



AI
Bees

Generative AI

Generative AI — Complete Revision Summary

AI/ML • NLP • Transformers • LLMs • Prompt Engineering • RAG • Evaluation • On-Premise • Guardrails

Revision Notes for Students — All Topics, High-Level Theory & Quick Reference

Course Roadmap — Learning Order

Each section builds on the previous. Work through them in order for the clearest understanding:

#	Topic
1	AI / ML Foundations — Types, Regression, Classification, Clustering
2	Deep Learning — Neural Networks, Backprop, Architectures
3	NLP — Traditional Pipeline, Tokenisation, TF-IDF
4	Generative AI & Transformers — How LLMs Work, Attention, GPT, BERT
5	LLM Setup — Virtual Environments, Poetry, LangChain, Gemini API
6	Prompt Engineering — Zero-Shot, Few-Shot, CoT, ReAct, RAG Pattern
7	RAG & Vector Search — Embeddings, Chunking, Vector DBs, iRAG
8	Evaluation Metrics — BLEU, ROUGE, BERTScore, RAGAS, DeepEval
9	On-Premise LLMs & Ollama — Why Local, Setup, Model Commands
10	Fine-Tuning, PEFT, LoRA, QLoRA — Theory & Differences
11	Guardrails — PII Detection, Input/Output Filters, RAG Integration

1. AI / ML Foundations

1.1 The AI Hierarchy

AI is the broadest field — any technique enabling machines to simulate intelligence. Everything below is a specialised subset nested inside it like Russian dolls:

Layer	Core Idea	Examples
AI	Any machine intelligence	Chess engines, expert systems, everything below
Machine Learning	Learns patterns from data — no hard-coded rules	Random Forest, XGBoost, K-Means
Deep Learning	Multi-layer neural networks — learns features automatically	CNN (images), LSTM (sequences), Transformers

NLP	Processing and understanding human language	Spam filter, translation, ChatGPT
Generative AI	Creates new content from learned patterns	ChatGPT, DALL-E, GitHub Copilot, Sora

🔑 **Remember:** ChatGPT is simultaneously AI → ML → DL → NLP → GenAI. All labels are correct at different levels of the hierarchy.

1.2 Three Types of Machine Learning

- **Supervised Learning:** trains on labelled data (input + known output). Goal: predict outputs for new inputs. Examples: Regression (house price), Classification (spam/not spam).
- **Unsupervised Learning:** no labels — algorithm finds hidden structure. Example: Clustering (customer segmentation with K-Means).
- **Reinforcement Learning:** agent learns by trial and error, receives rewards for good actions. Example: game-playing AI, robotics.

1.3 Regression vs Classification vs Clustering

Type	Output	Example Task	Key Metric
Regression	Continuous number	Predict house price (\$320,000)	RMSE — lower is better
Classification	Category / label	Spam or Not Spam	Precision, Recall, F1
Clustering	Discovered groups	Customer segments (no labels given)	Silhouette score

2. Deep Learning

2.1 Neural Network Basics

A neural network is a series of layers of neurons. Each neuron: receives inputs → multiplies by weights → sums → applies activation function → passes to next layer. Training adjusts weights to minimise prediction error.

- **Input Layer:** one neuron per feature — receives raw data.
- **Hidden Layers:** learn intermediate representations. More layers = "deeper" network.
- **Output Layer:** one neuron for regression; one per class for multi-class classification.
- **Backpropagation:** sends errors backwards through the network, nudging each weight in the direction that reduces the error.
- **Epoch:** one complete pass through the entire training dataset. Training typically needs tens to hundreds of epochs.

💡 **Key Point:** Activation functions: ReLU (hidden layers — fast, avoids vanishing gradient), Sigmoid (binary output 0–1), Softmax (multi-class — probabilities sum to 1).

2.2 Common Architectures

Architecture	Best For	Examples
CNN	Images & visual data	Face recognition, medical imaging, self-driving cars
RNN	Sequential / time-series	Stock price prediction, sensor forecasting
LSTM	Long sequences with context	Text generation, speech recognition

Transformer	Language, long-range dependencies	GPT, BERT, ChatGPT, Claude, Gemini, Google Translate
GAN	Generating realistic new data	Image synthesis, deepfakes, AI art

3. Natural Language Processing (NLP)

3.1 Traditional NLP Pipeline

Before deep learning, NLP used a manual step-by-step pipeline. Understanding this shows why modern LLMs are such a leap forward:

- **Step 1 — Tokenisation:** "I love ML!" → ["I", "love", "ML", "!"] — splitting text into individual units.
- **Step 2 — Stop Word Removal:** remove common low-meaning words ("the", "is", "a") to reduce noise.
- **Step 3 — Stemming/Lemmatisation:** reduce to root form — "running" → "run", "better" → "good".
- **Step 4 — Feature Extraction (BoW / TF-IDF):** convert text to numerical vectors. BoW counts word frequency. TF-IDF weights words by uniqueness across all documents.
- **Step 5 — Model Training:** feed vectors into ML algorithm (Naive Bayes, SVM, Logistic Regression).

⚠ Warning: Big limitation of traditional NLP: "Dog bites man" and "Man bites dog" produce the same Bag-of-Words vector — yet mean opposite things. Word order and context are lost.

3.2 Classic NLP Tasks

Modern LLMs handle all of these in a single model. Knowing the task names helps when reading documentation:

Task	What It Does / Example
Sentiment Analysis	"Great product!" → Positive
Text Classification	News article → Sports / Politics / Tech
Named Entity Recognition	"Apple in California" → [ORG][LOC]
Machine Translation	"Bonjour" → "Hello"
Text Summarisation	5-page article → 3-sentence summary
Question Answering	Q: Who founded Apple? A: Steve Jobs

4. Generative AI & Transformers

4.1 What is Generative AI?

Generative AI creates new, original content — text, images, audio, video, code — that did not previously exist. It goes beyond analysing or classifying data: it generates something entirely new based on learned patterns from training data.

- **Text generation:** ChatGPT, Claude, Gemini — write essays, answer questions, summarise documents.
- **Image generation:** DALL-E 3, Midjourney, Stable Diffusion — create images from text prompts.
- **Code generation:** GitHub Copilot, Code Llama — auto-complete and write code.
- **Audio/Video:** ElevenLabs (voice), Suno (music), Sora (video).

🔑 Remember: How text generation works: Given a prompt, the LLM predicts the most likely next token, appends it, and repeats — called autoregressive generation. The model does not "look up" facts — it learned statistical patterns from training.

4.2 Transformers — The Architecture Behind All Modern LLMs

The Transformer, introduced in the 2017 paper "Attention Is All You Need" (Google), is the neural network architecture that powers every modern LLM: GPT, BERT, Claude, Gemini, Llama, and Mistral. Understanding it at a high level is essential.

The Core Problem Transformers Solve: RNNs and LSTMs processed text sequentially — one word at a time. This was slow and they struggled to connect words that were far apart in a sentence. The Transformer processes all words simultaneously and uses "attention" to directly connect any word to any other word, regardless of distance.

4.3 Self-Attention — The Key Mechanism

Self-attention allows the model to weigh how important every other word is to each word being processed. For each word, the model asks: "which other words in this sentence should I pay most attention to?"

- Example: In "The animal didn't cross the street because it was too tired" — what does "it" refer to? Self-attention lets the model connect "it" directly to "animal" by learning that this connection is most important in context.
- Every word produces three vectors: Query (Q), Key (K), Value (V). Attention score = $(Q \times K) / \sqrt{d}$, then softmax → weighted sum of V vectors.
- Multi-head attention runs several attention computations in parallel, each focusing on different aspects (subject-verb, coreference, etc.), then concatenates the results.

4.4 Transformer Architecture — Two Flavours

Component	Architecture	Training Task	Use Case & Examples
Encoder	BERT, RoBERTa	Masked Language Model — predict hidden words	Text classification, NER, semantic search, embeddings
Decoder	GPT, Llama, Mistral, Gemini	Causal LM — predict next token	Text generation, chatbots, code completion
Encoder-Decoder	T5, BART	Seq2Seq — input to output sequence	Translation, summarisation, Q&A

💡 Key Point: GPT-style (decoder-only) models are the foundation of ChatGPT, Llama, and Claude. BERT-style (encoder-only) models are the foundation of embedding models used in RAG. This is why you need a different model for generating text vs for creating searchable embeddings.

4.5 Key Transformer Concepts

Term	What It Means
Token	The unit the model processes — roughly 0.75 words. "machine" might be one token; "unbelievable" might be 3.
Context Window	Maximum tokens the model can process at once — GPT-4: 128K, Claude: 200K. Sets the "memory" limit per conversation.
Positional Encoding	Since Transformers process all words simultaneously, positional encodings are added to embeddings to preserve word order.
Pre-training	Training on massive web-scale text to learn general language understanding. Happens once — very expensive.
Fine-tuning / RLHF	Further training on curated data + human feedback to make the model follow instructions and be helpful and safe.

Temperature	0 = deterministic output (best for facts/code); 0.7 = balanced (chatbots); 1.5+ = creative/unpredictable.
Hallucination	When an LLM confidently generates incorrect information not supported by evidence. The model predicts statistically plausible text — it does not verify facts.

5. LLM Application Development Setup

5.1 Virtual Environment

An isolated Python installation for one project — prevents version conflicts between projects. Always create one before starting.

Terminal — Create & Activate Virtual Environment

```
# Windows (PowerShell)
py -3.12 -m venv .venv
.\venv\Scripts\activate


# macOS / Linux (Bash)
python3.12 -m venv .venv
source .venv/bin/activate
# Terminal prompt changes to (.venv) when active
```

5.2 Poetry — Dependency Management

Poetry replaces pip + requirements.txt with a single clean workflow. It tracks exact package versions in poetry.lock, guaranteeing reproducible builds across all machines.

Terminal — Install Poetry & Add Packages

```
pip install poetry          # install Poetry once
poetry init                 # add Poetry to existing project
poetry add langchain==0.3.27 # install + pin a package
poetry add python-dotenv langchain-google-genai streamlit
poetry show                 # list all installed packages
poetry run python script.py # run without activating venv
```

 **Remember:** Always commit both pyproject.toml AND poetry.lock to Git. Never commit your .env file — add it to .gitignore.

5.3 Basic LLM Call with LangChain + Gemini

Python — 1_llm_call.py

```
import os
from dotenv import load_dotenv
from langchain_google_genai import ChatGoogleGenerativeAI

load_dotenv() # reads GOOGLE_API_KEY from .env file

llm = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash",
    google_api_key=os.getenv("GOOGLE_API_KEY"),
    temperature=0.7
)

messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Explain RAG in simple terms."}
]
response = llm.invoke(messages)
print(response.content)
```

5.4 Streamlit Web UI

Streamlit builds interactive web apps in pure Python — no HTML/CSS needed. Every widget interaction re-runs the entire script.

Python — 3 llm_with_ui.py (run with: poetry run streamlit run 3_llm_with_ui.py)


```
import streamlit as st
# ... (same llm setup as above)

st.title("Gemini LLM App")
question = st.text_input("Enter your question:")
if st.button("Get Answer"):
    messages = [{"role": "system", "content": "You are helpful."},
                {"role": "user", "content": question}]
    st.write(llm.invoke(messages).content)
# Opens at http://localhost:8501
```

6. Prompt Engineering


6.1 Why Prompts Matter

A prompt is the input text you send to an LLM. LLMs are next-token predictors — the prompt is the "starting text" that shapes everything that follows. The same question phrased differently can produce wildly different outputs.

 **Key Point:** Golden rule: the more specific and structured your prompt, the more specific and structured the output. Always specify: Role, Format, Scope, Length.

6.2 Prompt Techniques — Quick Reference

Technique	Core Idea	When to Use
Zero-Shot	Task only — no examples given	Simple, well-known tasks. Starting point.
Few-Shot	2–5 worked input→output examples before the task	Strict output format (JSON, labels), extraction tasks
Chain of Thought	Add "Let's think step by step." — forces reasoning	Maths, logic, multi-step reasoning
Self-Consistency	Same prompt N times → majority vote on answer	High-accuracy critical reasoning tasks
Role / Persona	"You are a senior data scientist..." — assigns identity	Tone, audience targeting, specialised assistants
Format Control	Specify exact output structure (JSON, bullets, table)	Pipelines, production apps parsing LLM output
ReAct	Reason → Act (use tool) → Observe → Repeat loop	LLM Agents with tools (search, calculator, DB)
Contextual / RAG	Inject retrieved context before asking the question	Private docs, Q&A bots, knowledge bases
Constrained	Explicit do-nots: "Return ONLY JSON. No explanation."	When output is parsed by code; strict format needed

 **Remember:** These techniques combine. A production system prompt often uses: Role + Few-Shot + Format Control + Constraints together.

7. RAG & Vector Search

7.1 Embeddings

An embedding is a dense vector (list of numbers) representing the meaning of text in high-dimensional space. Similar meanings → geometrically close vectors. This is the foundation of semantic search.

Python — Embedding Example

```
# 'cat' → [0.013, -0.278, 0.114, ...] 768 numbers
# 'kitten' → [0.015, -0.265, 0.118, ...] ← very similar!
# 'airplane' → [-0.412, 0.753, ...] ← very different

cosine_similarity("What is the capital of France?",
                  "Which city is the capital of France?") → 0.97
cosine_similarity("What is the capital of France?",
                  "How do I bake a cake?") → 0.12
```

💡 **Key Point:** Cosine similarity measures the angle between two vectors. 1.0 = identical meaning. 0 = unrelated. In RAG, this is how the vector DB finds relevant chunks.

7.2 Why Traditional Search Fails

Method	How it Works	Key Limitation
Keyword / Exact Match	Find documents containing the exact query words	Misses synonyms. "feline" ≠ "cat" to the computer.
Bag of Words (BoW)	Count word frequencies per document	"A dog chased a cat" = "A cat chased a dog" — same vector!
TF-IDF	Weight words by uniqueness across corpus	Better than BoW but still no semantic understanding
Semantic / Embedding	Dense neural embeddings — meaning-aware	Finds synonyms, paraphrases, related concepts ✓

7.3 Vector Databases

Specialised databases for storing and fast-searching high-dimensional vectors using ANN (Approximate Nearest Neighbour) algorithms like HNSW. Instead of exact SQL queries, they answer: "find the 5 vectors most similar to this query."

Tool	Type	Best For	Key Feature
ChromaDB	Open-source	Learning, prototyping ← START HERE	Simple Python API, no config
FAISS	Library (Meta)	Research, offline, local	Fastest; in-memory (no persistence)
Pinecone	Managed cloud	Production, enterprise	Fully managed, auto-scaling
Qdrant	Open-source	Production, on-premise	Rich filtering + vectors

7.4 Chunking — Splitting Documents

Large documents must be split into chunks before embedding. Each chunk is embedded independently and stored as one searchable unit. Chunk size directly affects retrieval quality.

Chunk Size	Trade-off
< 100 tokens	Too small — loses context. Not recommended.
300–500 tokens <input checked="" type="checkbox"/>	Best balance of precision and context. Recommended default.
> 1000 tokens	Too large — retrieval imprecise. Defeats the purpose of RAG.

Chunk Overlap (50–100 tokens): repeat text at boundaries so concepts split across chunks are captured in both adjacent chunks. Example: chunk_size=400, overlap=50.

7.5 RAG Pipeline — Two Phases

Phase	Steps
INDEXING (once, offline)	1. Load documents (PDF, DOCX, web, CSV) → 2. Chunk → 3. Embed each chunk → 4. Store (vector, text, metadata) in vector DB
QUERY (per question)	1. Embed user question → 2. Similarity search → top-K chunks → 3. Build prompt (system + context chunks + question) → 4. LLM generates grounded answer

Python — RAG augmented prompt structure

```
System: Answer ONLY using the context below.  
       If not found, say "I don't know."  
Context: [Chunk 1 text] [Chunk 2 text] [Chunk 3 text]  
Question: What is the parental leave policy?  
Answer:
```

💡 **Key Point:** RAG quality = Retrieval quality × LLM quality. Spend 70% of optimisation effort on the retrieval step. A great LLM cannot compensate for bad retrieval.

7.6 Incremental RAG (iRAG)

Traditional RAG: add 1 new document to 1,000,000 documents → must re-embed ALL 1,000,000 documents.
iRAG: detect only changed chunks → embed only those → insert only those into vector DB.

Aspect	Traditional RAG	Incremental RAG (iRAG)
Update method	Re-index everything from scratch	Only process changed chunks
Update time	Hours for large collections	Seconds to minutes
Cost	Re-embed millions of chunks	Only new/changed chunks (~100x cheaper)
Vector DB needed	In-memory OK (FAISS)	Must be persistent (ChromaDB, Qdrant)

8. Evaluation Metrics

8.1 Classification Metrics Foundation

These metrics appear everywhere — ROUGE, BERTScore, and RAGAS all compute them. They originate from the Confusion Matrix:

Metric	Formula	Asks	High = Good When
Precision	$TP / (TP + FP)$	Of things I flagged, how many were correct?	False alarms are costly (spam filter)
Recall	$TP / (TP + FN)$	Of all true positives, how many did I catch?	Missing cases is costly (cancer diagnosis)
F1 Score	$2 \times (P \times R) / (P + R)$	Harmonic mean — balanced score	Imbalanced classes — use instead of Accuracy
Accuracy	$(TP + TN) / \text{Total}$	Overall, how often was I right?	Only when classes are roughly balanced

8.2 Text Similarity Metrics

Metric	How It Works	Strength	Limitation
BLEU	Precision of n-gram overlap with reference	Machine translation evaluation	Score = 0 if any n-gram size misses
ROUGE-1/L	Recall of reference words in generated text	Summarisation quality	Ignores synonyms — "cat" ≠ "feline"
BERTScore	Token-level cosine similarity via BERT embeddings	Understands synonyms & paraphrases	Needs BERT model; slower
Cosine Similarity	Angle between sentence embedding vectors	No reference needed — compare any two texts	Sentence-level only; needs embed model

8.3 RAG-Specific Evaluation — RAGAS

BLEU/ROUGE/BERTScore cannot evaluate RAG pipelines — they have no concept of retrieved context. RAGAS was built specifically for this:

Metric	Measures	Low Score Means
Faithfulness (#1 Priority)	Are all answer claims supported by the retrieved context?	LLM is hallucinating facts not in documents — critical issue
Answer Relevancy	Does the answer actually address the user's question?	Answer is correct but off-topic — fails its core purpose
Context Precision	What fraction of retrieved chunks are actually useful?	Retriever returning noise — fix chunking/embeddings
Context Recall	Were all needed document sections retrieved?	Missing critical chunks — answer will be incomplete

💡 Key Point: Context Precision and Recall measure the RETRIEVER quality, not the LLM. If they are low, fix your chunking, embeddings, or reranking — not the language model.





8.4 DeepEval — Production Evaluation

DeepEval adds safety checks and GEval (LLM-as-a-judge with custom criteria written in plain English):

- **Hallucination:** score ≤ 0.3 acceptable. Checks for invented facts and internal contradictions.
- **Toxicity:** score ≤ 0.1 . Checks for harmful or offensive language in responses.
- **Bias:** score ≤ 0.2 . Detects demographic, political, or perspective bias.
- **GEval:** write evaluation criteria in plain English → LLM judge scores output 0.0–1.0. Use for Completeness, Fairness, Omission — properties with no built-in metric.

9. On-Premise LLMs & Ollama

9.1 Why Use On-Premise LLMs?

Benefit	Details
 Data Privacy	Prompts and data never leave your network. Critical for healthcare (HIPAA), finance (PCI-DSS), legal. Data sovereignty.
 Zero API Costs	After download, no per-token fees. High-volume usage (millions of queries) becomes dramatically cheaper.
 Full Control	Choose exact model version, fine-tune it, not affected by provider outages or policy changes.
 Offline / Air-Gapped	Works without internet. Essential for classified facilities, remote locations, edge devices.

⚠ Warning: On-premise requires hardware (8GB+ GPU VRAM for 7B models), setup complexity, and manual model management. Use cloud APIs for prototyping. Switch to on-premise for sensitive data or high volume.

9.2 Ollama — Run LLMs Locally

Ollama is an open-source tool that makes running LLMs locally as easy as installing software. It handles downloading weights, loading them into memory, and exposing a REST API — all with single commands.

Terminal — Install & Run Models with Ollama

```
# Install (macOS/Linux)
curl -fsSL https://ollama.com/install.sh | sh
# Windows/Mac: download from https://ollama.com/download

# Pull (download) models
ollama pull llama3.2           # Meta Llama 3.2 3B   (2.0 GB)
ollama pull mistral            # Mistral 7B       (4.1 GB)
ollama pull gemma3:4b          # Google Gemma 3 4B (3.3 GB)
ollama pull deepseek-r1:7b     # DeepSeek R1 7B   (4.7 GB)
ollama pull phi4               # Microsoft Phi-4 14B (9.1 GB)

# Run interactive chat
ollama run llama3.2
ollama list                    # show downloaded models
```

Python — Ollama with LangChain

```
from langchain_ollama import ChatOllama

chat = ChatOllama(model="llama3.2", temperature=0.7)
from langchain_core.messages import HumanMessage, SystemMessage
response = chat.invoke([
    SystemMessage(content="You are a helpful AI tutor."),
    HumanMessage(content="What is RAG?")
])
```

```
|print(response.content)
```

🔗 **Remember:** Model Library: <https://ollama.com/library> | Hugging Face GGUF models: <https://huggingface.co/models>

10. Fine-Tuning, PEFT, LoRA & QLoRA

10.1 RAG vs Fine-Tuning vs Re-training

Feature	RAG	Fine-Tuning	Re-training
Model Change	None. Weights static.	Partial. Some weights updated.	Complete. All weights from scratch.
Knowledge	External (vector DB)	Internal (model params)	Internal (model params)
Cost & Time	Low-Moderate ✓	Moderate-High ⚠	Extremely High ✗
Adaptability	Dynamic — update vector DB	Static — need re-tuning	Static — need re-training
Use RAG when	Knowledge changes frequently; need citations; large docs	Need specific output style, domain vocabulary, or format	Almost never — only well-funded AI labs

10.2 PEFT — Parameter Efficient Fine-Tuning

Full fine-tuning updates all weights — expensive and causes "catastrophic forgetting" (model forgets general language abilities). PEFT updates only a tiny fraction:

- **Freeze all original weights** ❄ — the base model's billions of parameters are locked.
- **Insert small trainable modules (adapters)** 🔥 — only these change during training.
- **Result:** saves enormous compute, time, and VRAM. Adapter file is MBs vs full model GBs.

Full Fine-Tuning	PEFT (e.g. LoRA)
Update all ~7B parameters	Update only ~0.1–1% of parameters
Requires 40–80GB VRAM	Runs on 8–16GB consumer GPU
Full model copy per task (GBs)	Adapter ~10–100MB — share the base model
Risk of catastrophic forgetting	Base model preserved; generalisation maintained

10.3 LoRA — Low-Rank Adaptation

LoRA injects trainable low-rank matrices (A and B) into frozen model layers. Only A and B are trained. The key math:

- Original weight matrix W is frozen. LoRA decomposes the update $\Delta W = A \times B$.
- A has dimensions $d \times r$; B has dimensions $r \times k$. Rank r is typically 4–16.
- Final weight at inference = Original $W + \Delta W$ (matrices merged back together).
- Example: 1000×1000 matrix $W = 1,000,000$ params → LoRA $r=8$: $1000 \times 8 + 8 \times 1000 = 16,000$ params → 98.4% reduction!

☑ **Key Insight:** LoRA = ~98% fewer trainable parameters, same quality, tiny adapter file. It has become the default method for fine-tuning open-source LLMs.

10.4 QLoRA — Quantized LoRA

Problem: even with LoRA, the base model in FP16 takes ~14GB VRAM for a 7B model. QLoRA quantizes the base model to 4-bit to shrink it, then applies LoRA adapters in FP16 on top.

Precision	Bytes per Weight	7B Model Size
FP16 (standard)	2 bytes	~14 GB — needs large GPU
4-bit NF4 (QLoRA)	0.5 bytes (4× smaller)	~4 GB — runs on consumer GPU! ☑

Real-world example: Llama 65B in FP16 = ~130 GB (needs multiple A100s). In 4-bit = ~33 GB (fits on one A100). QLoRA made fine-tuning 65B models accessible to individual researchers for the first time.

☑ **Key Insight:** QLoRA = LoRA + Quantization. Democratizes fine-tuning large models on limited infrastructure. Key insight: adapters stay in FP16 for training accuracy; only the frozen base model is quantized to 4-bit.

11. Guardrails for RAG Systems

11.1 What Are Guardrails?

Guardrails are safety and compliance mechanisms that wrap an LLM pipeline to control what goes in and what comes out. They are independent of the LLM — portable across model providers.

⚠ **Warning:** A RAG system without guardrails in production is like leaving a customer database unlocked. The LLM will follow instructions — including malicious ones — unless you explicitly stop it.

11.2 Three-Layer Defence Strategy

#	Layer	Where in Pipeline	Method	What It Catches
1	Input Guardrail	BEFORE vector DB retrieval	Keyword matching on user query	Explicit PII requests — "show me the SSN"
2	Prompt Hardening	At LLM invocation (system prompt)	System prompt privacy rules	LLM instructed to refuse revealing PII from context
3	Output Guardrail	AFTER LLM response generated	Regex pattern matching	Any PII that appeared in the LLM's output

11.3 Where to Add Guardrails in Your RAG Code

Python — Guardrail integration pattern

```
guardrail = PIIGuardrail() # initialise once at startup

def handle_query(user_query: str) -> str:

    # — GUARDRAIL 1: Input check — BEFORE retrieval
    is_pii, details = guardrail.contains_pii_request(user_query)
    if is_pii:
        return "Cannot process requests for personal information."
```

```
# Your existing RAG pipeline (unchanged)
result      = rag.ask_question(user_query)
raw_answer  = result["result"]

# — GUARDRAIL 2: Output filter — AFTER LLM generation
redaction   = guardrail.filter_response(raw_answer)
return redaction.redacted # always return safe version
```

11.4 PII Types & Regex Approach

PII Type	Example	Risk	Replacement Token
Social Security Number	123-45-6789	● Critical	[SSN REDACTED]
Phone Number	(555) 123-4567	● High	[PHONE REDACTED]
Email Address	john@company.com	● High	[EMAIL REDACTED]
Credit Card	4111-1111-1111-1111	● Critical	[CARD REDACTED]
Date of Birth	01/15/1985	● High	[DOB REDACTED]

☒ **Key Insight:** The output guardrail is your safety net. Even if a malicious query slips past the input check, the regex-based output filter ensures no PII values ever reach the user's screen.

12. Master Glossary — All Key Terms

Term	One-Line Definition
Token	Unit of text the LLM processes (~0.75 words). All LLM costs and context limits are in tokens.
Embedding	Dense vector (list of numbers) representing the meaning of text. Similar meanings = geometrically close.
Cosine Similarity	Angle-based similarity between two vectors. 1.0 = identical meaning; 0 = unrelated.
Transformer	Neural network architecture using self-attention. Foundation of all modern LLMs (GPT, BERT, Claude, Gemini).
Self-Attention	Mechanism where each word in a sequence directly weighs the importance of every other word simultaneously.
Context Window	Maximum tokens an LLM can process at once. Sets the "memory" limit per conversation.
Hallucination	LLM generating confident but incorrect information not supported by source evidence.
RAG	Retrieval-Augmented Generation — retrieve relevant doc chunks, inject as context, generate grounded answer.
Chunking	Splitting large documents into smaller, independently searchable pieces before embedding.
Vector Database	Specialised DB for storing and fast-searching high-dimensional vectors using ANN algorithms.
HNSW	Hierarchical Navigable Small World — the index algorithm used by vector DBs for fast ANN search.
iRAG	Incremental RAG — update only changed document chunks instead of full rebuilds.

BLEU	N-gram precision: fraction of generated n-grams in reference. Used for machine translation.
ROUGE	N-gram recall: fraction of reference n-grams in generated text. Used for summarisation.
BERTScore	Token-level semantic similarity using BERT embeddings — understands synonyms.
RAGAS	RAG evaluation framework: Faithfulness, Answer Relevancy, Context Precision, Context Recall.
Faithfulness	RAGAS metric: fraction of answer claims supported by retrieved context. Most critical metric.
DeepEval	LLM eval framework with safety checks (Hallucination, Toxicity, Bias) + GEval custom metrics.
GEval	LLM-as-a-judge: write criteria in plain English, judge LLM scores output 0.0–1.0.
On-Premise LLM	LLM running entirely on your own hardware. No cloud API, no data leaving your network.
Ollama	Open-source tool to download and run LLMs locally with a single command.
Fine-Tuning	Update a subset of model weights on task-specific labelled data to improve narrow performance.
PEFT	Parameter Efficient Fine-Tuning — train only a tiny fraction of weights, freeze the rest.
LoRA	Low-Rank Adaptation — inject trainable A×B matrices into frozen layers. ~98% fewer trainable params.
QLoRA	Quantized LoRA — quantize base model to 4-bit, apply LoRA adapters in FP16. Runs on consumer GPUs.
Quantization	Reduce bit-width of model weights (FP32→FP16→INT8→4-bit) to shrink memory footprint.
Guardrail	Safety mechanism controlling what goes into or comes out of an LLM pipeline.
PII	Personally Identifiable Information — any data that can identify an individual (SSN, email, phone).
Prompt Engineering	Designing and structuring prompts to reliably get the best output from an LLM.
Zero-Shot	Prompting without examples — just describe the task.
Few-Shot	Provide 2–5 worked examples before the actual task to anchor format and style.
Chain of Thought	Add "Let's think step by step." to trigger step-by-step reasoning before the final answer.
ReAct	Reason + Act loop — LLM reasons, calls a tool, observes result, reasons further. Basis of Agents.
Poetry	Python dependency manager — tracks exact package versions in poetry.lock for reproducible builds.