7. Agentic Workflows & Reasoning Patterns
8. Model Context Protocol (MCP)
9. Designing LLM Applications (Demo → Production)
10. Evaluation & Benchmarks

Table of Contents

MODULE 1 | Artificial Intelligence Foundations

1.1 What is Artificial Intelligence?

Artificial Intelligence (AI) refers to the design of systems that can perform tasks that typically require human intelligence.

However, this definition is too informal for serious study. A more structured definition:

> *Artificial Intelligence is the engineering of goal-directed systems that perceive information, optimize decisions, and act under constraints.*

Three key elements appear repeatedly in every AI system:

- Perception – receiving input from the environment
- Reasoning / Optimization – selecting the best action
- Action – producing an output that influences the environment

AI is not magic. It is applied mathematics + computation + data.

1.2 Categories of AI Systems

AI systems have evolved through distinct paradigms, each with increasing capability and complexity.

Rule-Based Systems (Symbolic AI)

- Based on explicit, human-defined logic
- Uses IF–THEN conditional statements
- Produces deterministic, fully predictable behavior

Example:

```
IF temperature > 38°C THEN diagnose fever
```

Strengths: Transparent, predictable, auditable. Limitations: Cannot scale to complex, high-dimensional problems; no learning ability.

Statistical Systems

Statistical systems rely on probability and data modelling. Instead of strict rules, they estimate likelihoods and make decisions under uncertainty. A core conceptual idea is conditional probability:

$$P(Y \mid X)$$

This reads: What is the probability of outcome Y given input X? Example application: spam detection using probabilistic models.

Learning Systems (Machine Learning)

Modern AI systems learn patterns directly from data. Instead of writing rules manually, a machine learning system defines a model, provides data, and optimises parameters. This leads to the central mathematical formulation of AI:

$$\theta^* = \arg\min_\theta \ell(\theta)$$

Where:

- θ (theta) = model parameters
- ℓ(θ) (Loss) = the loss function measuring error
- θ* = the optimal parameters that minimise loss

Core Insight: AI is fundamentally about minimising error through systematic optimisation.

1.3 Narrow AI vs General Intelligence

Narrow AI (ANI)

Designed for specific, well-defined tasks. Examples include ChatGPT, image classifiers, and recommendation systems. These systems are highly capable but strictly domain-specific.

Artificial General Intelligence (AGI)

A hypothetical system capable of performing across all intellectual tasks at or above human level. AGI does not currently exist.

Important: Large Language Models remain Narrow AI systems, even when they appear general-purpose. They are highly specialised predictive engines.

1.4 What Makes a System Goal-Directed?

A system is goal-directed if it evaluates possible outcomes and selects actions to maximise objective fulfilment. In reinforcement learning, this is formalised as:

$$\pi^* = \arg\max_\pi {}^{\pi}C[R]$$

Where:

- π (pi) = policy — the decision rule
- R = reward signal
- ${}^{\pi}C$ = expected value operator

This means: choose actions that maximise expected cumulative reward. Modern LLM-based agents approximate this idea through prompt-guided reasoning and tool use.

1.5 Optimisation as the Core of Intelligence

At its foundation, all AI systems optimise something — classification error, prediction accuracy, reward, likelihood, or alignment objectives. Even large language models are trained to minimise next-token prediction error using cross-entropy loss:

$$\ell = -\sum y_i \cdot \log(\hat{y}_i)$$

Interpretation: The model is penalised when its predicted probability ŷ differs from the true distribution y. Intelligence in machines is therefore largely optimisation under constraints.

1.6 Intelligence as Search Under Constraints

AI systems search through possible actions or outputs and choose optimal ones. Real-world constraints include limited data, limited compute, limited memory (context window), safety boundaries, and real-time latency requirements. In agentic systems, this manifests as:

Perception → Planning → Action → Feedback → Update

1.7 Why This Matters for LLM Applications

Understanding AI foundations clarifies the purpose and architecture of modern systems:

- LLMs are optimisation engines trained on token prediction.
- Agents extend LLMs with perception, memory, and action capabilities.
- RAG improves knowledge accuracy by grounding generation in retrieved context.
- Fine-tuning adjusts the optimisation objective for specific domains.

> *Without understanding AI as optimisation, advanced systems appear mysterious. With it, they become structured engineering problems.*

1.8 Common Misconceptions About AI

1. AI "thinks" like humans — False. AI detects statistical patterns; it does not reason as humans do.
2. AI understands meaning intrinsically — Debatable; current systems operate on pattern matching without grounded semantics.
3. AI is conscious — No scientific evidence supports this.
4. AI systems are always objective — False; they inherit the biases present in training data.

1.9 Module 1 Summary

In this module, you learned:

- AI systems are goal-directed optimisation systems.

- AI evolved from rule-based systems to statistical and then learning-based systems.
- Machine learning formalises intelligence as loss minimisation.
- LLMs are Narrow AI systems trained through large-scale optimisation.
- Agents expand AI systems into action-taking entities operating in environments.

📝 Practice Questions

1. What differentiates rule-based AI from learning-based AI?

A) Rule-based AI uses neural networks
B) Learning-based AI encodes explicit IF-THEN logic
C) Rule-based AI is hand-crafted; learning-based AI adapts from data
D) There is no practical difference

2. What does the expression $\theta^* = \text{argmin}_\theta \ell(\theta)$ represent?

A) The largest possible parameter set
B) The parameters that minimise the loss function
C) The gradient of the loss
D) The output distribution of the model

3. Why are LLMs considered Narrow AI?

A) They use small parameter counts
B) They are trained only on images
C) They are specialised predictive engines, not general reasoners
D) They cannot generate text

4. What does it mean for a system to be goal-directed?

A) It always produces deterministic outputs
B) It evaluates outcomes and selects actions to fulfil an objective
C) It uses symbolic logic exclusively
D) It does not require training data

🤔 Reflection Prompts

5. *Why might AI systems appear to 'understand' without actually possessing semantic comprehension?*
6. *What real-world constraints most significantly limit AI performance in your domain?*
7. *Why is optimisation central to intelligence, both biological and artificial?*
8. *How does framing AI as 'search under constraints' change how you approach system design?*

MODULE 2 | Machine Learning & Deep Learning

2.1 What is Machine Learning?

Machine Learning (ML) is a subfield of AI focused on building systems that learn patterns from data rather than being explicitly programmed with rules. Formally, a model learns a function:

$$\hat{y} = f(x; \theta)$$

Where x = input, θ = model parameters, and ŷ = predicted output. The training objective is:

$$\theta^* = \text{argmin}_\theta \, \ell(y, \hat{y})$$

The model is trained to minimise the discrepancy between its predictions ŷ and the ground-truth labels y.

2.2 Types of Machine Learning

Supervised Learning

Training data has labels. The model learns a mapping from input to output. Examples: image classification, spam detection, sentiment analysis. A common loss function is cross-entropy:

$$\ell = -\Sigma \, y_i \cdot \log(\hat{y}_i)$$

Unsupervised Learning

No labels provided. The model discovers structure in data. Examples: clustering, dimensionality reduction, anomaly detection. Objective: minimise reconstruction error or maximise data likelihood.

Reinforcement Learning (RL)

An agent interacts with an environment and learns from reward signals. Goal: maximise cumulative expected reward.

$$\pi^* = \text{argmax}_\pi \, ^{\circ}C[R]$$

Used in robotics, game-playing agents, and RLHF for aligning LLMs with human preferences.

2.3 From Linear Models to Neural Networks

Linear Model

$$\hat{y} = Wx + b$$

This models simple linear relationships between input and output. However, most real-world relationships are nonlinear, which is where neural networks provide significant advantages.

Neural Networks

Neural networks stack nonlinear transformations:

$$a = \sigma(Wx + b)$$

Where σ is an activation function (such as ReLU or sigmoid) that introduces nonlinearity. Stacking multiple layers yields:

$$a^{-l} = \sigma(W^{-l} \cdot a^{-l-1} + b^{-l})$$

This enables the network to learn hierarchical representations — a process called representation learning.

2.4 Representation Learning

In traditional machine learning, humans design features manually (feature engineering). In deep learning, the model learns features automatically from raw data.

Example — image classifier:

- Early layers detect edges and textures
- Middle layers detect shapes and parts
- Later layers detect full objects and semantic concepts

> *Deep Learning = Automated hierarchical feature learning. This is the fundamental reason depth matters.*

2.5 Why Depth Matters

Deeper networks can represent more complex functions and approximate highly nonlinear mappings. Each layer transforms the input into a progressively more abstract representation. This gives deep models superior performance on complex tasks compared to shallow models with equivalent parameter counts.

2.6 Comparative Overview

Feature	Traditional ML	Deep Learning
Feature Engineering	Manual	Automatic
Data Requirement	Moderate	Large
Interpretability	Higher	Lower
Compute Requirement	Lower	High
Scalability	Limited	High

Feature	Deep Learning	Large Language Models
Scope	Broad technique	Specific architecture type
Data Type	Any (images, text, audio)	Primarily text (multimodal variants exist)
Training Objective	Task-specific	Next-token prediction
Scale	Small to large	Extremely large
Adaptability	Moderate	Very high

> *Important: LLMs are a specialised category within deep learning, not a separate paradigm.*

2.7 Mini Case Study: Spam Detection Evolution

This case study illustrates increasing abstraction and generalisation across AI generations:

9. Rule-Based: Handcrafted spam keywords — easy to bypass
10. Traditional ML: Logistic regression on bag-of-words features
11. Deep Learning: Neural networks learning word embeddings
12. LLM-Based: Context-aware detection understanding semantic meaning and paraphrase patterns

2.8 How This Connects to LLM Applications

Understanding ML and deep learning helps clarify:

- Why LLMs require massive training datasets
- Why scaling model size improves performance
- Why word and sentence embeddings work
- Why RAG and fine-tuning are effective

2.9 Module 2 Summary

In this module, you learned:

- Machine learning formalises pattern learning through optimisation
- Neural networks enable representation learning via nonlinear transformations
- Depth increases expressive power
- Deep learning automates feature extraction
- LLMs are a scaled, text-specialised application of deep learning

Learning Type	Data Labels	Feedback	Goal
Supervised	Yes	Error signal	Predict correct labels
Unsupervised	No	Structure metrics	Discover patterns
Reinforcement	Reward signal	Delayed reward	Maximise cumulative reward

📝 Multiple Choice Questions

1. What introduces non-linearity in neural networks?

A) Weights
B) Bias terms
C) Activation functions
D) Loss functions

2. In supervised learning, the model:

A) Discovers clusters from unlabelled data
B) Learns from labelled input-output pairs
C) Maximises entropy
D) Uses no objective function

3. LLMs are best described as:

A) A completely separate paradigm from deep learning
B) A form of rule-based AI
C) A scaled specialisation of deep learning
D) Purely symbolic systems

4. Representation learning refers to:

A) Manual feature engineering
B) Automatic feature extraction by the model
C) Data labelling pipelines
D) Reinforcement reward shaping

🤔 Reflection Prompts

13. *Why might deeper networks require more training data to perform well?*
14. *What trade-offs exist between model interpretability and raw performance?*
15. *Why are LLMs considered a natural evolutionary step from earlier deep learning models?*
16. *If compute were unlimited, what limitations would still remain in ML systems?*

MODULE 3 | From Sequence Models to Transformers

3.1 The Challenge of Sequence Modelling

Many real-world problems involve sequences: natural language (word by word), source code (token by token), genomic data (base by base), and time-series data (step by step). The core challenge is that the meaning of an element often depends on earlier elements.

Example: the word bank carries different meaning in different contexts.

- "The bank approved the loan." — financial institution
- "He sat by the bank of the river." — geographical feature

> *Sequence models must capture long-range contextual dependencies. This is a central challenge in natural language processing.*

3.2 Early Solution: Recurrent Neural Networks (RNNs)

RNNs process input sequentially, maintaining a hidden state as memory:

$$h_t = f(x_t, h_{t-1})$$

Where x_t = current input, h_{t-1} = previous hidden state, and h_t = updated memory state. This introduces recurrence (a form of memory).

Limitations of RNNs

- Long-range dependencies are difficult to capture — information fades over many steps
- Vanishing and exploding gradients during backpropagation through time
- Sequential processing prevents parallelisation, making training slow

More advanced variants — Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) — partially addressed these issues, but the real breakthrough came with the Transformer architecture.

3.3 The Transformer Architecture

Introduced in 2017 in the seminal paper "Attention Is All You Need" (Vaswani et al.), the Transformer architecture replaced recurrence entirely with an attention mechanism. Key insight: process all tokens in parallel, not sequentially.

> *Transformers are the backbone of virtually every modern large language model, including GPT, Claude, Gemini, LLaMA, and Mistral.*

3.4 The Attention Mechanism

Instead of compressing the entire history into a single hidden state, the attention mechanism:

17. Looks at all tokens simultaneously
18. Computes pairwise importance (relevance) scores between tokens
19. Produces a weighted combination of values based on those scores

Example: in the sentence "The cat sat on the mat because it was tired," the pronoun "it" attends strongly to "cat," not "mat." The model learns these relevance relationships from data.

3.5 Self-Attention (Mathematical Formulation)

The scaled dot-product attention formula is:

$$Attention(Q, K, V) = softmax(QK^T / \sqrt{d_k}) \cdot V$$

Where:

- Q = Query matrix (what we are looking for)
- K = Key matrix (what each token offers)
- V = Value matrix (the information to aggregate)
- d_k = dimension of key vectors (scaling factor to stabilise gradients)

Intuition:

20. Compute similarity between the query and all keys
21. Normalise scores using softmax to obtain attention weights
22. Compute a weighted sum of the value vectors

> *This mechanism allows each token to dynamically attend to any other token in the sequence, enabling the model to capture arbitrary long-range dependencies.*

3.6 Why Transformers Changed Everything

Transformers revolutionised NLP and AI broadly because they:

- Capture long-range dependencies without degradation
- Process entire sequences in parallel, enabling massive training efficiency
- Scale extremely well with increased data and parameters
- Transfer effectively to a wide range of downstream tasks

3.7 Tokens and Tokenisation

LLMs do not read characters or words directly. They operate on tokens — discrete units produced by a subword tokeniser (e.g., Byte Pair Encoding). Tokens can be whole words, common subwords, or individual characters.

Example: "unbelievable" → ["un", "believ", "able"]

Why subword tokenisation matters:

- Handles rare and out-of-vocabulary words gracefully
- Reduces vocabulary size while maintaining expressiveness
- Enables cross-lingual and multilingual modelling

3.8 Context Window

The context window defines the maximum number of tokens a model can process in a single forward pass. If a model has a 128,000-token context window (as in Claude 3.5 Sonnet), it cannot directly process inputs exceeding that limit.

This limitation motivates several engineering solutions:

- Retrieval-Augmented Generation (RAG) — retrieve relevant chunks rather than processing entire corpora
- Conversation summarisation — compress older context
- Chunking strategies — split long documents into manageable segments

---

3.9 Scaling Laws

Empirical research (Kaplan et al., 2020; Hoffmann et al., 2022 — the Chinchilla scaling laws) demonstrated that model performance improves predictably with increased parameters, training data, and compute. This relationship follows approximate power laws, suggesting that scaling is systematic rather than random. This insight drove the development of today's frontier models.

3.10 Mini Case Study: Why GPT-3 Was a Breakthrough

Before GPT-3 (2020), models required task-specific fine-tuning. GPT-3 demonstrated:

- Few-shot learning: performing tasks from a handful of examples in the prompt
- In-context learning: adapting behaviour through examples rather than weight updates
- Emergent abilities: capabilities not explicitly trained for

However, scale also introduced: higher operational cost, increased latency, and heightened hallucination risks — trade-offs that motivate the engineering decisions covered in later modules.

Feature	RNN	Transformer
Processing mode	Sequential	Parallel
Long-range dependencies	Weak	Strong

Feature	RNN	Transformer
Training speed	Slow	Fast
Scalability	Limited	Excellent
Dominant today?	No	Yes

Stage	Method	Key Limitation
Early NLP	N-grams / rule-based	No deep context
Mid-2010s	Recurrent Neural Networks (RNN)	Vanishing gradients; sequential bottleneck
Late 2010s	LSTM / GRU	Still sequential; limited scalability
2017–present	Transformer / Self-Attention	Compute-intensive; large memory footprint

📝 Multiple Choice Questions

1. What core problem do Transformers solve compared to RNNs?

A) Overfitting on small datasets
B) Sequential bottlenecks and vanishing gradients
C) Data labelling requirements
D) Feature engineering effort

2. Self-attention computes:

A) Token frequency distributions
B) Gradient descent update steps
C) Weighted combinations of values based on query-key similarity
D) Activation map gradients

3. The context window refers to:

A) The total model parameter count
B) The size of the training dataset
C) The maximum token input length per forward pass
D) The embedding dimension

4. Scaling laws suggest that model performance improves with:

A) Less compute and smaller models
B) Manual feature engineering
C) Increased parameters, data, and compute
D) Lower-dimensional embeddings

🤔 Reflection Prompts

23. *Why is parallel processing crucial for training large-scale models efficiently?*
24. *What trade-offs arise when expanding context window size?*
25. *Why did Transformers displace RNNs despite being more compute-intensive?*
26. *If compute were unlimited, what constraints would still limit model capability?*

MODULE 4 | Pre-training, Transfer Learning & Fine-Tuning

4.1 From Task-Specific Models to Foundation Models

In traditional machine learning, each model was trained from scratch for a single specific task: one model for spam detection, another for image recognition, and so on. Modern AI takes a fundamentally different approach:

> *Train once at massive scale on general data, then adapt for many tasks. This is the foundation model paradigm.*

4.2 What is Pre-training?

Pre-training involves training a large model on vast amounts of general data using a broad objective. For language models, the standard objective is next-token prediction:

$$P(x_t \mid x_1, x_2, \ldots, x_{t-1})$$

The model learns to predict the most probable next token given all preceding tokens. This is optimised using cross-entropy loss:

$$\ell = -\sum_t \log P(x_t \mid x_{<t})$$

Through this process, the model implicitly learns language structure, world knowledge, reasoning patterns, code syntax, and mathematical conventions.

> *Pre-training builds a general-purpose language representation engine, not a task-specific tool.*

Representative Pre-trained Models (as of 2025–2026)

Proprietary models: GPT-4o (OpenAI), Claude 3.5 Sonnet / Claude 3 Opus (Anthropic), Gemini 1.5 Pro (Google DeepMind)

Open-weight models: LLaMA 3.1 (Meta), Mistral Large, Falcon 2, Qwen 2

> *Key distinction: Open-weight models allow inspection, modification, and local deployment of model parameters. Proprietary models are accessed exclusively through APIs.*

4.3 Transfer Learning

Transfer learning is the principle that knowledge acquired in one domain can improve performance in another. In neural networks:

- Early layers learn general, reusable features (e.g., syntax, grammar)
- Later layers specialise for task-specific patterns

In LLMs: pre-training produces general language understanding; fine-tuning adapts it to specific tasks. Transfer learning dramatically reduces data requirements, training cost, and engineering effort.

4.4 Fine-Tuning Methods

Full Fine-Tuning

Update all model parameters on task-specific data. Requires substantial compute and carries risk of catastrophic forgetting (overwriting general capabilities).

Instruction Tuning

Train the model on curated instruction-response pairs. Example:

> Input: "Explain retrieval-augmented generation simply."Output: [Structured, helpful explanation]

This significantly improves the model's helpfulness and instruction-following behaviour.

Reinforcement Learning from Human Feedback (RLHF)

The RLHF pipeline aligns model outputs with human preferences:

27. The model generates multiple candidate outputs
28. Human annotators rank the outputs by quality
29. A reward model learns to predict human preference scores
30. The policy (LLM) is updated via reinforcement learning to maximise reward

$$\pi^* = argmax\_\pi \, ^{\circ}C[R\_\varphi(\pi)]$$

RLHF is central to how ChatGPT, Claude, and other aligned models are trained. Direct Preference Optimisation (DPO) is a more recent, computationally efficient alternative.

Parameter-Efficient Fine-Tuning (PEFT)

Instead of updating all parameters, PEFT methods insert small trainable modules and freeze the bulk of the model. LoRA (Low-Rank Adaptation) decomposes weight updates into low-rank matrices, drastically reducing trainable parameters. Benefits: lower memory footprint, faster training, reduced cost.

4.5 When to Fine-Tune vs When to Use RAG

Fine-tuning is appropriate when:

- You need consistent domain-specific language or tone
- You require stable structured output formats
- You are performing domain adaptation with sufficient training data

Fine-tuning is less appropriate when:

- You only need knowledge grounding (information, not behaviour change)
- The information changes frequently
- Labelled data is limited
- You need real-time updates

Feature	Fine-Tuning	RAG
Modifies Model Weights	Yes	No
External Knowledge	Embedded in weights	Retrieved dynamically
Adaptability to Change	Slower — requires retraining	Fast — update documents
Compute Cost	High	Low to moderate
Best For	Behaviour and style	Knowledge grounding
Hallucination Risk	Moderate	Lower (with quality docs)

> *Core Distinction: Fine-tuning changes how the model behaves. RAG changes what information the model can access.*

4.6 Mini Case Study: Legal Document Assistant

A law firm requires an AI assistant for case research.

Option 1 — Fine-tuning on legal documents:

Expensive. Hard to update as laws change. Risk of outdated embedded knowledge.

Option 2 — RAG over a maintained legal database:

Index the firm's legal database. Retrieve relevant case law sections. Provide grounded, citable answers with low hallucination risk.

> *Conclusion: RAG is superior when knowledge evolves frequently or must be kept current and auditable.*

4.7 Limitations of Fine-Tuning

- Catastrophic forgetting of general capabilities
- High compute and data labelling cost
- Regulatory concerns around data used in training
- Deployment complexity for version management
- Misalignment risk if training data contains errors

4.8 Module 4 Summary

In this module, you learned:

- Pre-training builds general-purpose representations through next-token prediction
- Transfer learning makes pre-training reusable across tasks
- Multiple fine-tuning strategies exist (full, instruction, RLHF, PEFT)

- RLHF aligns model behaviour with human preferences
- RAG and fine-tuning serve fundamentally different purposes

Strategy	Compute Cost	Flexibility	Primary Use Case
Train from Scratch	Very High	Low	Specialised research models
Full Fine-Tuning	High	Moderate	Domain behaviour adaptation
PEFT / LoRA	Low to Moderate	Moderate	Efficient task adaptation
RAG	Low to Moderate	High	Dynamic knowledge grounding
Prompt Engineering	Very Low	Moderate	Behaviour and format shaping

📝 Multiple Choice Questions

1. Pre-training primarily optimises:

A) Sentiment classification accuracy
B) Next-token prediction
C) Image segmentation
D) Policy reward

2. Fine-tuning modifies:

A) Input tokens
B) The vector database
C) Model weights
D) The retrieval pipeline

3. RAG is most useful when:

A) Knowledge must be dynamic and current
B) Compute resources are unlimited
C) The model is very small
D) No documents exist

4. Transfer learning primarily reduces:

A) Context window size
B) Training cost and data requirements
C) Vocabulary size
D) Token count per inference

🤔 Reflection Prompts

31. *Why might fine-tuning introduce regulatory or compliance risks in sensitive industries?*
32. *What trade-offs guide the choice between RAG and fine-tuning in your domain?*

33. *Why is pre-training considered the foundational step in modern LLM development?*
34. *When building an AI assistant for your use case, under what conditions would you invest in fine-tuning?*

MODULE 5 | Retrieval-Augmented Generation (RAG)

5.1 Why RAG Exists

Large Language Models have fundamental limitations that make them unreliable for knowledge-intensive tasks:

35. Knowledge Cutoff — The model only knows what was in its training data up to a certain date.
36. Hallucination — The model may generate plausible but factually incorrect information.
37. No Real-Time Updates — The model cannot access live data without external integration.
38. Limited Context Window — The model cannot fully ingest large document corpora.

> *RAG Solution: Combine the generation capability of an LLM with dynamic external knowledge retrieval to ground responses in accurate, up-to-date information.*

5.2 Core Idea of RAG

Instead of relying solely on internal model weights for knowledge:

39. Retrieve relevant documents or passages from an external knowledge base
40. Insert them into the prompt as context
41. Generate an answer grounded in the retrieved information

This makes the system more accurate, more controllable, and more easily updated — simply by refreshing the document store.

5.3 Embeddings: Representing Meaning as Vectors

The retrieval step in RAG relies on semantic similarity, which requires converting text into dense numerical vectors called embeddings:

$$text \rightarrow v \in \mathbb{R}^d$$

Where d is the embedding dimension (typically 768 to 3072 dimensions depending on the model). Semantically similar texts produce vectors that are geometrically close. For example:

- "Fine-tuning adapts pre-trained models for specific tasks."
- "Model adaptation through targeted training on task data."

These two sentences produce similar embedding vectors even though they share few exact words.

5.4 Similarity Search

To retrieve relevant documents, we measure the geometric similarity between the query embedding and all stored document embeddings. The most common metric is cosine similarity:

$$cos(\theta) = (A \cdot B) / (|A| \cdot |B|)$$

Where A = query embedding and B = document embedding. A cosine similarity of 1.0 indicates identical direction (semantically equivalent); a value near 0 indicates no semantic relationship.

5.5 Vector Databases

Vector databases are purpose-built to store embeddings and perform approximate nearest-neighbour (ANN) search at scale. Leading vector databases include Pinecone, Weaviate, FAISS (open source), Chroma, and Qdrant.

Core function: given a query vector, return the top-k most semantically similar document chunks.

5.6 Chunking Strategies

Large documents must be split into smaller, semantically coherent chunks before embedding and storage. The choice of chunking strategy significantly affects retrieval quality.

Common strategies:

- Fixed token size (e.g., 500 tokens per chunk) — simple but may split concepts
- Sliding window with overlap (e.g., 500 tokens, 50-token overlap) — better continuity
- Semantic chunking — splits at natural section boundaries using structure detection

Characteristic	Larger Chunks	Smaller Chunks
Context richness	More surrounding context	Less surrounding context
Retrieval precision	Lower — includes noise	Higher — more focused
Fragmentation risk	Low	Higher for complex concepts

5.7 The Full RAG Pipeline

A complete RAG system involves two distinct phases:

Indexing Phase (Offline)

42. Ingest source documents
43. Split into chunks using a chosen strategy
44. Generate embeddings for each chunk
45. Store embeddings and metadata in a vector database

Inference Phase (Online)

46. Receive user query
47. Generate query embedding
48. Perform similarity search to retrieve top-k relevant chunks
49. Inject retrieved chunks into the prompt as context
50. Generate a grounded response

5.8 Types of RAG Systems

Naïve RAG

Simple retrieval followed by prompt injection. Fast to implement but may retrieve irrelevant chunks for complex queries.

Hybrid RAG

Combines dense vector search with sparse keyword search (BM25). The combination improves recall for both semantic and lexical queries.

Agentic RAG

The retrieval process is managed by an agent that decides whether to retrieve, what queries to issue, and how many retrieval steps to perform. This is the most powerful but also most complex variant.

5.9 Mini Case Study: AI Course Assistant

Problem: Students ask questions from a structured course pack. Without RAG, the model may hallucinate or answer beyond the scope of the course material. With RAG:

51. Course material is indexed into a vector store
52. Student queries are embedded and matched to relevant lesson chunks
53. Retrieved context is injected into the prompt
54. The model answers strictly from retrieved text, with citations

> *Result: Higher factual reliability, controlled knowledge boundaries, and improved learning outcomes compared to a standalone LLM.*

5.10 Limitations of RAG

RAG is a powerful technique but not a complete solution:

- Poor chunking leads to fragmented or irrelevant retrieval
- Embedding model mismatch may cause semantic misalignment
- Retrieval noise from low-quality source documents

- Long contexts increase inference cost and may exceed context window limits
- Security risk: retrieved content from untrusted sources may contain adversarial text (prompt injection)

> *A RAG system is only as good as its documents, its chunking strategy, and its prompt design.*

5.11 Why RAG Matters for Agentic AI

In agentic systems, RAG functions as a knowledge tool that the agent can invoke. The agent decides when retrieval is needed, what query to issue, and how to integrate the results into multi-step reasoning. This combination — agent reasoning + dynamic knowledge retrieval — is foundational to enterprise AI systems.

5.12 Module 5 Summary

In this module, you learned:

- Why LLMs hallucinate and how RAG addresses this
- How embeddings represent semantic meaning as vectors
- How cosine similarity enables semantic search
- The full RAG pipeline — indexing and inference phases
- Three types of RAG: naïve, hybrid, and agentic
- When RAG outperforms fine-tuning

Dimension	RAG	Fine-Tuning
Knowledge Update	Instant — update documents	Requires model retraining
Behavioural Change	No	Yes
Hallucination Reduction	Strong with quality docs	Limited
Compute Cost	Lower	High
Explainability	Higher — citable sources	Lower — embedded weights

---

📝 Multiple Choice Questions

1. What problem does RAG primarily address?

A) Model training cost
B) Knowledge grounding and hallucination reduction
C) Tokenisation complexity
D) Activation function design

2. Cosine similarity measures:

A) Token count difference

B) Gradient magnitude
C) Directional similarity between vectors
D) Loss convergence rate

3. Fine-tuning differs from RAG because it:

A) Only updates retrieved documents
B) Modifies model weights
C) Changes embedding dimensions only
D) Expands the context window

4. Poor chunking may result in:

A) Better embedding quality
B) Irrelevant or fragmented retrieval results
C) Smaller embedding vectors
D) Faster model training

🤔 Reflection Prompts

55. *Why does RAG reduce hallucination compared to a standalone LLM?*
56. *What risks arise from poorly curated or outdated documents in a RAG system?*
57. *When would hybrid RAG outperform naïve RAG?*
58. *If context windows were unlimited, would RAG still provide value? Why or why not?*

MODULE 6 | What Is an AI Agent?

6.1 What Is an AI Agent?

An AI Agent is a system that perceives information, reasons about it, and takes purposeful actions to achieve a goal within an environment. Unlike a standalone LLM that only generates text, an agent:

- Takes actions in the world
- Uses tools (APIs, databases, calculators, web search)
- Maintains and updates memory across steps
- Operates in reasoning loops rather than single forward passes
- Interacts with external systems to produce real-world effects

Formally, an agent can be described by the tuple:

> *Agent = (Perception, Reasoning, Action, Memory)*

The agent-environment loop: Environment → Observation → Reasoning → Action → Environment update. This loop continues until the goal is satisfied.

6.2 LLM vs AI Agent — A Critical Distinction

> *Common misconception: 'LLM' and 'Agent' are often used interchangeably. They are not the same thing.*

Feature	LLM (Standalone)	AI Agent
Generates Text	Yes	Yes
Uses External Tools	No	Yes
Takes Real-World Actions	No	Yes
Maintains Structured Memory	Limited	Yes (short- and long-term)
Goal-Oriented Operation	Implicit	Explicit
Multi-step Reasoning	Single pass only	Iterative loops

> *Core principle: An LLM is a reasoning engine. An AI Agent is a decision-making system built around an LLM.*

6.3 Components of an AI Agent

LLM Core

The central reasoning engine. Processes observations and produces reasoning outputs and action decisions.

Memory

Short-term memory is stored within the context window (conversation history, immediate observations). Long-term memory is stored externally — in vector databases, relational databases, or structured state stores — and retrieved when needed.

Tools

External capabilities the agent can invoke: web search, calculator, database queries, code execution, API calls, file read/write operations.

Planner

Determines what steps to take, in what order, and when to stop. In simple agents, this is implicit in the prompt. In complex agents, this is a dedicated reasoning step.

Environment

The system or world in which the agent's actions produce observable effects — a file system, a database, a web interface, or a user conversation.

6.4 Tool Invocation

The tool-invocation cycle is what transforms a passive LLM into an active agent system:

59. The agent decides external information or action is needed
60. It selects the appropriate tool
61. It formats the input parameters for the tool
62. The tool executes and returns structured output
63. The agent incorporates the result into its ongoing reasoning

Example: User asks, 'What is the current stock price of Tesla?' A standalone LLM cannot answer accurately. An agent invokes a financial data API, retrieves the current price, and responds with grounded information.

6.5 Reactive vs Planning Agents

Reactive Agent

Responds directly to the current input with minimal long-term strategy. Suitable for: customer support chatbots, FAQ assistants, simple QA with retrieval.

Planning Agent

Decomposes tasks into sub-steps, executes them sequentially, and reflects on outcomes. Example: a research assistant agent tasked with writing a market analysis report.

Planning agent steps for 'Write a report on AI regulation trends':

64. Search academic and news sources
65. Summarise findings by region and policy domain
66. Organise report structure
67. Draft each section
68. Critique and revise the draft

6.6 Multi-Agent Systems

Multiple specialised agents collaborate to complete complex tasks. Example roles:

- Research agent — gathers and summarises external information
- Summarisation agent — condenses outputs into structured summaries
- Critic agent — evaluates quality and flags errors
- Planner agent — orchestrates the overall workflow

Benefits: specialisation, modularity, improved reasoning quality. Risks: increased coordination complexity, compounding errors, cost escalation.

6.7 Agent Autonomy Levels

Level	Description	Typical Use Case
0	Pure LLM — no tools, no loop	Simple Q&A, summarisation
1	LLM + Retrieval (RAG)	Knowledge-grounded assistants
2	LLM + Tools + Action	Research, data analysis agents
3	Multi-step planning agent	Complex automation, report generation
4	Multi-agent autonomous system	Enterprise workflow orchestration

> *Most current production 'agent' systems operate at Levels 1–3. Full Level 4 autonomy requires robust safety guardrails.*

6.8 Failure Modes in Agent Systems

Agent systems introduce new categories of risk:

- Infinite reasoning loops — the agent fails to converge on a solution
- Tool misuse — incorrect tool selection or malformed parameters
- Hallucinated tool calls — the agent invents tool outputs rather than calling tools
- Over-retrieval — retrieving excessive context, degrading focus

- Cost explosion — unbounded API and compute spend
- Latency stacking — each sequential tool call adds response delay

Mitigation: implement max step limits, tool call validation, budget constraints, and confidence thresholds.

6.9 Module 6 Summary

In this module, you learned:

- What distinguishes an AI agent from a standalone LLM
- The five core components of agent architecture
- How tool invocation works
- Reactive vs planning agent design
- Multi-agent systems and their trade-offs
- Agent autonomy levels and associated risks

📝 Multiple Choice Questions

1. What most clearly distinguishes an AI agent from a standalone LLM?

A) The tokenisation method
B) Tool use and action capability in a loop
C) The loss function used in training
D) Context window size

2. A planning agent differs from a reactive agent because it:

A) Uses fewer tools
B) Avoids retrieval entirely
C) Decomposes goals into sequential sub-steps
D) Operates without memory

3. Multi-agent systems increase:

A) System simplicity
B) Determinism of outputs
C) Coordination complexity and potential error propagation
D) Token limits

4. A standalone LLM cannot:

A) Generate text responses
B) Predict the next token
C) Access real-time APIs or take actions
D) Perform inference

🤔 Reflection Prompts

69. *Why are agents more powerful but also more risky than simple LLM applications?*
70. *What safeguards should be standard in any production autonomous system?*
71. *When is a standalone LLM sufficient without full agent architecture?*
72. *What level of autonomy is appropriate for an educational AI system, and why?*

MODULE 7 | Agentic Workflows & Reasoning Patterns

7.1 What Makes a System Agentic?

An agentic system goes beyond a single model inference pass. It:

- Operates in structured reasoning loops
- Uses deliberate reasoning before selecting actions
- Invokes tools based on intermediate conclusions
- Reflects on intermediate outputs and adjusts behaviour
- Pursues goals across multiple sequential steps

The core loop of agentic behaviour can be expressed as:

> *Think → Act → Observe → Update → (repeat)*

7.2 The ReAct Pattern (Reason + Act)

ReAct (Reason + Act) is one of the most influential agentic reasoning patterns. Rather than generating a final answer in a single step, the agent interleaves reasoning steps and tool calls:

ReAct Loop Structure

73. Thought — the model reasons about the current state and what is needed
74. Action — the model selects and calls a tool
75. Observation — the tool returns a result
76. Thought — the model integrates the observation and reasons further
77. Repeat until the goal is achieved

Example — User query: 'What is the GDP of France and how has it changed over the last five years?'

- Thought: I need current GDP data and historical values.
- Action: Call economic data API with France + year range.
- Observation: Receives structured GDP data.
- Thought: Calculate the percentage change across years.
- Action: Compute delta.
- Final Answer: Structured response with cited figures.

> *ReAct dramatically increases accuracy on knowledge-intensive and multi-step tasks compared to single-pass generation.*

7.3 Planning Loops

Planning agents explicitly decompose goals before executing. The formal planning cycle:

> *Goal → Plan → Execute → Evaluate → Refine*

Example: Research Report Agent for 'Renewable energy adoption trends':

78. Search and retrieve academic and industry sources
79. Extract key statistics and trends
80. Identify regional and sectoral patterns
81. Structure report sections and outline
82. Draft each section
83. Critique the draft and refine for accuracy and coherence

7.4 Tool-Use Chains

Agentic systems often chain multiple tools in sequence to complete complex information tasks. Example pipeline:

User Query → Search Tool → Content Extraction Tool → Summarisation Tool → Formatting Tool → Final Output

Each tool in the chain operates on the output of the previous step, creating structured workflow automation.

7.5 Reflection and Self-Correction

Advanced agents evaluate their own intermediate outputs, detect inconsistencies, and revise before producing final responses. The reflection pattern:

> *Initial Output → Critic Step → Revised Output*

Reflection improves output quality by catching factual errors, logical inconsistencies, and incomplete reasoning. However, it increases token consumption, inference latency, and API cost — engineering trade-offs that must be managed in production.

7.6 Memory Integration Strategies

Short-Term Memory

Stored within the active context window. Contains the current conversation history, retrieved context, and recent observations.

Long-Term Memory

Stored externally and retrieved when relevant. Strategies include:

- Conversation summarisation — compress older turns before they exceed context limits

- Vector-based memory retrieval — embed and store past interactions for semantic retrieval
- Structured state tracking — maintain explicit variables (e.g., student level, completed topics)

Example for an educational agent: long-term memory tracks student proficiency level, weak topic areas, and past quiz scores to personalise responses.

7.7 Failure Modes in Agentic Systems

Failure Mode	Description	Mitigation
Infinite loop	Agent cannot converge on a decision	Maximum step limit
Tool misuse	Wrong tool selected or malformed parameters	Tool schema validation
Hallucinated tool calls	Agent fabricates tool outputs	Require actual execution and verify return
Plan drift	Agent deviates from original goal	Re-inject goal at each step
Cost explosion	Unbounded token or API spend	Budget constraints and token limits
Latency stacking	Sequential tool calls accumulate delay	Parallel execution where possible

---

7.8 Cost and Latency Considerations

Agent loops multiply the cost of a single inference:

$$\textit{Total Cost} \propto \textit{Steps} \times \textit{Tokens per Step} \times \textit{Token Price}$$

Engineering decisions around agent design must therefore balance: task quality, number of reasoning steps, token budget per step, and acceptable latency.

7.9 Module 7 Summary

In this module, you learned:

- What makes a system agentic
- The ReAct reasoning pattern and why it improves accuracy
- Planning loops and structured task decomposition
- Tool-use chaining for multi-step automation
- Reflection and self-correction patterns
- Memory strategies for stateful agents
- Failure modes and engineering mitigations

Workflow Pattern	Complexity	Reliability	Approximate Cost
Single-pass LLM	Low	Moderate	Low
RAG-augmented LLM	Moderate	High	Moderate
ReAct Agent	High	Very High	Higher
Multi-agent System	Very High	Variable	Very High

📝 Multiple Choice Questions

1. ReAct stands for:

A) Retrieve and Act
B) Reason and Act
C) Reflect and Compile
D) Reinforce and Act

2. A planning agent differs from a reactive agent because it:

A) Avoids tool invocation
B) Explicitly decomposes goals into sequential sub-steps
C) Uses fewer tokens per response
D) Ignores memory entirely

3. Reflection improves output quality but increases:

A) Accuracy with no trade-offs
B) Compute usage, token consumption, and latency
C) Context window size
D) Embedding dimensionality

4. Agentic systems typically operate through:

A) Single-step responses
B) Looped reasoning, action, and observation cycles
C) Purely deterministic pipelines
D) Rule-based decision frameworks

🤔 Reflection Prompts

84. *When does agentic complexity introduce unnecessary risk without meaningful benefit?*
85. *How would you prevent infinite loops in an autonomous system deployed in production?*
86. *Should an educational AI agent be fully autonomous? What risks would that introduce?*
87. *How do cost constraints fundamentally shape agent architecture decisions?*

MODULE 8 | Model Context Protocol (MCP)

8.1 Why MCP Matters

As AI systems become increasingly agentic, they interact with a growing ecosystem of external tools: databases, APIs, web services, code executors, file systems, and more. Without standardisation, every tool integration is custom-built, fragile, and difficult to secure or audit.

> *Model Context Protocol (MCP) addresses this by providing a standardised framework for structured, secure, and interoperable communication between language models and external tools.*

8.2 The Core Problem MCP Solves

Without a structured protocol:

- Tool call formats are inconsistent across implementations
- Security vulnerabilities arise from unvalidated inputs
- Context injection becomes ambiguous and error-prone
- Systems from different providers cannot interoperate

MCP introduces standard schemas, structured tool definitions, and controlled context exchange — reducing ambiguity and establishing a common interface layer.

8.3 What Is Model Context Protocol?

MCP is a structured interface layer that governs how LLMs request, receive, and process contextual information and tool outputs. Think of it as:

> *LLM ⇆ Structured Middleware (MCP) ⇆ External Tools*

MCP defines: how tools are described to the model, how the model requests tool execution, how responses are structured and returned, and how retrieved context is injected back into the model.

8.4 Core Components of MCP

Tool Definition Schema

Each tool is described with a structured schema that includes its name, description, required inputs, and output format. Example conceptual schema:

{ "tool_name": "get_stock_price", "description": "Retrieve the current market price for a stock ticker", "input_schema": { "ticker": "string (required)", "currency": "string (optional, default: USD)" }, "output_schema": { "price": "number", "currency": "string", "timestamp": "ISO 8601 datetime" }}

This prevents ambiguous or malformed tool calls.

Context Injection Rules

MCP defines what context the model can access, what tool outputs are injected into the reasoning context, which prior actions are included in the model's view, and what information is withheld for security or privacy reasons.

Execution Layer

The execution layer is responsible for validating tool call parameters, executing tool operations in a sandboxed environment, and returning structured, auditable responses.

8.5 Why Standardisation Is Critical

Consider a multi-tool agent interacting with web search, a calculator, a database, and a code executor. Without standardisation, each tool behaves differently, error handling is inconsistent, and adding new tools requires bespoke integration work. With MCP:

- All tools follow a common interface contract
- Agents become modular — tools are interchangeable
- Systems from different vendors can be composed

8.6 Security Implications

Agentic AI systems operating through tools introduce serious security risks:

- Prompt injection — adversarial content in retrieved data manipulates the agent
- Malicious tool calls — the agent is tricked into issuing destructive actions
- Data exfiltration — sensitive data is passed to unauthorised endpoints
- Privilege escalation — the agent accesses resources beyond its authorised scope

MCP mitigates these risks through:

- Strict input/output schema enforcement
- Role-based tool permission systems
- Comprehensive interaction logging and audit trails
- Sandboxed execution environments

> *Security is not an optional add-on in production agentic systems. It must be architected in from the beginning.*

8.7 MCP in Agentic Systems

The agent-MCP-tool interaction cycle:

88. Agent produces a structured tool request based on its reasoning
89. MCP validates the request against the tool schema
90. MCP executes the tool safely in an isolated environment
91. Structured output is returned to the agent's context
92. Agent continues reasoning with the new observation

> *This creates a controlled loop: Reason → Structured Request → Validated Execution → Safe Observation.*

8.8 MCP vs Ad-Hoc Tool Integration

Feature	Ad-Hoc Integration	MCP-Based Integration
Standardisation	Low — custom per tool	High — common schema
Security enforcement	Risky — inconsistent validation	Controlled — schema enforcement
Scalability	Limited — each tool is independent	High — modular and composable
Interoperability	Poor — vendor-locked	Strong — open standard
Maintainability	Complex — bespoke code	Modular — standardised interface

---

8.9 Mini Case Study: Educational Agent with Secure Tool Access

Scenario: Your AI Course Assistant is extended with tool access to a student database.

Without structured protocol: the model could issue arbitrary database queries, access unauthorised records, or expose sensitive data.

With MCP:

- Tool definitions specify exactly which database fields the model can query
- Input schemas prevent SQL injection and malformed queries
- Execution is sandboxed and logged for audit
- Only authorised fields (e.g., quiz score, module completion) are accessible

> *MCP enables real educational infrastructure — not just a chatbot — while maintaining data protection and compliance.*

8.10 Module 8 Summary

In this module, you learned:

- Why standardised tool integration is critical as AI systems scale

- What MCP is and what problem it solves
- The core components: tool schemas, context injection rules, execution layer
- Why security must be architected into agent systems from the start
- How MCP improves scalability, safety, and interoperability

📝 Multiple Choice Questions

1. MCP primarily standardises:

A) Model training procedures
B) Tokenisation methods
C) Model-tool interaction interfaces
D) Loss function design

2. Structured tool schemas help prevent:

A) Gradient explosion
B) Prompt injection and malformed tool calls
C) Token truncation
D) Overfitting during training

3. MCP improves:

A) Raw inference speed
B) Interoperability between models and tools
C) Embedding vector dimensions
D) Vocabulary coverage

4. Without structured protocol enforcement, agents risk:

A) Reduced compute consumption
B) Improved hallucination resistance
C) Security vulnerabilities and inconsistent behaviour
D) Smaller model footprints

🤔 Reflection Prompts

93. *Why is standardisation critical as AI systems integrate with more external services?*
94. *Should all available tools be accessible to an agent by default?*
95. *How would you design tool permission levels for an educational AI system?*
96. *What are the most serious risks that emerge when LLMs control external systems?*

MODULE 9 | Designing LLM Applications: From Demo to Production

9.1 Why Design Matters

Many LLM applications fail not because the model is incapable, but because of poor system architecture. Common failure causes:

- Prompts are poorly structured or ambiguous
- Context is not managed effectively
- Cost and latency constraints are ignored
- Guardrails and safety filters are absent
- Evaluation is informal or non-existent

> *Building production LLM systems is systems engineering, not just prompting. The model is one component in a larger architecture.*

9.2 Core Components of an LLM Application

A production-grade LLM application is structured in layers:

97. Input Layer — user query, input preprocessing, validation and sanitisation
98. Context Layer — memory retrieval, RAG, system prompt management
99. Reasoning Layer — LLM inference, optional agent loops
100. Tool Layer — APIs, databases, external services (if agentic)
101. Output Layer — formatting, validation, safety filtering, response delivery

Each layer has distinct responsibilities and failure modes that must be engineered against.

9.3 Prompt Design Principles

Prompts define the behaviour and boundaries of the system. Effective prompt design includes:

- Role specification — define who the model is and what it should do
- Output format constraints — specify structure (JSON, markdown, plain text)
- Behavioural boundaries — what topics to cover and which to decline
- Refusal instructions — how to handle out-of-scope or unsafe queries
- Context grounding — instruct the model to answer only from retrieved context

Example system prompt for a course assistant:

> You are an educational assistant for this course. Answer only from the provided course material context. If a topic is not covered in the context, respond: 'This topic is not covered in the current module.' Provide step-by-step explanations when appropriate.

> *Prompt engineering is constraint design — it shapes behaviour, not just output style.*

9.4 System Prompt vs User Prompt vs Retrieved Context

Prompt Component	Purpose	Controls
System Prompt	Defines model behaviour and persona	Role, tone, boundaries, refusals
User Prompt	Specifies the immediate task or question	Task intent
Retrieved Context (RAG)	Provides factual grounding	Knowledge scope and accuracy

Clear separation of these three components improves reliability, control, and safety.

9.5 Context Management

The context window is a finite, costly resource. Key management strategies:

- Summarisation — compress older conversation turns before they exceed window limits
- Selective retrieval — retrieve only the most relevant chunks rather than injecting entire documents
- Truncation policies — define clear rules for what to drop when context is full
- Dynamic context selection — adaptively choose what context to include based on the current query

Trade-off principle:

> *More Context ⟹ Better Reasoning, Higher Cost, Higher Latency*

9.6 Guardrails and Safety

LLMs can hallucinate, produce unsafe content, and deviate from system intent. Production systems require multiple layers of guardrail:

- Refusal instructions in system prompts
- Input validation — sanitise and check user queries before processing
- Output validation — verify response format and content before delivery
- Content moderation filters — detect and block policy-violating outputs
- Step limits — cap the number of agent reasoning cycles

> *Design principle: Assume that every component can fail. Build defensive systems that degrade gracefully.*

9.7 Deterministic vs Stochastic Decoding

LLMs generate tokens probabilistically. The temperature parameter controls the trade-off between creativity and consistency:

- Temperature = 0: Greedy decoding — always select the highest-probability token. Fully deterministic, least creative.
- Temperature = 0.7: Balanced — moderate randomness. Commonly used for general assistants.
- Temperature = 1.0+: High creativity — more diverse but less predictable outputs.

Additional decoding strategies include Top-K sampling (restrict selection to K most probable tokens) and Top-P / nucleus sampling (restrict to tokens comprising the top P% of probability mass).

> *Educational and knowledge-grounded systems typically use lower temperatures (0.0–0.3) to prioritise factual consistency over creative variation.*

9.8 Cost and Latency Considerations

Inference cost is a function of token volume:

$$Cost \propto (Input\ Tokens + Output\ Tokens) \times Token\ Price$$

Agentic systems multiply this by the number of reasoning steps. Latency increases with each tool call, long context, and additional reasoning loops. Production design must explicitly balance accuracy, speed, and cost through architecture choices — not only model selection.

9.9 Demo vs Production Systems

Dimension	Demo System	Production System
Prompt design	Simple / ad hoc	Structured and versioned
Retrieval	Basic keyword or semantic	Optimised hybrid pipeline
Guardrails	Minimal or absent	Strict and multi-layered
Logging	None	Comprehensive, queryable
Evaluation	Informal	Formal metrics and regression tests
Monitoring	None	Continuous dashboards
Cost management	Ignored	Budgeted and tracked

> *Demo systems demonstrate possibility. Production systems ensure reliability, safety, and scalability under real-world load.*

9.10 Observability and Monitoring

A production AI system must log and monitor:

- All inputs and outputs — for debugging, auditing, and improvement
- Tool calls and their results — to trace agent behaviour
- Token usage per session — for cost management
- Error rates and failure modes — to identify systemic issues
- Latency distribution — to detect degradation

> *Observability transforms experimentation into engineering. Without it, you are operating blind.*

9.11 LLM Application Architecture Stack

Layer	Purpose	Key Technologies
Foundation Model	Core reasoning engine	GPT-4o, Claude 3.5, LLaMA 3
Prompt Layer	Behaviour definition	System prompt, few-shot examples
Retrieval Layer	Knowledge grounding	Vector DB, chunking, embeddings
Agent Layer	Multi-step reasoning	ReAct, planning loops
Tool Layer	External action	APIs, code execution, databases
Governance Layer	Safety and compliance	MCP, guardrails, audit logging

> *As systems scale, the governance layer becomes as important as the intelligence layer.*

9.12 Module 9 Summary

In this module, you learned:

- LLM applications require structured, layered architecture
- Prompt design is constraint engineering, not just instruction writing
- Context management is critical for cost and quality
- Guardrails are mandatory in production systems
- Temperature and decoding strategies shape reliability vs creativity
- Monitoring and observability differentiate demos from production systems

---

📝 Multiple Choice Questions

1. Guardrails in LLM systems are designed to:

A) Increase raw model parameter count

B) Prevent unsafe, incorrect, or out-of-scope outputs

C) Reduce token length
D) Improve embedding quality

2. Higher temperature settings generally:

A) Increase output determinism
B) Increase output diversity and creativity
C) Increase context window size
D) Reduce loss function values

3. Production systems differ from demos primarily because they:

A) Use smaller, simpler models
B) Eliminate prompt engineering
C) Add monitoring, reliability, and governance layers
D) Avoid retrieval entirely

4. Context management is necessary because:

A) Models forget everything between sessions
B) Context windows are finite and costly
C) Embeddings are too small for long text
D) Loss functions become unstable with long inputs

🤔 Reflection Prompts

102. *Why do many LLM demos succeed in testing but fail in production?*
103. *For an educational AI, should the system prioritise creativity or reliability?*
104. *How would you design cost controls for a multi-step agentic system?*
105. *What five metrics would you track in a production AI system, and why?*

MODULE 10 | Evaluation & Benchmarks

10.1 Why Evaluation Is Complex in LLM Systems

Traditional ML evaluation is relatively straightforward: classification accuracy, regression error, or detection precision/recall. LLM evaluation is fundamentally different because LLMs:

- Generate open-ended text with multiple valid correct answers
- Operate probabilistically — same input can yield different outputs
- May reason in multi-step chains
- Interact with tools and environments in agentic settings
- Must be assessed for safety, alignment, and fairness, not just capability

> *LLM evaluation is inherently multidimensional. No single metric captures system quality.*

10.2 Types of Evaluation

Capability Evaluation

Measures task performance on standardised benchmarks: question answering, coding, mathematical reasoning, factual recall.

Behavioural Evaluation

Measures: helpfulness, safety, alignment with guidelines, appropriate refusal behaviour.

System-Level Evaluation

Measures operational characteristics: latency distribution, cost per query, failure rate, stability under load.

Agent Evaluation

Measures agentic performance: task completion rate, tool selection accuracy, plan quality, loop efficiency.

10.3 Major Benchmark Datasets

MMLU (Massive Multitask Language Understanding)

Covers 57 academic subjects including mathematics, law, medicine, and ethics. Multiple-choice format. Widely used as a general reasoning benchmark.

HELM (Holistic Evaluation of Language Models)

Comprehensive framework evaluating models across multiple dimensions: accuracy, fairness, robustness, efficiency, and toxicity.

HumanEval

Measures functional code generation ability by evaluating whether generated code passes unit tests. Standard benchmark for coding capability.

GSM8K

Multi-step grade school mathematics reasoning benchmark. Tests mathematical problem-solving and chain-of-thought reasoning.

GPQA (Graduate-Level Google-Proof Q&A)

Evaluates reasoning on expert-level questions across biology, chemistry, and physics that cannot be easily looked up. Increasingly used to assess frontier model capability.

> *Important caveat: Benchmarks measure capability in controlled settings — not real-world reliability. A high MMLU score does not guarantee production performance.*

10.4 Offline vs Online Evaluation

Offline Evaluation

Uses predefined test datasets with static, repeatable evaluation. Advantages: controlled, comparable across models, reproducible. Disadvantages: may not reflect real user behaviour or distribution.

Online Evaluation

Evaluation through live user interactions: A/B testing different system versions, collecting explicit feedback (thumbs up/down), implicit signals (session length, query reformulation). Advantages: real-world grounding. Disadvantages: harder to control, raises ethical considerations.

10.5 Human-in-the-Loop Evaluation

Many production systems rely on systematic human review, particularly for:

- Ranking multiple system outputs on quality
- Detecting hallucinations and factual errors
- Assessing safety and alignment with guidelines

Human evaluation is central to RLHF training pipelines and ongoing safety audits. It remains the gold standard for evaluating open-ended generation quality, despite being expensive and slow.

10.6 Evaluating Hallucination

Hallucination occurs when a model generates factually incorrect information that appears plausible and confident. Detection methods include:

- Cross-checking generated claims against retrieved RAG sources

- Consistency checks — regenerating answers and comparing for self-contradiction
- Fact verification tools — automated pipelines that validate claims against knowledge bases
- Confidence calibration — measuring how well model confidence predicts factual accuracy

Hallucination rate is a critical metric in RAG systems and knowledge-intensive applications.

10.7 Evaluating Agent Systems

Metric	Definition	Why It Matters
Task Success Rate	Percentage of goals fully completed	Primary measure of agent effectiveness
Tool Selection Accuracy	Correct tool chosen for each action	Measures agent planning quality
Step Efficiency	Average reasoning steps per completed task	Balances thoroughness against cost
Cost per Task	Total token and API cost per completed goal	Production sustainability
Failure Mode Rate	Frequency of loops, errors, and incorrect tool calls	Safety and reliability indicator

Agent systems must be evaluated beyond text quality. Task completion, tool accuracy, and cost efficiency are equally important.

10.8 Reliability vs Raw Capability

A model may achieve high scores on MMLU while still hallucinating in production applications. Benchmark performance and production reliability are distinct concerns:

- Benchmarks evaluate capability in controlled, single-turn settings
- Production reliability requires consistency across diverse real-world inputs
- Stability under adversarial or edge-case inputs is often not captured by benchmarks

Production evaluation must focus on consistency, robustness, and failure rate management — not just headline benchmark scores.

10.9 Mini Case Study: Evaluating an AI Course Assistant

Metrics for a production educational AI system:

106. Answer accuracy — percentage of answers matching authoritative course material
107. Hallucination rate — frequency of claims not grounded in retrieved context
108. Quiz generation correctness — validity of generated assessment questions

109. Student satisfaction score — learner feedback ratings
110. Cost per student session — total token cost per learning interaction
111. Average response latency — time from query to delivered response

This shifts the evaluation question from 'How smart is the model?' to 'How reliable and educationally effective is the system?'

10.10 Continuous Evaluation Pipeline

Evaluation in production is not a one-time event. Modern systems implement continuous evaluation:

112. Log all interactions with metadata
113. Tag errors and failure modes for root cause analysis
114. Collect user feedback signals
115. Run automated regression tests on prompt changes
116. Iterate prompts, chunking, and retrieval parameters based on evidence

> *Continuous evaluation transforms a static system into an improving one. It is the engineering discipline that closes the loop between deployment and development.*

10.11 The Engineering Trade-Off Triangle

LLM system design involves balancing three competing dimensions:

$$\textit{Capability} \Leftrightarrow \textit{Cost} \Leftrightarrow \textit{Reliability}$$

Optimising for any one dimension typically requires trade-offs with the others. System architects must make explicit, deliberate choices about where on this triangle the product should sit, based on user needs and resource constraints.

10.12 Module 10 Summary

In this module, you learned:

- Why LLM evaluation is inherently multidimensional
- Major benchmarks: MMLU, HELM, HumanEval, GSM8K, GPQA
- Offline vs online evaluation trade-offs
- Human-in-the-loop evaluation and its role in RLHF
- How to evaluate hallucination in RAG systems
- Agent-specific evaluation metrics
- The critical distinction between capability and production reliability
- Continuous evaluation as an engineering discipline

Evaluation Type	Focus	Best Applied For
Benchmark Testing	Model capability	Research comparison, model selection
Human Review	Quality and safety	Alignment auditing, RLHF
System Metrics	Cost and latency	Production operations
Agent Metrics	Task completion and efficiency	Autonomous system monitoring
Continuous Pipeline	Improvement over time	Production optimisation

📝 Multiple Choice Questions

1. MMLU primarily evaluates:

A) API cost efficiency
B) Model capability across diverse academic subjects
C) Tool invocation accuracy
D) Context window capacity

2. Online evaluation differs from offline evaluation because it:

A) Uses no human users
B) Operates on static test sets
C) Reflects real-world user interactions
D) Avoids feedback loops

3. Agent evaluation specifically requires measuring:

A) Embedding vector dimension
B) Tokenisation quality
C) Task success rate and tool selection accuracy
D) Activation function gradients

4. A high benchmark score does not guarantee:

A) Token generation capability
B) Production reliability and robustness
C) Model scaling capacity
D) Embedding similarity computation

🤔 Reflection Prompts

117. *Should an educational AI system prioritise benchmark scores or production reliability? Why?*
118. *How would you design a hallucination detection system for your AI Course Assistant?*
119. *Why is human feedback still essential even for highly capable models?*

120. *Which evaluation metrics matter most in agentic systems, and why?*

Glossary of Key Terms

Foundations of LLM Applications & Agentic AI Systems

Companion Reference — Modules 1–10

A

Activation Function

A mathematical function applied to a neuron's output to introduce non-linearity, enabling neural networks to learn complex, non-linear mappings.

$$a = \sigma(Wx + b)$$

Common examples: ReLU (max(0, x)), sigmoid, tanh, and GELU (used in Transformers).

See also: Neural Network, Deep Learning, Transformer

Agent

A system that perceives input, reasons about it, and takes actions to achieve a goal within an environment. Unlike a standalone LLM, an agent can use tools, maintain memory, and operate in multi-step loops.

$$Agent = (\ Perception,\ \ Reasoning,\ \ Action,\ \ Memory\ )$$

See also: Agentic AI, Planning Agent, Reactive Agent, LLM

Agentic AI

AI systems that operate in iterative reasoning–action loops and can use tools, memory, and planning to complete multi-step tasks. Distinct from simple LLM inference, which is single-pass.

$$Think \rightarrow Act \rightarrow Observe \rightarrow Update$$

See also: Agent, ReAct Pattern, Multi-Agent System

Alignment

The process of ensuring AI systems behave according to intended human values, objectives, and constraints. Practically implemented through RLHF, Constitutional AI, and instruction tuning.

See also: RLHF, Instruction Tuning, Guardrails

Attention Mechanism

A core component of Transformers that allows each token to dynamically attend to all other tokens in the sequence, computing a weighted combination of their representations based on relevance.

$$Attention(Q, K, V) = softmax(QK^T / \sqrt{d\_k}) \cdot V$$

See also: Transformer, Self-Attention, Query / Key / Value

Approximate Nearest Neighbor (ANN)

A class of search algorithms that efficiently find the most similar vectors in high-dimensional space without exhaustive comparison. Used in vector databases for fast RAG retrieval.

See also: Vector Database, RAG, Cosine Similarity

B

Backpropagation

An algorithm that computes gradients of a loss function with respect to model parameters, enabling weight updates via gradient descent. The backbone of neural network training.

See also: Gradient Descent, Loss Function, Neural Network

Benchmark

A standardized dataset or evaluation task used to measure model performance. Examples: MMLU (general knowledge), HumanEval (coding), GSM8K (math reasoning), GPQA (expert science), and LiveCodeBench (real-world coding).

See also: MMLU, HELM, HumanEval, GSM8K, GPQA, Evaluation

Bias

(1) Statistical: Systematic distortion in model predictions caused by skewed training data or flawed modeling assumptions. (2) Architectural: A learnable parameter added to the linear transformation in a neural network layer (the b in Wx + b).

See also: Overfitting, Parameter, Weight

BM25

A classical keyword-based ranking function used in information retrieval. Often combined with vector search in hybrid RAG systems to improve recall.

See also: Hybrid RAG, Vector Similarity Search, RAG

C

Chinchilla Scaling Laws

Empirical findings by Hoffmann et al. (2022) showing that optimal LLM performance requires balancing model size and training data proportionally — not just maximizing parameters. Smaller, data-rich models (like Chinchilla) can outperform larger, data-poor models.

See also: Scaling Laws, Pre-training, Foundation Model

Chunking

Splitting large documents into smaller, retrievable segments for use in RAG systems. Common strategies include fixed-size chunking (e.g., 512 tokens), sliding window with overlap, and semantic chunking based on meaning boundaries.

See also: RAG, Context Window, Vector Database

Context Window

The maximum number of tokens a model can process in a single inference call. Modern models range from 8K to over 1M tokens. Exceeding the context window requires chunking, summarization, or retrieval strategies.

See also: Token, RAG, Memory (Short-Term)

Cosine Similarity

A metric measuring the angle between two vectors, used to quantify semantic similarity in embedding space. A value of 1 indicates identical direction; 0 indicates orthogonality.

$$\cos(\theta) = (A \cdot B) / (|A| \cdot |B|)$$

See also: Embedding, Vector Similarity Search, RAG

Cross-Entropy Loss

A loss function used in classification and language modeling. Penalizes the model when its predicted probability distribution diverges from the true distribution.

$$L = -\Sigma \; y \cdot \log(\hat{y})$$

See also: Loss Function, Pre-training, Optimization

D

Dataset

A structured collection of data used to train, validate, or evaluate machine learning models. Quality, diversity, and scale of datasets are critical determinants of model capability.

See also: Pre-training, Supervised Learning, Benchmark

Decoding

The process of selecting the next token during text generation. Strategies include greedy (highest probability), top-k sampling, top-p (nucleus) sampling, and beam search. Temperature controls randomness.

See also: Temperature, Stochastic Decoding, Deterministic Decoding, Token

Deep Learning

A subfield of machine learning that uses multi-layer neural networks to automatically learn hierarchical representations from data, eliminating the need for manual feature engineering.

See also: Neural Network, Representation Learning, Machine Learning

Deterministic Decoding

A text generation strategy, such as greedy decoding, that always selects the highest probability next token — producing consistent, reproducible outputs.

See also: Stochastic Decoding, Temperature, Decoding

E

Embedding

A dense numerical vector representation of text (or other data) that captures semantic meaning. Semantically similar inputs produce vectors close together in embedding space.

$$\textit{Text} \rightarrow \boldsymbol{v} \in \mathbb{R}^{\boldsymbol{d}}$$

Embedding dimension d typically ranges from 768 to 3072 in modern models.

See also: Vector Database, RAG, Cosine Similarity, Query Embedding

Environment (Agent Context)

The external system in which an agent operates, takes actions, and receives observations. The environment provides feedback that informs the agent's next reasoning step.

See also: Agent, Reinforcement Learning, Agentic AI

Evaluation

The process of measuring model or system performance. In LLM systems, evaluation is multidimensional: capability (benchmarks), behavior (alignment/safety), system-level (cost/latency), and agent-level (task success rate).

See also: Benchmark, HELM, MMLU, LLM-as-Judge, Hallucination

F

Failure Mode

A predictable pattern in which an AI system malfunctions. Common agent failure modes include infinite loops, tool misuse, hallucinated tool calls, plan drift, over-retrieval, and cost explosion.

See also: Guardrails, Agentic AI, Hallucination

Few-Shot Learning

The ability of a model to perform a task given only a small number of examples (shots) in the prompt, without updating model parameters. Demonstrated prominently by GPT-3.

See also: In-Context Learning, Pre-training, Prompt Engineering

Fine-Tuning

Adapting a pre-trained model to a specific task or domain by continuing training on targeted data. Types include full fine-tuning, instruction tuning, RLHF, and PEFT/LoRA.

See also: RLHF, Instruction Tuning, LoRA, PEFT, Transfer Learning

Foundation Model

A large model trained on broad, general data that can be adapted to many downstream tasks. Examples include GPT-4o, Claude 3.5 Sonnet, Gemini 1.5 Pro, LLaMA 3.x, and Mistral.

See also: Pre-training, Transfer Learning, LLM

G

Generative Model

A model that produces new content — text, images, audio, or code — rather than classifying existing inputs. LLMs are a primary example of generative models.

See also: LLM, Foundation Model, Decoding

GPQA (Graduate-Level Google-Proof Q&A)

A challenging frontier benchmark designed to test expert-level reasoning in science and other domains. Used to evaluate the most capable LLMs beyond standard academic knowledge tests.

See also: Benchmark, MMLU, Evaluation

Gradient Descent

An iterative optimization algorithm that minimizes a loss function by updating parameters in the direction of the negative gradient.

$$\theta = \theta - \alpha \cdot \nabla L(\theta)$$

Where α = learning rate, $\nabla L(\theta)$ = gradient of the loss.

See also: Backpropagation, Learning Rate, Optimization

GSM8K

A benchmark consisting of grade school math word problems, used to evaluate multi-step arithmetic and reasoning ability in LLMs.

See also: Benchmark, Evaluation, MMLU

Guardrails

Safety and constraint mechanisms built into LLM applications to prevent unsafe outputs, scope violations, tool misuse, or harmful behavior. Implemented through prompt constraints, output filters, content moderation, and step limits.

See also: Alignment, Failure Mode, Prompt Engineering

H

Hallucination

When an LLM generates incorrect, fabricated, or misleading information that appears plausible but is not grounded in retrieved or factual context. A primary challenge in deployed LLM systems.

See also: RAG, Guardrails, Evaluation, Grounding

HELM (Holistic Evaluation of Language Models)

A comprehensive benchmark framework that evaluates LLMs across multiple dimensions including accuracy, fairness, robustness, and efficiency — going beyond single-task metrics.

See also: Benchmark, Evaluation, MMLU

Hybrid RAG

A retrieval strategy combining keyword-based search (e.g., BM25) with vector similarity search to improve retrieval recall and precision over either method alone.

See also: RAG, BM25, Vector Similarity Search, Agentic RAG

HumanEval

A benchmark from OpenAI measuring the coding ability of LLMs by testing functional code correctness on programming problems.

See also: Benchmark, Evaluation, MMLU

I

In-Context Learning

The ability of LLMs to adapt their behavior based on examples or instructions provided within the prompt, without any weight updates. Enabled by the scale of pre-training.

See also: Few-Shot Learning, Prompt Engineering, Pre-training

Inference

The process of using a trained model to generate predictions or outputs for new inputs. Distinct from training (which updates weights). Inference cost scales with input/output token count.

See also: Decoding, Token, Latency, Cost

Instruction Tuning

A form of supervised fine-tuning in which a model is trained on instruction–response pairs to improve helpfulness, instruction-following, and output consistency.

See also: Fine-Tuning, RLHF, Alignment

L

Latency

The time taken for a model or agent system to produce a complete response. In agentic systems, latency stacks across multiple tool calls and reasoning loops.

See also: Inference, Cost, Agentic AI

Learning Rate

A hyperparameter (α) that controls the size of parameter update steps during gradient descent. Too high causes instability; too low causes slow convergence.

See also: Gradient Descent, Optimization, Fine-Tuning

LiveCodeBench

A recent benchmark evaluating LLM coding performance on real-world programming tasks, designed to reduce benchmark contamination by using continuously updated problems.

See also: Benchmark, HumanEval, Evaluation

LLM (Large Language Model)

A large-scale neural network based on the Transformer architecture, trained on vast text data using next-token prediction. LLMs are the reasoning engine underlying modern AI applications and agents.

See also: Transformer, Foundation Model, Agent, Pre-training

LLM-as-Judge

An evaluation technique in which a capable LLM (e.g., GPT-4o or Claude 3.5 Sonnet) is used to assess the quality of another model's outputs at scale, supplementing or partially replacing human review.

See also: Evaluation, Benchmark, RLHF

LoRA (Low-Rank Adaptation)

A parameter-efficient fine-tuning method that inserts small trainable low-rank matrices into model layers instead of updating all weights. Significantly reduces compute and memory requirements for fine-tuning.

See also: PEFT, Fine-Tuning, Transfer Learning

Loss Function

A mathematical function measuring the difference between model predictions and true targets. Minimizing the loss function is the core objective of model training.

$$L = -\Sigma \; y \cdot \log(\hat{y})$$

See also: Optimization, Cross-Entropy Loss, Gradient Descent

M

MCP (Model Context Protocol)

An open protocol developed by Anthropic — and broadly adopted across the AI ecosystem — that standardizes how LLMs communicate with external tools and resources. MCP defines tool schemas, context injection rules, execution layers, and resource definitions.

$$LLM \leftrightarrow MCP\ Middleware \leftrightarrow Tools\ /\ Resources$$

See also: Tool Use, Agentic AI, Guardrails, Agent

Memory (Short-Term)

Conversation history and intermediate outputs stored within the context window during a single session. Volatile — lost when the session ends or the context window is exceeded.

See also: Context Window, Memory (Long-Term), Agent

Memory (Long-Term)

External storage used to persist information beyond the context window. Implemented via vector databases, relational databases, or structured state stores. Enables agents to recall past interactions and user preferences.

See also: Vector Database, Memory (Short-Term), Agentic AI

MMLU (Massive Multitask Language Understanding)

A widely used benchmark covering 57 academic subjects in multiple-choice format, testing general reasoning, knowledge breadth, and subject-matter understanding across LLMs.

See also: Benchmark, HELM, GPQA, Evaluation

Multi-Agent System

An architecture in which multiple specialized agents collaborate to complete complex tasks. Each agent may have a distinct role (e.g., research, summarization, critique, planning). Increases capability but also coordination complexity and cost.

See also: Agent, Agentic AI, Planning Agent

N

Neural Network

A computational model composed of stacked layers of interconnected nodes (neurons). Each layer applies a linear transformation followed by a non-linear activation function, enabling hierarchical representation learning.

$$a^{(l)} = \sigma(W^{(l)} \cdot a^{(l-1)} + b^{(l)})$$

See also: Deep Learning, Activation Function, Transformer

Next-Token Prediction

The primary pre-training objective of autoregressive LLMs: predict the probability of the next token given all preceding tokens.

$$P(x_t \mid x_{<t})$$

This simple objective, applied at enormous scale, gives rise to emergent language understanding and reasoning capabilities.

See also: Pre-training, LLM, Cross-Entropy Loss

O

Open-Weight Model

A model whose parameters are publicly released, allowing inspection, local deployment, and fine-tuning. Examples: LLaMA 3.x (Meta), Mistral, Mixtral, Falcon, Qwen, and DeepSeek-R1. Contrasted with proprietary models (GPT-4o, Claude, Gemini).

See also: Foundation Model, Fine-Tuning, LLM

Optimization

The process of minimizing a loss function to improve model performance. The central mathematical activity in machine learning.

$$\theta^* = \arg\min_\theta L(\theta)$$

See also: Gradient Descent, Loss Function, Parameter

Overfitting

A training failure in which a model learns to perform well on training data but generalizes poorly to unseen data, by memorizing noise rather than learning meaningful patterns.

See also: Regularization, Dataset, Fine-Tuning

P

Parameter

A trainable numerical value inside a model — weights and biases — that is updated during training via gradient descent. Modern LLMs contain billions to trillions of parameters.

See also: Weight, Bias, Optimization, Fine-Tuning

PEFT (Parameter-Efficient Fine-Tuning)

A family of fine-tuning techniques that update only a small subset of model parameters — typically via adapters or low-rank matrices — while keeping most weights frozen. Includes LoRA, Prefix Tuning, and Adapters.

See also: LoRA, Fine-Tuning, Transfer Learning

Planning Agent

An agent that decomposes tasks into structured sub-steps, executes them sequentially or in parallel, evaluates intermediate results, and revises its plan as needed.

$$\textit{Goal} \rightarrow \textit{Plan} \rightarrow \textit{Execute} \rightarrow \textit{Evaluate} \rightarrow \textit{Refine}$$

See also: Reactive Agent, ReAct Pattern, Agentic AI

Pre-training

Large-scale training of a foundation model on general, diverse data using a broad objective (next-token prediction). Pre-training builds the general-purpose language representations that enable downstream adaptation.

See also: Foundation Model, Fine-Tuning, Next-Token Prediction, Chinchilla Scaling Laws

Prompt Engineering

The practice of designing structured instructions — system prompts, user prompts, examples, and constraints — to reliably guide LLM behavior and outputs. In production systems, prompt engineering is constraint design.

See also: System Prompt, In-Context Learning, Guardrails

Q

Query Embedding

A vector representation of a user's query generated by an embedding model, used in RAG systems to perform similarity search against indexed document embeddings.

See also: Embedding, Cosine Similarity, RAG, Vector Database

Query / Key / Value (Q, K, V)

The three projection matrices in the self-attention mechanism. The Query represents what a token is looking for; the Key represents what each token offers; the Value is the actual content retrieved.

$$\mathit{Attention}(Q, K, V) = \mathit{softmax}(QK^T / \sqrt{d\_k}) \cdot V$$

See also: Self-Attention, Attention Mechanism, Transformer

R

RAG (Retrieval-Augmented Generation)

A system architecture combining external document retrieval with LLM generation. The model retrieves relevant chunks, injects them into the prompt, and generates grounded answers — reducing hallucination and enabling knowledge updates without retraining.

See also: Embedding, Vector Database, Chunking, Hybrid RAG, Agentic RAG

Reactive Agent

An agent that responds directly to the current input without maintaining long-term plans or decomposing tasks. Suitable for straightforward question-answering and retrieval tasks.

See also: Planning Agent, Agent, Agentic AI

ReAct Pattern

An influential agent reasoning pattern combining Reasoning and Acting in an iterative loop: the model reasons about what to do, executes a tool action, observes the result, and continues until the goal is achieved.

$$\textit{LLM} \rightarrow \textit{Reasoning} \rightarrow \textit{Tool Call} \rightarrow \textit{Observation} \rightarrow \textit{Reasoning} \rightarrow \textit{Answer}$$

See also: Planning Agent, Agentic AI, Tool Use

Reinforcement Learning (RL)

A learning paradigm in which an agent learns by taking actions in an environment and receiving reward signals. The goal is to find a policy that maximizes expected cumulative reward.

$$\pi^* = \textit{argmax}_\pi \; E[R]$$

See also: RLHF, Agent, Optimization

Representation Learning

The automatic learning of useful feature representations from raw data, eliminating the need for manual feature engineering. Deep neural networks perform representation learning by transforming inputs into increasingly abstract representations across layers.

See also: Deep Learning, Embedding, Neural Network

RLHF (Reinforcement Learning from Human Feedback)

An alignment technique in which model outputs are ranked by human annotators, a reward model is trained on these rankings, and the LLM policy is then optimized to maximize the learned reward.

$$\pi^* = \text{argmax}_\pi \; E[R]$$

See also: Reinforcement Learning, Alignment, Instruction Tuning, Fine-Tuning

S

Scaling Laws

Empirical relationships showing that LLM performance improves predictably as a function of model size (parameters), training data, and compute budget. The Chinchilla scaling laws (Hoffmann et al., 2022) refined earlier findings by showing optimal compute allocation requires proportionally scaling both data and parameters.

See also: Chinchilla Scaling Laws, Pre-training, Foundation Model

Self-Attention

A mechanism in which every token in a sequence attends to every other token, computing a weighted combination of their value representations based on query-key similarity. The foundation of the Transformer architecture.

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k}) \cdot V$$

See also: Attention Mechanism, Transformer, Query / Key / Value

Stochastic Decoding

Text generation strategies that introduce randomness, such as top-k sampling and top-p (nucleus) sampling. Controlled by the temperature parameter.

See also: Temperature, Deterministic Decoding, Decoding

Supervised Learning

A machine learning paradigm in which models are trained on labeled input–output pairs. The objective is to minimize prediction error on the labeled targets.

$$L = -\Sigma\ y \cdot \log(\hat{y})$$

See also: Unsupervised Learning, Reinforcement Learning, Loss Function

System Prompt

A prompt provided to the model before the user turn that defines model behavior, persona, scope, and constraints. In production systems, the system prompt is the primary mechanism for behavioral control.

See also: Prompt Engineering, Guardrails, Alignment

T

Temperature

A decoding parameter controlling the randomness of token selection. Higher temperature increases diversity and creativity; lower temperature increases determinism and consistency. Typically set lower in production educational and enterprise systems.

See also: Decoding, Stochastic Decoding, Deterministic Decoding

Token

The fundamental unit of text processed by a language model. Tokens can represent full words, subwords, punctuation, or characters depending on the tokenizer. Most modern LLMs use subword tokenization (e.g., Byte Pair Encoding).

See also: Tokenization, Context Window, LLM

Tokenization

The process of splitting raw text into tokens before passing it to a language model. The tokenizer maps text to integer IDs that correspond to model vocabulary entries. Example: 'unbelievable' → ['un', 'believ', 'able'].

See also: Token, LLM, Context Window

Tool Use

The ability of an LLM-based agent to invoke external tools — such as web search, calculators, code execution environments, databases, and APIs — to extend its capabilities beyond text generation.

See also: Agent, MCP, ReAct Pattern, Agentic AI

Transfer Learning

The practice of applying knowledge learned during pre-training on large general data to new, more specific tasks. Transfer learning is the foundation of the modern LLM application paradigm.

See also: Pre-training, Fine-Tuning, Foundation Model

Transformer

The dominant neural network architecture for LLMs, introduced in 'Attention Is All You Need' (Vaswani et al., 2017). Processes sequences in parallel using self-attention, enabling long-range dependency capture and massive scalability.

See also: Self-Attention, Attention Mechanism, LLM

U

Unsupervised Learning

A machine learning paradigm that discovers patterns, structure, or representations in data without labeled outputs. Examples include clustering, dimensionality reduction, and anomaly detection.

See also: Supervised Learning, Reinforcement Learning, Representation Learning

V

Vector Database

A database optimized for storing, indexing, and retrieving high-dimensional vector embeddings. Enables fast approximate nearest-neighbor search for RAG retrieval. Examples: Pinecone, Weaviate, FAISS, Chroma, Qdrant, and pgvector.

See also: Embedding, RAG, Cosine Similarity, Approximate Nearest Neighbor

Vector Similarity Search

The process of finding the most semantically similar documents to a query by computing distances (e.g., cosine similarity) between their embeddings in high-dimensional space.

$$\cos(\theta) = (A \cdot B) / (|A| \cdot |B|)$$

See also: Cosine Similarity, Vector Database, RAG

W

Weight

A trainable parameter in a neural network that determines the strength of connections between neurons. Weights are updated during training via backpropagation and gradient descent.

See also: Parameter, Bias, Backpropagation, Gradient Descent

Z

Reserved for future additions as the field evolves.

End of Glossary

Foundations of LLM Applications & Agentic AI Systems — Modules 1–10

📚 End of Core Curriculum

Foundations of LLM Applications & Agentic AI Systems

Modules 1–10 Complete