THE ARCHITECTURE FIELD GUIDE

GenAI Architecture Field Guide

A technical reference for how generative-AI systems are actually built — every layer, every tool, every pattern. A 9-layer stack with worked examples, 15+ tool comparisons by use-case fit, production RAG blueprints with evaluation templates, and the GSDC roadmap from foundations to production.

WHAT'S INSIDE

9	25+	3	9
ARCHITECTURE LAYERS	TOOLS COMPARED	RAG BLUEPRINTS	MODULES MAPPED

A technical companion to GSDC's expert-tools resources. The GenAI tooling landscape moves quickly; tool names and capabilities reflect early-2026 reporting and should be re-verified before production decisions. Throughout, one running example — a **customer-support assistant** — shows how each layer contributes.

START HERE

The mental model: a stack, not a tool

Beginners ask “which AI tool should I use?”. Architects ask “what does each layer of my system need to do, and what's the best tool for that layer?”. This guide teaches the second question — the one that scales to production.

PRINCIPLE 1

Separate the layers

A model, a retriever, an orchestrator and a guardrail are different jobs. Mixing them into “the AI” is what makes systems impossible to debug or improve.

PRINCIPLE 2

Data quality dominates

A great framework over poorly prepared data still fails. Most RAG problems are data and retrieval problems, not model problems.

PRINCIPLE 3

Evaluate everything

If you can't measure retrieval and generation quality, you can't improve them. Evaluation is a first-class layer, not an afterthought.

PRINCIPLE 4

Compose, don't marry

The mature 2026 pattern mixes best-of-breed tools per layer — e.g. one library for retrieval, another for agent orchestration.

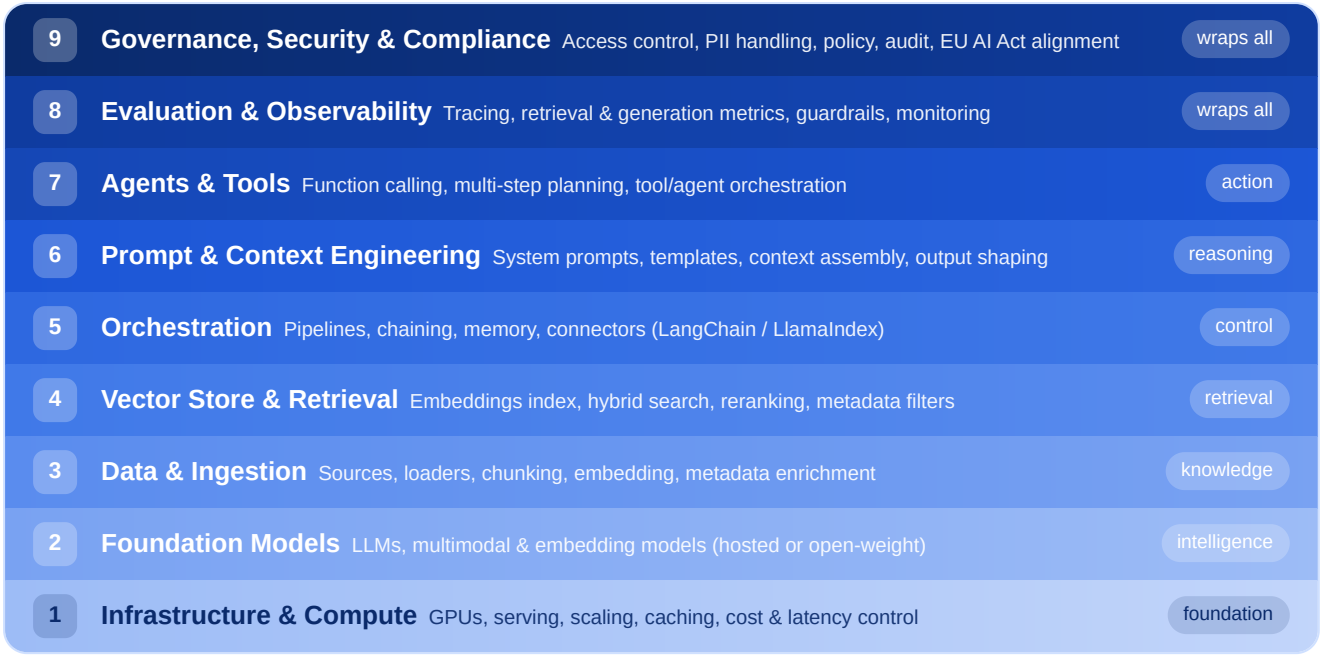
HOW TO READ THIS GUIDE

Page 3 shows the whole stack on one diagram. The pages after walk each layer with a worked example from our customer-support assistant. Then come the tool comparisons, RAG blueprints, evaluation templates, and the path to production.

01 · THE STACK

The 9-layer GenAI architecture

Read it bottom-up (what it's built on) or top-down (what governs it). A user request flows down into retrieval and back up through generation; governance and evaluation wrap the whole stack.



↓ Request flows down into retrieval

Generation flows back up ↑

Layers 8–9 are cross-cutting: they observe and govern every other layer rather than sitting in the request path. Most “it works in a demo but not in production” failures live in layers 3, 4, and 8.

01 · LAYER WALK-THROUGH

Layers 1–2 · the base

1

FOUNDATION · INFRASTRUCTURE & COMPUTE

Where it runs, and what it costs

Serving, scaling, caching, and the cost/latency budget. Architectural choices here — hosted API vs. self-hosted, caching layers, model size — quietly decide whether your system is affordable and fast enough to ship.

> SUPPORT ASSISTANT
>
> We start on a hosted model API to move fast, add a response cache for repeat questions, and set a per-conversation token budget so costs stay predictable as traffic grows.

2

INTELLIGENCE · FOUNDATION MODELS

The reasoning engine

The LLM (and embedding/multimodal models) that generate and understand. Choose on capability, context length, latency, cost and whether you need open weights for data control — not on brand familiarity.

> SUPPORT ASSISTANT
>
> A capable hosted LLM handles answers; a separate embedding model powers retrieval. We keep the model behind an interface so we can swap providers without rewriting the app.

01 · LAYER WALK-THROUGH

Layers 3–4 · the knowledge path

3

KNOWLEDGE · DATA & INGESTION

Turning documents into searchable meaning

Loaders pull in sources; chunking splits them sensibly; an embedding model turns each chunk into a vector; metadata is attached for filtering. This layer decides retrieval quality more than any model choice — garbage in, hallucination out.

> **SUPPORT ASSISTANT**
>
> Help-centre articles are chunked by section (not arbitrary length), tagged with product and version metadata, embedded, and stored. Bad chunking here would surface the wrong paragraph later.

4

RETRIEVAL · VECTOR STORE & SEARCH

Finding the right context fast

A vector database serves similarity search at scale. Production systems combine vector search with keyword (BM25) search, apply metadata filters, and add a reranker to push the best chunks to the top — the jump from “okay” to “reliable.”

> **SUPPORT ASSISTANT**
>
> A query retrieves the top candidates by vector + keyword, filters to the user's product version, then reranks so the single most relevant passage leads the context.

ARCHITECT'S NOTE

Naive retrieval (embed, grab top-k, stuff into the prompt) tends to plateau around 70–80% precision. Hybrid search + reranking + clean chunking is the practical production baseline.

01 · LAYER WALK-THROUGH

Layers 5–6 · control & reasoning

5

CONTROL · ORCHESTRATION

Wiring the pieces together

The orchestration layer assembles the pipeline: load → retrieve → build prompt → call model → post-process, plus memory and connectors. Frameworks here save you from hand-coding the plumbing and give you observability hooks.

> **SUPPORT ASSISTANT**
> An orchestration framework runs the retrieve-then-generate flow, manages conversation memory, and exposes each step for tracing so we can see exactly where an answer came from.

6

REASONING · PROMPT & CONTEXT ENGINEERING

Telling the model how to behave

System prompts set role and rules; templates assemble retrieved context with the question; output formats are specified. This is where grounding is enforced — e.g. “answer only from the context; cite sources; say you don't know if absent.”

> **SUPPORT ASSISTANT**
> The system prompt instructs the model to answer only from retrieved articles, cite the article title, and escalate to a human when confidence is low.

ARCHITECT'S NOTE

Layers 5 and 6 are where most quality lives once retrieval is solid: good orchestration makes systems debuggable, and disciplined prompting turns retrieved facts into trustworthy, cited answers.

01 · LAYER WALK-THROUGH

Layers 7–9 · action, evaluation & trust

7

ACTION · AGENTS & TOOLS

When the system needs to *do*, not just answer

Tools (search, APIs, calculators) and agent logic let the system take multi-step actions. Add this only when a plain retrieve-and-answer flow isn't enough — agents add power and failure modes in equal measure.

> **SUPPORT ASSISTANT**
> For “where is my order?”, an agent calls the order-status API, reads the result, and composes a grounded reply — retrieval alone couldn't answer it.

8

CROSS-CUTTING · EVALUATION & OBSERVABILITY

Knowing whether it actually works

Tracing every step, scoring retrieval and generation quality, and guardrails on inputs/outputs. Without this layer you're flying blind; with it, you can improve deliberately.

> **SUPPORT ASSISTANT**
> Every answer is logged with its retrieved context and scored for groundedness; a weekly eval set catches regressions before users do.

9

CROSS-CUTTING · GOVERNANCE & SECURITY

Safe, compliant, auditable

Access control, PII handling, content policy, and an audit trail — mapped to regulation such as the EU AI Act. The layer that lets an organisation actually deploy.

> **SUPPORT ASSISTANT**
> Customer data is masked before it reaches the model, every interaction is logged for audit, and a policy blocks out-of-scope requests.

02 · TOOL COMPARISONS

Layer 2 · foundation & embedding models

Match the model to the job, not the hype. Hosted models lead on convenience and capability; open-weight models lead on control, customisation and data residency.

MODEL FAMILY	PROVIDER	WEIGHTS	STRONG-FIT USE CASE
GPT	OpenAI	Hosted	Broad general-purpose, largest ecosystem
Claude	Anthropic	Hosted	Long-context work, careful reasoning, large documents
Gemini	Google	Hosted	Multimodal tasks; Google Cloud / Workspace shops
Llama	Meta	Open	Self-hosting, fine-tuning, strict data control
Mistral	Mistral AI	Open	Efficient open models, on-prem & cost-sensitive

Embedding models (power retrieval — Layer 3/4)

OPTION	TYPE	STRONG-FIT USE CASE
OpenAI embeddings	Hosted	Easy default, good quality, no infra
Cohere Embed	Hosted	Multilingual retrieval, rerank pairing
BGE / E5 (open)	Open	Self-hosted retrieval, cost & data control

Keep models behind an interface so you can swap providers as capabilities and prices shift — the landscape changes monthly. Names current as of early 2026.

02 · TOOL COMPARISONS

Layer 4 · vector stores

The retrieval backbone. Pick on operational model (managed vs. self-hosted), scale, and whether you want vectors alongside existing infrastructure.

VECTOR STORE	MODEL	STRONG-FIT USE CASE
Pinecone	Managed	Fastest path to production, zero-ops
Weaviate	OSS / managed	Hybrid search, modular, flexible schema
Qdrant	OSS / managed	Cost-effective scaling, strong filtering
Milvus	OSS	Very large scale, GPU-accelerated options
Chroma	OSS (embedded)	Prototyping & local development
pgvector	Postgres extension	Add vectors to an existing Postgres DB
Vespa	OSS / managed	Vector + BM25 + ML ranking at billions of docs

DECISION SHORTCUT

Prototyping → **Chroma**. Already on Postgres → **pgvector**. Want managed and fast → **Pinecone**. Hybrid search & flexibility → **Weaviate/Qdrant**. Web-scale low latency → **Vespa/Milvus**.

02 · TOOL COMPARISONS

Layers 5 & 8 · orchestration, managed RAG, eval

Orchestration frameworks (Layer 5)

FRAMEWORK	STRONG-FIT USE CASE
LangChain	Largest ecosystem (500+ integrations); agents via LangGraph
LlamaIndex	Retrieval-first; ~92% retrieval accuracy; 160+ data connectors
Haystack	Structured, audit-friendly pipelines for regulated domains
Semantic Kernel	Microsoft / .NET enterprise environments
DSPy	Programmatic prompt optimisation; research & tuning

Managed / cloud-native RAG & evaluation

TOOL	LAYER	STRONG-FIT USE CASE
Vectara / Ragie	Managed RAG	Fastest time-to-value, least pipeline control
Bedrock KB / Azure AI Search / Vertex AI Search	Cloud-native	Zero-ops RAG where your data already lives
LangSmith / Phoenix	Observability	Tracing & debugging pipelines
Ragas / TruLens / DeepEval	Evaluation	Scoring retrieval & generation quality

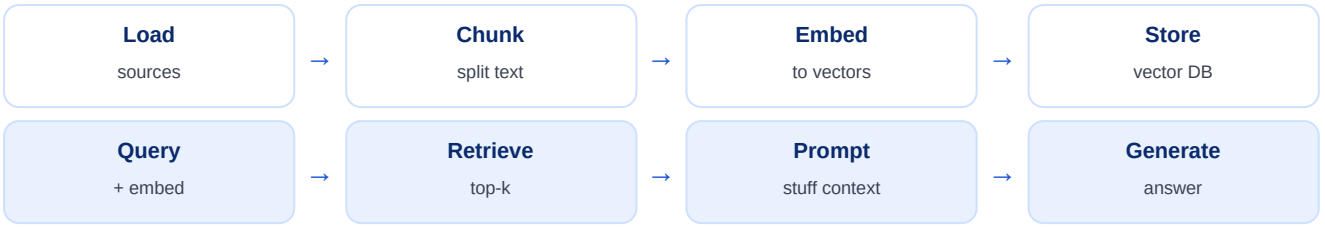
The mature 2026 pattern composes layers: e.g. LlamaIndex for retrieval, LangGraph for agents, Ragas for eval. Choosing one layer's tool doesn't lock the others.

03 · RAG BLUEPRINTS

From naive to production RAG

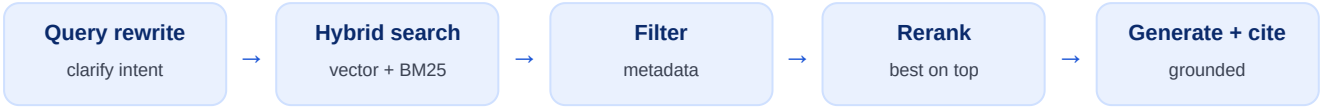
Blueprint 1 · Naive RAG (the baseline)

Indexing (offline), then query (online). Good enough for small, stable document sets.



Blueprint 2 · Production RAG (the baseline that ships)

Adds the steps that lift precision past the naive plateau and make answers trustworthy.



THE DIFFERENCE THAT MATTERS

Hybrid retrieval, metadata filtering and a reranker are what separate a convincing demo from a system you'd trust with customers — plus citations so answers are checkable.

03 · RAG BLUEPRINTS

Agentic RAG & the two failure modes

Blueprint 3 · Agentic RAG

A router decides how to answer; the system can retrieve multiple times, call tools, and self-check before responding.

Route classify query → **Plan** steps / tools → **Retrieve / call** loop as needed → **Synthesise** combine → **Self-check** verify

Use agentic RAG when queries vary widely or require actions and multi-step reasoning — not for simple lookups, where it adds cost and failure surface.

RAG fails in exactly two places

FAILURE A

Bad retrieval

The wrong chunks reach the model. Causes: poor chunking, weak embeddings, no reranking, missing metadata filters. **Fix the data & retrieval layers (3–4).**

FAILURE B

Bad generation

Good chunks, bad answer — hallucination or ignoring context. Causes: weak prompting, no grounding instruction, no citations. **Fix the prompt & eval layers (6, 8).**

DIAGNOSTIC HABIT

When an answer is wrong, first ask: *were the right chunks retrieved?* If no → it's a retrieval problem. If yes → it's a generation problem. This one question routes almost every debugging session.

04 · EVALUATION TEMPLATES

Measure retrieval and generation separately

Because RAG fails in two places, you evaluate in two places. Track these metrics on a fixed evaluation set and re-run them on every change.

Retrieval metrics (did we fetch the right context?)

METRIC	WHAT IT MEASURES	AIM FOR
Context Recall	Did retrieval include the needed info?	High
Context Precision	How much retrieved context is relevant?	High
Recall@k / MRR	Is the right chunk in the top results?	High, low k

Generation metrics (did we answer well from it?)

METRIC	WHAT IT MEASURES	AIM FOR
Faithfulness / Groundedness	Is the answer supported by the context?	High
Answer Relevance	Does it actually address the question?	High
Hallucination rate	Claims not supported by any source	Low

EVAL TEMPLATE IN PRACTICE

Keep a set of ~50 real questions with known-good answers. On every pipeline change, re-score retrieval and generation metrics and compare. A change that helps one and hurts another is the signal to investigate.

PATTERNS

Architecture decisions, made simple

A few recurring decisions shape most GenAI systems. Here are clear defaults — with the conditions that change them.

DECISION	DEFAULT	SWITCH WHEN…
Prompt vs. RAG vs. fine-tune	Start with prompting; add RAG for knowledge	Fine-tune only for style/format at scale, not facts
Hosted vs. open-weight model	Hosted to move fast	Switch to open for data residency / heavy customisation
Build vs. buy retrieval	Managed RAG to start	Build when you need fine control or scale
Plain RAG vs. agentic	Plain retrieve-and-answer	Go agentic for actions & multi-step reasoning
One framework vs. compose	Compose best-of-breed per layer	Single framework only for simple, stable apps

RULE OF THUMB

RAG for facts, fine-tune for form

Use retrieval to give the model knowledge; use fine-tuning to shape *how* it responds — rarely to teach it new facts.

RULE OF THUMB

Add complexity only on evidence

Each layer you add (agents, fine-tuning) is more to break. Add it when your evaluation set proves the simpler version isn't enough.

05 · MODULE MAPPING

Each module to its architecture layer

The GSDC certification is structured to build the stack from the ground up. Here's how the nine modules line up with the nine layers in this guide.

SYLLABUS MODULE	BUILDS WHICH LAYER(S)
M1 · Foundations of Generative AI	The whole-stack mental model
M2 · How LLMs Work	Layer 2 · Foundation models
M3 · Prompt Engineering	Layer 6 · Prompt & context
M4 · Tools & Multimodal	Layers 2 & 5 · models & orchestration tools
M5 · Building: RAG & Grounding	Layers 3 & 4 · data & retrieval
M6 · Automation & Agents	Layer 7 · Agents & tools
M7 · Evaluating AI Output	Layer 8 · Evaluation & observability
M8 · Governance & Responsible AI	Layer 9 · Governance & security
M9 · Capstone & Implementation	Layer 1 + integrating the full stack

06 · THE ROADMAP

From foundations to production

A clear progression: understand the pieces, build a grounded system, make it act, prove it works, and ship it safely. Each stage adds layers of the stack.

1. **Foundations · Modules 1–2**
   Understand models and the stack. Outcome: you can reason about any GenAI system's layers.
2. **Prompting & tools · Modules 3–4**
   Control models and choose tools per layer. Outcome: reliable single-call applications.
3. **Build & ground · Module 5**
   Add data, retrieval and a production RAG pipeline. Outcome: grounded, cited answers.
4. **Automate · Module 6**
   Add agents and tools where needed. Outcome: systems that act, not just answer.
5. **Evaluate & govern · Modules 7–8**
   Measure quality and add governance. Outcome: trustworthy, compliant systems.
6. **Ship · Module 9**
   Integrate the full stack in a capstone. Outcome: a production-shaped project and a credential.

06 · PRODUCTION

The production-readiness checklist

A demo proves it can work once; production proves it works reliably, affordably and safely. Before you ship, check every layer.

LAYERS 1–2

Cost & latency

Token budgets, caching, right-sized models, fallback on provider outage.

LAYERS 3–4

Retrieval quality

Clean chunking, hybrid search, reranking, metadata filters, refresh strategy.

LAYERS 5–6

Grounding & control

Citations, “don't know” behaviour, output formatting, prompt versioning.

LAYER 7

Safe actions

Tool permissions, human-in-the-loop for irreversible steps, timeouts.

LAYER 8

Observability

Tracing, an eval set, regression alerts, user-feedback capture.

LAYER 9

Governance

PII handling, access control, audit log, policy & regulatory alignment.

CAREER

Architecture skills the market pays for

The roles that command premiums are the ones that can reason across the whole stack — not just call an API. Here's where these skills map to in-demand roles.

STACK SKILL	MAPS TO ROLES	WHY IT PAYS
RAG & retrieval design	GenAI Engineer, Solutions Architect	Powers most real products
Agents & orchestration	Agentic AI Developer	Carries a 15–20% premium
Evaluation & observability	Senior / Lead Engineer	Marker of production maturity
Governance & security	AI Governance Specialist	Fastest-growing, premium track
Cost & latency engineering	AI Platform / MLOps Engineer	Directly controls spend

THE HIRING SIGNAL

Employers increasingly screen for people who can take a system from prototype to production — exactly the span this guide covers. A portfolio that shows a working, evaluated, governed pipeline is worth more than any single tool on a CV. AI skills broadly carry a substantial wage premium, and architecture-level skill sits at the top of that curve.

Role and pay context is drawn from public 2025–26 labour-market reporting; ranges vary by region, employer and experience.

RECAP

The stack on one page

Your quick-reference: the nine layers, the production baseline, and the words you'll use most.

THE 9 LAYERS

1 Infra · **2** Models · **3** Data/ingestion · **4** Vector store/retrieval · **5** Orchestration · **6** Prompt/context · **7** Agents/tools · **8** Evaluation/observability · **9** Governance/security.

Chunking

Splitting documents into retrievable pieces — ideally by meaning, not fixed length.

Embedding

Turning text into a vector so similar meaning sits close together.

Hybrid search

Combining vector similarity with keyword (BM25) search.

Reranking

A second pass that orders retrieved chunks by true relevance.

Grounding

Forcing answers to come from retrieved context, with citations.

Faithfulness

Whether an answer is actually supported by its sources.

THE ONE-LINE TAKEAWAY

Good GenAI systems are **well-composed stacks**: clean data, strong retrieval, disciplined prompting, real evaluation, and governance around it all — with the model as just one layer.

WHERE TO GO NEXT

You can read the stack. Now build it.

You've seen all nine layers, the tools at each, the blueprints that ship, and how to prove a system works. The gap between understanding architecture and being trusted to build it is hands-on practice — and a credential that says you can.

UNDERSTAND
The nine layers and how they compose.

BUILD
A grounded, evaluated, governed pipeline.

PROVE
It with a portfolio and a credential.

The bottom line

Anyone can call a model. The people the market values can architect a system around it — retrieval that's reliable, evaluation that's honest, governance that's airtight. This guide mapped that system; the certification is how you learn to build it end-to-end.