TECHNICAL REFERENCE EBOOK • COMPLETE EDITION

The Complete Guide to Large Language Models (LLMs)

From Transformer Foundations and Training Pipelines to RAG, Fine-Tuning, Agents, and Production Deployment

Published by **www.tulasinepali.com.np** • Comprehensive AI & Machine Learning Study Handbook

TABLE OF CONTENTS

1. **Foundations of Large Language Models:** Evolution from N-grams & RNNs to Transformers
2. **Tokenization, Embeddings & Positional Encoding:** How Text Becomes Geometry
3. **Inside the Transformer Architecture:** Self-Attention, GQA, MoE & Decoding
4. **The LLM Training Lifecycle:** Pre-training, Scaling Laws, SFT, RLHF, DPO & Reasoning RL
5. **Inference Optimization & Local Deployment:** KV Cache, Quantization & Serving Engines
6. **Prompt Engineering & Structured Reasoning:** Zero-Shot, Few-Shot, CoT & JSON Schemas
7. **Retrieval-Augmented Generation (RAG):** Chunking, Vector Databases, Hybrid Search & Reranking
8. **Parameter-Efficient Fine-Tuning (PEFT):** LoRA, QLoRA & Dataset Engineering
9. **Function Calling, MCP & Autonomous Agents:** Tool Use, ReAct Loops & Memory
10. **Evaluation, Benchmarks, Safety & Security:** Metrics, Hallucination Mitigation & Guardrails

CHAPTER 1

Foundations of Large Language Models

A **Large Language Model (LLM)** is a deep neural network trained on massive text and multimodal datasets to model probability distributions over sequences of tokens. At its core, an autoregressive LLM solves one fundamental mathematical objective: given a sequence of preceding tokens ($x_1, x_2, \ldots, x_{t-1}$), predict the probability distribution of the next token x_t.

By scaling this simple next-token prediction objective across trillions of words and billions of parameters, LLMs exhibit **emergent capabilities**—including translation, code synthesis, logical deduction, summarization, and multi-step planning—without requiring task-specific architectural changes.

www.tulasi

Evolution of Language Modeling

Era / Paradigm	Core Mechanism	Key Limitation
Statistical N-Grams	Counts conditional word co-occurrence frequencies over fixed windows of *n* words.	Sparsity explosion for large *n*; zero semantic generalization to unseen phrasing.
RNNs & LSTMs	Processes tokens sequentially one by one while updating a recurrent hidden state vector.	Vanishing gradients over long distances; sequential bottleneck prevents GPU parallelization.
Transformers (2017+)	Processes all tokens in parallel using multi-head self-attention across the entire context.	Quadratic memory/compute scaling $O(n^2)$ with respect to sequence length n.

Encoder-Only vs. Encoder-Decoder vs. Decoder-Only

- **Encoder-Only (e.g., BERT, RoBERTa):** Bidirectional attention; every token attends to tokens both before and after it. Ideal for text classification, named entity recognition, and generating dense search embeddings.
- **Encoder-Decoder (e.g., T5, BART):** Encodes the input sequence into contextual representations and autoregressively decodes a target sequence. Historically used for machine translation and seq2seq summarization.
- **Decoder-Only (e.g., GPT, Llama, Claude, Gemini):** Uses **causal (masked) self-attention** so each token only attends to itself and preceding tokens. Dominant architecture for modern general-purpose LLMs due to superior scaling efficiency and unified generation.

www.tulasinepali.com.np

CHAPTER 2

Tokenization, Embeddings & Positional Encoding

Neural networks cannot operate directly on Unicode characters or strings; they require continuous floating-point tensors. Transforming raw text into model inputs happens in three sequential stages.

1. Subword Tokenization

Character-level vocabularies create excessively long sequences, while word-level vocabularies suffer from millions of rare words and out-of-vocabulary failures. Modern LLMs use **subword tokenization** algorithms—primarily **Byte-Pair Encoding (BPE)** and **SentencePiece**—which keep frequent words intact while splitting rare or morphological words into reusable pieces (e.g., `unbelievable` → `un` + `believ` + `able`).

- **Vocabulary Size:** Modern frontier models use vocabularies between `32,000` and `256,000` tokens. Larger vocabularies compress text into fewer tokens per sentence, speeding up inference and improving multilingual coverage.
- **Rule of Thumb:** In English prose, 1 token equals roughly 4 characters or ~0.75 words. Numbers, whitespace indentation in code, and non-Latin scripts often consume more tokens per word.

2. Token Embeddings

Each integer token ID acts as an index into a learned **Embedding Matrix** $W_E \in \mathbb{R}^{V \times d}$, where V is the vocabulary size and d is the model's hidden dimension (e.g., `d = 4096`). Through training, semantically related concepts self-organize into nearby geometric regions in this high-dimensional vector space.

3. Positional Encoding

Because self-attention processes all tokens simultaneously without recurrence, the architecture has no innate awareness of word order (`"Dog bites man"` vs. `"Man bites dog"`). Models inject order via positional representations:

- **Sinusoidal / Learned Absolute Embeddings:** Adds fixed or learned position vectors directly to token embeddings (early Transformers).
- **RoPE (Rotary Position Embeddings):** Rotates Query and Key vectors in complex 2D subspaces proportional to token position m. Because the dot product depends only on relative distance $(m - n)$, RoPE enables strong long-context extrapolation and is standard across Llama, Mistral, Qwen, and Gemma architectures.

CHAPTER 3

Inside the Transformer Architecture

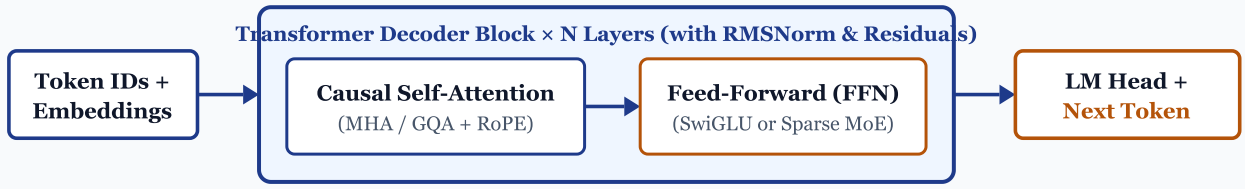


Figure 1: Data flow inside a modern decoder-only Transformer architecture.

Scaled Dot-Product Self-Attention

For every token representation X, learned projection matrices compute three vectors: **Query (Q)** (what information this token is looking for), **Key (K)** (what information this token offers), and **Value (V)** (the actual content passed forward).

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

Here, $\sqrt{d_k}$ stabilizes gradient variance, and M is a causal upper-triangular mask set to $-\infty$ so tokens cannot peek at future positions during training. Modern architectures optimize memory bandwidth using **Grouped-Query Attention (GQA)**, where multiple Query heads share a smaller number of Key and Value heads.

www.tulasinepali.com.np

Feed-Forward Networks & Mixture-of-Experts (MoE)

After attention mixes information across tokens, a position-wise Feed-Forward Network (typically using **SwiGLU** activations and **RMSNorm** pre-normalization) transforms each token independently. In **Sparse Mixture-of-Experts (MoE)** models (e.g., Mixtral, DeepSeek), the dense FFN is replaced by multiple parallel "expert" networks plus a learned router. For each token, the router activates only the top-k experts (e.g., 2 out of 8), dramatically lowering inference compute per token while retaining massive total parameter capacity.

Decoding & Sampling Strategies

Parameter / Strategy	How It Works	Recommended Setting
Greedy Decoding	Always picks the single token with the highest probability ($T = 0$).	Deterministic code, math, entity extraction.
Temperature (T)	Scales logits before softmax: $p_i \propto exp(z_i / T)$. Lower sharpens peaks; higher flattens distribution.	`0.0–0.2` factual; `0.7–0.9` creative writing.
Top-k Sampling	Truncates candidate pool to the k most probable tokens before sampling.	`k = 40` or `50` to prevent tail-token nonsense.
Top-p (Nucleus)	Selects the smallest set of top tokens whose cumulative probability exceeds p.	`p = 0.90–0.95` for adaptive diversity.

CHAPTER 4

The LLM Training Lifecycle

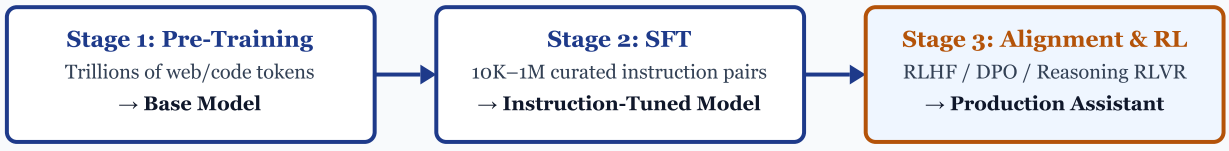


Figure 2: From raw unsupervised text prediction to aligned conversational reasoning.

- **Stage 1 — Pre-Training & Scaling Laws:** Trained using cross-entropy loss across trillions of filtered, deduplicated tokens. Research such as the *Chinchilla Scaling Laws* demonstrated that compute-optimal pre-training scales model parameters (N) and training tokens (D) in equal proportion (~20 tokens per parameter), though modern models over-train smaller architectures on 10T–15T+ tokens to make inference dramatically cheaper.
- **Stage 2 — Supervised Fine-Tuning (SFT):** Teaches the base model conversational formatting (system/user/assistant roles), refusal boundaries, and multi-turn instruction adherence using high-quality human-written and verified synthetic demonstrations.

- **Stage 3 — Preference Alignment & Reasoning Reinforcement Learning:**
  - **RLHF (PPO / GRPO):** Human annotators rank multiple candidate answers; a Reward Model learns human preferences, and reinforcement learning updates the policy model while a KL-divergence penalty prevents reward hacking.
  - **DPO (Direct Preference Optimization):** Eliminates the separate reward model by optimizing the LLM directly on pairs of $(y_{chosen}, y_{rejected})$ responses via a closed-form classification loss.
  - **Reasoning RL (RLVR):** Trains models with verifiable rewards on math proofs and unit-tested code, encouraging the model to generate extended internal Chain-of-Thought verification steps before answering.

CHAPTER 5

Inference Optimization & Local Deployment

Deploying LLMs efficiently requires overcoming two distinct performance regimes: the **Prefill Phase** (compute-bound parallel processing of the input prompt) and the **Decode Phase** (memory-bandwidth-bound sequential generation of one output token at a time).

www.tulasinepali.com.np

- **KV Cache (Key-Value Cache):** During autoregressive generation, Key and Value tensors for past tokens remain unchanged. Caching them in GPU VRAM avoids recomputing the entire prefix at every step, reducing generation complexity from $O(n^2)$ to $O(n)$ per step at the cost of higher memory usage.
- **PagedAttention & Continuous Batching:** Serving engines like **vLLM** and **SGLang** manage KV cache memory in non-contiguous virtual blocks (eliminating fragmentation) and dynamically insert new user requests into active GPU batches without waiting for long generations to finish.
- **Speculative Decoding:** A tiny, fast "draft" model proposes 4–6 candidate tokens ahead, and the large target model verifies all of them in a single parallel forward pass—yielding 2×–3× speedups with mathematically identical output distributions.
- **Quantization:** Compresses model weights from 16-bit (`BF16` / `FP16`) to 8-bit or 4-bit integers. Popular formats include **GGUF** (for CPU/Apple Silicon/hybrid execution via `llama.cpp` and `Ollama`) and **AWQ / GPTQ / FP8** (for high-throughput GPU inference). As a rule of thumb, a 4-bit quantized model requires roughly `0.6 GB of VRAM per billion parameters` (e.g., a 70B 4-bit model fits comfortably in ~42 GB VRAM).

CHAPTER 6

Prompt Engineering & Structured Reasoning

Prompt engineering is the systematic design of inputs and context structures to steer model behavior reliably without modifying weights.

Technique	Structure & Mechanism	Best Use Case
System Role & Constraints	Defines persona, tone, domain rules, and explicit negative constraints in the system message.	Consistent assistant behavior and policy compliance.
Few-Shot Prompting	Provides 2–5 concrete input-output examples directly inside the prompt.	Teaching exact formatting, edge-case handling, and classification labels.
Chain-of-Thought (CoT)	Instructs the model to decompose problems into intermediate reasoning steps before concluding.	Arithmetic, multi-hop logic, debugging, and complex analysis.
Structured Outputs	Enforces a strict JSON Schema or Pydantic grammar mask during token decoding.	Reliable API pipelines, database ingestion, and UI rendering.

> **Production Tip: Separate instructions from untrusted user data using clear XML delimiters (e.g., <context>...</context>) and always specify fallback behavior ("If the answer is not in the context, state Insufficient Information").**

CHAPTER 7

Retrieval-Augmented Generation (RAG)

RAG grounds an LLM in external, up-to-date, or private knowledge bases at query time—solving knowledge cutoffs and drastically reducing factual hallucinations without expensive retraining.

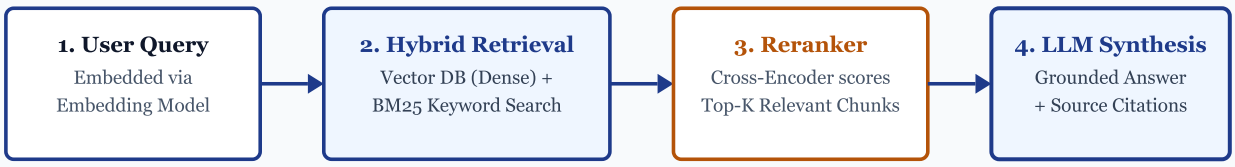


Figure 3: Production Hybrid RAG pipeline with cross-encoder reranking.

End-to-End RAG Best Practices

1. **Document Parsing & Chunking:** Clean PDFs/HTML into structured Markdown preserving headings and tables. Split text using recursive semantic chunking (typically `400–800 tokens` with `10–15% overlap`) so ideas stay intact.
2. **Hybrid Search (Dense + Sparse):** Combine semantic vector similarity (cosine similarity over embeddings) with lexical keyword search (**BM25**) using Reciprocal Rank Fusion (RRF). Dense vectors capture conceptual synonyms; BM25 catches exact product codes, acronyms, and legal citations.
3. **Cross-Encoder Reranking:** Retrieve the top 25–50 candidate chunks quickly, then pass `(Query, Chunk)` pairs through a cross-encoder reranker to select the top 5 highest-precision passages for the final prompt.

CHAPTER 8

Parameter-Efficient Fine-Tuning (PEFT)

When prompt engineering and RAG cannot enforce a specialized domain style, latency budget, or complex task schema, **Fine-Tuning** adapts the model's internal parameters directly.

- **Full Fine-Tuning vs. LoRA:** Updating all 70B parameters requires massive multi-GPU clusters and risks *catastrophic forgetting*. **LoRA (Low-Rank Adaptation)** freezes the original pretrained weights W_0 and injects tiny trainable rank-decomposition matrices $\Delta W = B \cdot A$ (where rank $r \ll d$, e.g., `r = 16` or `32`) into attention and MLP projections—reducing trainable parameters by over 99%.
- **QLoRA:** Quantizes the frozen base model to 4-bit NormalFloat (`NF4`) with paged optimizers while computing LoRA adapter gradients in `BF16`, enabling fine-tuning of 30B–70B models on a single workstation GPU.
- **Data Quality Rule:** A curated dataset of `1,000–5,000` clean, diverse, human-verified examples consistently outperforms 100,000 noisy scraped examples.

Decision Criteria	Choose RAG When…	Choose Fine-Tuning (LoRA) When…
Knowledge Type	Facts change frequently (docs, policies, inventory, live tickets).	Behavior, formatting, tone, or domain syntax must be internalized.
Traceability	You require strict inline citations and document-level access permissions.	You want compact prompts without stuffing long style guides into every call.
Latency & Cost	Context retrieval overhead is acceptable for high factual accuracy.	Distilling a large frontier model into a fast, low-cost 7B/8B specialist.

CHAPTER 9

Function Calling, MCP & Autonomous Agents

An **AI Agent** pairs an LLM reasoning engine with external tools, memory, and an iterative execution loop to accomplish multi-step goals autonomously.

- **Function Calling (Tool Use):** Developers pass JSON tool definitions (name, description, parameter schema) to the LLM. Instead of answering in plain text, the model emits a structured tool-call object (e.g., `query_sql_db(query="SELECT ...")`); the application executes the function securely and feeds the output back into the model's context.
- **Model Context Protocol (MCP):** An open standard that decouples AI applications from external data sources and tools via standardized MCP servers—allowing any MCP-compatible client to plug into databases, Git repositories, file systems, and APIs uniformly.

- **The ReAct Loop (Reason + Act):** Agents operate through cycles of *Thought → Tool Action → Observation* until a termination condition is met. Production systems constrain agents with maximum step budgets, deterministic workflow graphs (state machines), and human-in-the-loop approval gates for destructive actions (e.g., sending emails, executing payments, or modifying production databases).

CHAPTER 10

Evaluation, Benchmarks, Safety & Security

Rigorous evaluation and defense-in-depth security are mandatory before shipping LLM applications to real users.

Evaluation Framework

- **Public Benchmarks:** General knowledge & reasoning (*MMLU-Pro, GPQA Diamond*), software engineering (*SWE-bench Verified, LiveCodeBench*), mathematics (*MATH-500, AIME*), and human preference leaderboards (*Chatbot Arena Elo*).
- **Application-Level Evals:** Custom golden test sets scored via deterministic assertions (regex/JSON validity/unit tests), **LLM-as-a-Judge** rubrics, and RAG-specific metrics (*Context Precision, Faithfulness/Groundedness, and Answer Relevance*).

Security Risks & Guardrails

Vulnerability / Risk	Attack / Failure Mechanism	Engineering Mitigation
Prompt Injection (Direct & Indirect)	Malicious instructions hidden in user input, web pages, or emails override system rules.	Privilege separation, untrusted data delimiters, input classifiers, and human confirmation for sensitive tool calls.
Hallucinations	Model fabricates plausible-sounding facts, APIs, or legal citations.	Hybrid RAG grounding, low temperature, self-consistency checks, and citation verification.
Data & PII Leakage	Sensitive user or proprietary training data exposed in generated outputs or logs.	Pre-inference PII redaction/masking, zero-data-retention API agreements, and role-based RAG ACLs.
Insecure Output Handling	LLM-generated SQL, shell commands, or HTML executed directly without sanitization.	Parameterized SQL queries, sandboxed code containers (Docker/gVisor), and strict output validation.

> **Final Summary: Master the fundamentals—Transformers process tokens via self-attention, pre-training builds world knowledge, alignment shapes helpfulness, RAG provides live verifiable facts, LoRA specializes behavior, and rigorous evaluations keep production systems safe and reliable.**