Modern AI Systems Engineer Fieldbook

Modern AI Systems Engineer Fieldbook

> A current implementation guide to Spring AI 2, semantic search, RAG, embeddings, Ollama, OpenAI, Gemini, agents, fine-tuning, evaluation, security, and production operations.

\newpage

00. The modern AI systems roadmap

An AI-powered system is ordinary software around a probabilistic model. The model predicts useful output; your application owns data, tools, permissions, state, validation, cost, latency, evidence, and recovery.

Current baseline, August 2026

Layer	Current production baseline	Important note
Spring AI	2.0.0 GA	Spring Boot 4.0/4.1 and Spring Framework 7
Java	21+	Use the version supported by your Boot line
OpenAI	Responses API; GPT-5.6 family	Route Sol/Terra/Luna by workload role
Gemini	Gemini 3.6 Flash; 3.5 Flash-Lite	Stable model IDs can still deprecate later
Ollama	0.32.5	Local runtime; capabilities depend on the chosen model
Google embeddings	`gemini-embedding-2`	GA multimodal embeddings, 128–3072 dimensions
OpenAI embeddings	`text-embedding-3-small/large`	1536/3072 default dimensions
Local embeddings	`embeddinggemma`, `qwen3-embedding`	Keep index and query model identical
Vector search	pgvector HNSW or managed equivalent	Measure recall, latency, filters, and operations
Agent integration	Tool calling and MCP	Tools need authorization, schemas, budgets, and audit

“Latest” means the newest compatible, stable set that fits quality, latency, cost, privacy, and lifecycle requirements. A flagship model is not the default for every classification or extraction request.

The seven layers

```
experience → orchestration → model gateway → retrieval/tools → enterprise systems →
safety/evaluation → operations
```

Spring AI sits mainly in orchestration and provider integration. Ollama, OpenAI, and Gemini are inference providers. A vector database is retrieval infrastructure. None of them alone creates a trustworthy product.

Build sequence

1. Define the user decision or job.
2. Create an evaluation dataset before choosing architecture.
3. Establish a simple prompt baseline.
4. Add structured output when software consumes the result.

5. Add RAG only when external knowledge is needed.
6. Add tools only when the system must act or obtain live state.
7. Add an agent loop only when a fixed workflow cannot express the task.
8. Fine-tune only after measured prompt/RAG/tool limitations.

Feynman check

Explain the system as a librarian, a writer, and a careful operator. Retrieval finds evidence; the model writes; application code checks and acts.

\newpage

01. Tokens, transformers, context, and uncertainty

A language model receives tokens and predicts a probability distribution for the next token. Repeating that operation produces text, JSON, tool calls, code, or multimodal descriptions. It does not query a hidden database of guaranteed facts.

Tokens and context

Tokens are pieces of text, not words. Input instructions, retrieved documents, tool schemas, history, images, and output all consume context and money. Context capacity is not memory quality: filling a large window can reduce signal, increase latency, and expose unnecessary data.

Parameters and inference

Model parameters contain learned statistical patterns. Inference uses fixed weights to produce an answer. Fine-tuning changes some weights. RAG changes the input context without changing weights. Tool calling lets the model request deterministic work from software.

Sampling

Sampling settings alter output diversity but do not repair missing evidence. Current provider families increasingly manage sampling and reasoning internally. Do not copy temperature defaults between providers. Establish a task-specific baseline and evaluate it.

Hallucination

“Hallucination” means unsupported or fabricated content. Reduce it with:

- authoritative retrieval and citations;
- constrained structured output;
- tools for current facts and calculations;
- explicit abstention when evidence is insufficient;
- claim-level groundedness tests;
- authorization and deterministic validation outside the model.

Reasoning

More reasoning effort can improve difficult tasks while raising latency and cost. It cannot make an impossible or under-specified task valid. Route by task: cheap models for bounded extraction; balanced models for most interactions; flagship reasoning for complex, high-value decisions with strong evaluation.

Feynman check

The model is a brilliant sentence completer with learned patterns. Explain why that can create both an excellent analyst and a confident mistake.

\newpage

02. Spring AI 2.0 architecture and project setup

Spring AI 2.0 provides portable Java APIs for chat, embeddings, images, audio, vector stores, tools, memory, RAG, MCP, evaluation, and observability. It uses Spring Boot 4.x and Spring Framework 7.

Dependency management

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.ai</groupId>
      <artifactId>spring-ai-bom</artifactId>
      <version>2.0.0</version>
      <type>pom</type><scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Add only the provider and vector-store starters the service actually uses:

```
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-starter-model-openai</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.ai</groupId>
```

```
  <artifactId>spring-ai-starter-vector-store-pgvector</artifactId>
</dependency>
```

Spring AI 2.0 focuses its core providers on OpenAI, Anthropic, Bedrock, Google GenAI, Mistral, DeepSeek, and Ollama. It prefers vendor SDK integrations so new provider capabilities can arrive without pretending every provider is equal.

API levels

- `ChatClient`: normal fluent application API;
- `ChatModel`: lower-level model abstraction;
- Advisors: reusable interception/orchestration such as memory, RAG, logging, tool execution, and semantic cache;
- provider options: escape hatch for capabilities that are not portable.

Configuration ownership

Keep secrets outside source. Bind provider endpoints, model roles, timeouts, and budgets through validated configuration. Fail startup when a required provider is absent; allow an explicitly designed degraded mode otherwise.

Upgrade warning

Spring AI 2.0 includes breaking starter names, immutable option builders, Jackson 3, JSpecify null-safety, and tool-calling lifecycle changes. Do not paste 1.x examples into a 2.0 application without the upgrade notes.

Feynman check

Spring AI is an adapter and orchestration toolbox. Explain what business rules, data permissions, and reliability work still belong to your application.

\newpage

03. ChatClient, prompts, streaming, and structured output

`ChatClient` is the normal application boundary. Create role-specific clients rather than one global client with every advisor and tool.

```
record Answer(String summary, List<String> citations, double confidence) {}

Answer answer = chatClient.prompt()
```

```
    .system(s -> s.text(SUPPORT_POLICY).param("region", region))
    .user(question)
    .call()
    .entity(Answer.class);
```

Prompt contract

An effective prompt states:

- user-visible outcome;
- evidence and business constraints;
- available tools and when to use them;
- output schema;
- stopping and abstention conditions.

Keep system/developer policy stable and put changing user data later. Avoid contradictory absolute rules and repeated examples. Treat prompt templates as versioned code with tests and ownership.

Structured output

Use schema-constrained provider output where supported and validate the decoded Java record. Spring AI 2.0 supports provider-native structured output and self-correcting schema validation. A valid JSON shape can still contain an invalid business decision, so run domain validation afterward.

Streaming

Streaming improves perceived latency but complicates moderation, citations, tool loops, cancellation, retries, and UI recovery. Never repeat already displayed tokens after reconnect. A streamed partial sentence is not a committed business result.

Timeouts and cancellation

Set connect, read, overall-request, and tool-loop deadlines. Propagate client cancellation. Do not automatically retry a request after a tool may have performed a non-idempotent action.

Feynman check

The prompt is a job contract; structured output is a form; validation is the supervisor checking the filled form before software trusts it.

\newpage

04. Provider abstraction, model routing, and fallbacks

Provider portability is valuable at stable concepts—messages, embeddings, tools, and vector stores—but model behavior is not interchangeable.

Route by role

Define configuration roles instead of scattering model IDs:

```yaml
ai:
  roles:
    complex-reasoning: openai:gpt-5.6-sol
    interaction: google:gemini-3.6-flash
    high-volume: google:gemini-3.5-flash-lite
    private-draft: ollama:qwen3
    embedding: google:gemini-embedding-2
```

Every role needs measured quality, p50/p95 latency, cost, context needs, regional availability, tool/JSON support, and lifecycle status.

Fallback semantics

A fallback is safe only if it preserves:

- data residency and retention rules;
- prompt and output contracts;
- tool schemas and authorization;
- embedding dimensions and vector-space identity;
- multimodal formats;
- acceptable quality and latency.

Do not catch every provider error and silently send confidential data to another vendor. Circuit breakers protect capacity; they do not define product correctness.

Capability registry

Store model/provider capabilities in configuration: reasoning, streaming, structured output, image/audio input, tools, parallel tools, context limits, fine-tuning, lifecycle date, and approved data classes. Test the registry against live provider smoke calls.

Shadow and canary

Shadow requests can compare a candidate without changing user output, but they double data exposure and cost. Redact, sample, and obtain policy approval. Canary a new model by role and tenant; compare the same evaluation slice.

Feynman check

A provider adapter makes plugs look similar. It does not make the electricity, price, or appliance behavior identical.

\newpage

05. OpenAI Responses API, GPT-5.6, tools, and embeddings

The Responses API is the current unified OpenAI surface for stateful, multimodal, tool-using applications. Current model guidance identifies the GPT-5.6 family: Sol for flagship quality, Terra for balanced work, and Luna for high-volume/latency-sensitive roles.

Request shape

```
String result = openAiClient.prompt()
    .system("Answer from approved evidence. Cite source ids.")
    .user(question)
    .tools(new AccountTools(accountService))
    .call()
    .content();
```

Use Spring AI for portable application flow and the provider SDK/API when a Responses capability is not yet abstracted.

Hosted and custom tools

OpenAI supports web search, file search, function tools, remote MCP, tool search, code execution, and other hosted capabilities. Expose only relevant tools. Schemas must declare required fields and reject extra properties. Application code authorizes every requested call.

Model migration

Do not replace every old model with Sol. Preserve each route's quality, latency, cost, reasoning, endpoint, cache, and tool semantics. Keep model registries, allowlists, pricing metadata, tests, and UI labels aligned.

Embeddings

`text-embedding-3-small` defaults to 1536 dimensions and `text-embedding-3-large` to 3072. The `dimensions` parameter can shorten them. Changing model or dimensions creates a new vector space and requires re-embedding plus index migration.

State and privacy

Decide explicitly whether the API or your application stores conversation state. Do not place secrets in prompts. Treat provider request IDs, token use, tool calls, citations, and refusal/abstention as observable events.

Feynman check

OpenAI provides models and tools. Your application decides which employee may ask which model to use which tool on which customer account.

\newpage

06. Gemini 3.x, Google GenAI, multimodality, and grounding

The current stable Gemini family includes `gemini-3.6-flash` for balanced agentic/multimodal work and `gemini-3.5-flash-lite` for low-cost high-volume automation. Preview IDs are not stable production dependencies.

Spring AI integration

Spring AI 2.0 uses the Google GenAI SDK integration:

```
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-starter-model-google-genai</artifactId>
</dependency>
```

Use provider options for thinking, multimodal inputs, native tools, grounding, and features not represented by common `ChatOptions`.

Tools and structured output

Gemini can use managed Google Search, Maps, URL context, File Search, and code execution, plus custom function calls. Function calling requests an intermediate action; structured output constrains the final

answer. Preserve all required model response parts and tool IDs across a manual loop.

Embedding 2

`gemini-embedding-2` is a GA multimodal embedding model for text, images, video, audio, and PDF. It accepts up to 8,192 input tokens and produces 128–3072 dimensions; 768, 1536, and 3072 are recommended. Truncated vectors are automatically normalized.

Tuning reality

Gemini API model tuning is not currently supported: the last tuning model was shut down in May 2025. Use prompts, RAG, tools, or another approved training path; do not design around a dead endpoint.

Lifecycle discipline

Read the Gemini changelog and deprecation page automatically in release management. Model aliases, parameters, and preview capabilities can change.

Feynman check

Gemini is a multimodal engine with Google tools. Explain why a stable model ID, an embedding model, and a managed research agent are three different products.

\newpage

07. Ollama 0.32, local models, operations, and privacy

Ollama runs supported open models locally or through configured services. The current release baseline is 0.32.5. Its API provides chat, generate, embed, model management, tool calling, structured output, and OpenAI-compatible routes.

Spring integration

```
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-starter-model-ollama</artifactId>
</dependency>
```

```
spring.ai.ollama:
  base-url: http://ollama:11434
```

```
  init:
    pull-model-strategy: never
```

Pin model names/digests in production. Pull models during an explicit provisioning stage, not every application startup.

Capacity

Model size, quantization, context length, concurrent requests, GPU/CPU memory, KV cache, batch size, and keep-alive determine latency and throughput. A local endpoint is not free: hardware, power, model storage, warm-up, and operations are costs.

Capabilities belong to models

Ollama may expose a tools or JSON API, but the selected model must perform the behavior well. Test schema adherence, parallel tools, long context, multilingual quality, and prompt injection for each model and quantization.

Security

Do not expose port 11434 publicly. Put authentication, authorization, rate limits, request size, and tenant isolation in a gateway. Local inference reduces external data transfer; it does not automatically provide encryption, auditing, deletion, or safe model provenance.

Feynman check

Ollama is a local model kitchen. The menu says what can be served; your hardware determines how fast; your application still checks who may order.

\newpage

08. Embeddings, similarity, dimensions, and vector identity

An embedding turns content into a numeric vector whose direction represents learned semantic relationships. Similar meanings tend to be near each other in that model's vector space.

Similarity

Cosine similarity compares direction:

```
cos(a,b) = (a · b) / (||a|| ||b||)
```

Dot product is equivalent for unit-normalized vectors. Euclidean distance can work with the index and normalization chosen. Never compare raw threshold numbers across different models or distance functions.

Vector identity

Store with every vector:

- embedding provider, exact model, revision, and dimensions;
- normalization behavior;
- source document/version and chunker version;
- content hash, tenant, language, and authorization metadata;
- creation time and ingestion run.

Index and query content must use the same embedding space. Changing dimensions or model requires a parallel index, re-embedding, evaluation, traffic switch, and rollback window.

Dimension trade-off

More dimensions may improve nuance but increase storage, memory, network, and index cost. Benchmark the actual corpus at 768/1536/3072 or model-appropriate sizes. “Largest” is not a universal quality decision.

Multimodal spaces

Gemini Embedding 2 can map text, images, audio, video, and PDFs into one space. Evaluate cross-modal retrieval separately: text-to-image recall is a different task from text-to-text question answering.

Privacy

Embeddings can leak membership or meaning and remain personal data when tied to people. Apply retention, encryption, access control, deletion, and tenant isolation to vectors and source content.

Feynman check

An embedding is a map coordinate for meaning. Coordinates from two different mapmakers cannot be mixed and expected to point to the same street.

\newpage

09. Semantic, lexical, hybrid search, and

reranking

Semantic search retrieves meaning, lexical search retrieves exact terms, and hybrid search combines both. Production search usually needs all three plus metadata filtering and reranking.

Retrieval pipeline

```
query → normalize → intent/tenant filter → lexical candidates + vector candidates → fuse → rerank → threshold → diversify → evidence
```

Vector search is strong for paraphrases. BM25/keyword search is strong for error codes, product IDs, names, dates, and rare exact terms. Reciprocal rank fusion combines rankings without pretending incomparable raw scores match.

Reranking

A cross-encoder or capable reranking model scores query-document pairs more precisely after broad retrieval. Retrieve perhaps 30–100 candidates, rerank, then send a small evidence set to generation. Measure added latency and cost.

Filters and authorization

Apply tenant, region, document status, effective date, and permission filters inside retrieval—not after the model sees results. Metadata is part of the security boundary.

Evaluation

Use labeled queries with relevant document IDs. Measure recall@k, precision@k, MRR, nDCG, no-result rate, latency, and permission violations. Then measure answer groundedness separately. A fluent answer cannot prove retrieval worked.

Thresholds

A similarity threshold is a rejection policy, not a magic truth line. Tune per embedding model and corpus. If no candidate qualifies, ask a narrower question or abstain rather than filling context with noise.

Feynman check

Keyword search finds the same label; semantic search finds the same idea; reranking is a careful second reader choosing which results deserve attention.

\newpage

10. Vector stores, pgvector, HNSW, schemas, and migration

Spring AI's `VectorStore` API supports pgvector, Redis, Qdrant, Pinecone, Weaviate, Oracle, S3, and other implementations. Portability covers basic operations; indexing, filtering, consistency, scaling, and backups remain store-specific.

pgvector baseline

```yaml
spring.ai.vectorstore.pgvector:
  index-type: HNSW
  distance-type: COSINE_DISTANCE
  dimensions: 1536
  initialize-schema: false
  schema-validation: true
```

Schema initialization is opt-in. Production migrations should create extensions, tables, vector dimensions, metadata indexes, and HNSW/IVFFlat indexes through normal database change control.

HNSW and IVFFlat

HNSW offers strong query speed/recall with higher memory and build cost. IVFFlat requires representative data and tuning of lists/probes. Exact search is useful for small corpora and evaluation truth. Approximate nearest-neighbor parameters trade recall for latency.

Multi-tenancy

Choose per-tenant database/schema/table/partition or shared rows with mandatory filters from threat and scale requirements. Do not trust model-generated filters. Construct filters from authenticated application context.

Migration

Use versioned collections/tables:

1. create the new index;
2. backfill from immutable source;
3. verify counts and hashes;
4. run offline retrieval evals;
5. dual-read/shadow;

6. switch an alias;
7. retain rollback;
8. delete only after the retention window.

Operations

Back up source and metadata, not only vectors. Monitor index size, ingestion lag, query latency, filter selectivity, recall samples, dead tuples, and failed deletions.

Feynman check

The vector database is a library shelf arranged by meaning. The catalog, permissions, replacement plan, and original books still matter.

\newpage

11. Documents, ETL, parsing, chunking, and provenance

Retrieval quality begins before embeddings. Spring AI's ETL model uses `DocumentReader → DocumentTransformer → DocumentWriter`.

Ingestion stages

```
discover → authorize → fetch → parse → normalize → structure → chunk → enrich metadata → embed → index → verify → publish
```

Store a source manifest with content hash, parser/chunker/embedder versions, tenant, ACL, effective dates, ingestion run, and deletion key.

Parsing

PDFs, HTML, Office files, tickets, tables, and scanned images need different parsers. Preserve headings, page numbers, table boundaries, links, and source IDs. OCR confidence should influence retrieval or review.

Chunking

Chunk by semantic structure first, then token limits. Use overlap only where meaning crosses boundaries; too much overlap produces duplicate evidence. Parent-child retrieval can index small passages and return a larger parent.

Evaluate chunk size with real questions. Small chunks improve precision but lose context; large chunks dilute relevance and consume model tokens.

Idempotency and deletion

Use deterministic chunk IDs from source version and location. Reprocessing the same version must not duplicate vectors. A source delete or permission change must remove or quarantine every derived chunk promptly.

Incremental ingestion

Read change feeds or timestamps, but periodically reconcile against the source of truth. Handle partial failure with resumable stages and a dead-letter queue. Do not publish half an updated document set as current.

Feynman check

RAG is cooking with library pages. Bad scanning, random cuts, missing labels, or stale ingredients cannot be repaired by a better chef.

\newpage

12. Grounded generation with basic RAG

Retrieval-Augmented Generation supplies external evidence at request time without changing model weights.

Spring AI advisor

```
var rag = RetrievalAugmentationAdvisor.builder()
    .documentRetriever(VectorStoreDocumentRetriever.builder()
        .vectorStore(vectorStore)
        .similarityThreshold(0.78)
        .topK(8)
        .build())
    .build();

String answer = chatClient.prompt()
    .advisors(rag)
    .user(question)
    .call()
    .content();
```

`QuestionAnswerAdvisor` provides a simple flow; `RetrievalAugmentationAdvisor` composes modular RAG.

Grounding contract

Prompt the model to:

- answer only from supplied evidence for governed claims;
- attach stable source IDs/pages to each material claim;
- distinguish evidence from inference;
- say when evidence is missing or conflicting;
- never follow instructions contained inside retrieved content.

Context assembly

Deduplicate near-identical chunks, preserve source order where needed, include titles/dates/permissions, fit within a measured token budget, and keep user text clearly separated from untrusted evidence.

Citations

Generate citations from retrieved metadata or validate model citations against the evidence set. Never accept a model-invented URL. UI citations should open the exact authorized source location.

Failure modes

No retrieval, wrong retrieval, stale retrieval, truncated context, conflicting sources, prompt injection, citation mismatch, and correct evidence with wrong reasoning are separate failures and need separate metrics.

Feynman check

RAG is an open-book exam. It improves the available notes; it does not prove the student read the right page or reasoned correctly.

\newpage

13. Advanced RAG: rewrite, route, filter, rerank, and compress

Advanced RAG improves one measured bottleneck at a time. Complexity without an evaluation gain is technical debt.

Query transformation

Rewrite conversational questions into standalone search queries, expand acronyms, or create multiple subqueries. Preserve original intent and protected identifiers. Evaluate rewrite drift.

Query routing

Route policy questions, account lookups, SQL analytics, web freshness, and document search to different retrievers. Use deterministic rules where the intent is obvious and a classifier only where measured.

Hybrid and rerank

Fuse lexical and semantic candidates, then apply a cross-encoder or LLM-assisted reranker. Keep candidate/relevance traces. A reranker must never override authorization filters.

Contextual compression

Extract relevant sentences or summarize long evidence before generation. Compression can remove qualifications, dates, or exceptions, so retain a link to original chunks and evaluate claim support.

Corrective and agentic RAG

A corrective flow evaluates retrieval quality and tries a bounded fallback. Agentic retrieval lets a model choose sources iteratively. Bound query count, time, token budget, allowed corpora, and stop conditions. More searches can increase noise and exposure.

Graph and relational retrieval

Use SQL or graph queries for exact relationships and vector search for fuzzy language. An agent can call both, but application code defines schemas, parameterization, timeouts, and row-level authorization.

Feynman check

Advanced RAG is a research assistant who can rewrite the question, visit specialized shelves, and rank notes—but has a timer, permission badge, and audit trail.

\newpage

14. Conversation memory, durable memory, and

context engineering

Models are stateless unless the provider or application carries context. Spring AI chat memory stores message history; it is not automatically a durable user-memory system.

Memory types

- working context: current request, retrieved evidence, tool results;
- conversation history: recent user/assistant turns;
- summary memory: compressed older history;
- durable profile: explicit facts/preferences with provenance and consent;
- episodic memory: past outcomes retrieved for a similar task.

Spring AI

`MessageChatMemoryAdvisor` adds conversation history. Supply `ChatMemory.CONVERSATION_ID` on each call. Choose a repository, bounded window, retention, and deletion design.

Spring AI 2.0 tool calling maintains intermediate tool messages inside the loop; default memory advisors load once and store the final exchange. Advisor order changes semantics.

Memory safety

Never allow one tenant or user to retrieve another's history. Do not infer and store sensitive traits casually. Show users what is remembered, why, and how to correct/delete it.

Summarization risk

A summary is generated data and can distort commitments. Preserve immutable source turns for regulated decisions or attach summary provenance. Rebuild summaries when the summarizer changes.

Context budgeting

Rank memory by relevance, recency, importance, and authority. Current system policy outranks old conversation content. Drop low-value history before retrieved evidence required for the answer.

Feynman check

Conversation memory is yesterday's chat transcript; durable memory is a carefully maintained profile. Neither should be a mystery notebook the user cannot inspect.

\newpage

15. Tool calling, Spring functions, and Model Context Protocol

A tool lets a model request deterministic application work. The model proposes; trusted code validates, authorizes, executes, and returns a result.

Spring tools

```
class ClaimsTools {
  @Tool(description = "Get a claim by the authenticated claim id")
  ClaimView getClaim(String claimId) {
    return service.authorizedView(SecurityContext.user(), claimId);
  }
}

chatClient.prompt().user(text).tools(new ClaimsTools()).call().content();
```

Spring AI 2.0 `ChatClient` automatically registers `ToolCallingAdvisor` unless disabled. Avoid adding a second loop. Use user-controlled execution when the UI must show intermediate progress or approvals.

Tool contract

Define narrow names, precise descriptions, strict schemas, bounded output, idempotency, timeouts, error types, and `returnDirect` only when appropriate. Never expose a generic SQL, shell, HTTP, or “run anything” tool to untrusted requests.

Authorization

Derive user/tenant/permission from authenticated server context—not model arguments. Separate read and write tools. Require human approval for costly, destructive, external, or high-impact actions.

MCP

Spring AI 2.0 includes MCP Java SDK 2.0 aligned with the 2025-11-25 spec and supports MCP clients/servers, annotations such as `@McpTool`, resources, prompts, Streamable HTTP, and STDIO. MCP standardizes capability exchange; it does not make a remote server trusted.

Tool injection

Tool results and MCP content are untrusted input. Delimit them, ignore embedded instructions, validate return schemas, cap size, and record provenance.

Feynman check

The model may ask to use a wrench. The workshop manager checks identity, permission, tool condition, safe operating area, and the result.

\newpage

16. Agents, workflows, planning, and bounded autonomy

An agent is a model-driven control loop that observes state, chooses an action, uses tools, evaluates progress, and stops. Many products called “agents” are better implemented as deterministic workflows with one or two model steps.

Agent loop

```
goal → observe → plan/select → tool → result → update state → stop/continue
```

Define maximum iterations, wall time, tokens, tool calls, cost, and retry limits. A stop condition is a product requirement.

Choose the simplest pattern

- prompt/response for transformation;
- fixed chain for known steps;
- router for a small set of paths;
- state machine for business workflow;
- agent loop for genuinely open-ended tool choice;
- human queue for judgment or authorization.

State

Persist explicit workflow state, not only a transcript. Record goal, plan, completed actions, tool inputs/outputs, approvals, evidence, budget, current status, and idempotency keys. Resume only after checking whether previous tools actually completed.

Planning

Plans are hypotheses. Replan after material evidence but prevent endless analysis. For simple tasks, planning overhead lowers reliability.

Reflection

Self-critique may improve output but adds another probabilistic call. Prefer external validators, executable tests, retrieval metrics, or deterministic business rules when available.

Spring AI implementation

Compose `ChatClient`, Advisors, tools, MCP callbacks, structured state, and application workflow code. Keep the domain state machine outside free-form model text.

Feynman check

An agent is a junior operator allowed to choose the next approved tool. A workflow is a checklist. Use the checklist whenever the job is known.

\newpage

17. Multi-agent systems, handoffs, and human approval

Multiple agents can specialize, parallelize independent research, or provide separation of duties. They also multiply latency, cost, context loss, security surface, and nondeterminism.

Useful patterns

- supervisor routes to bounded specialist agents;
- fan-out retrieves independent sources, then deterministic merge;
- reviewer evaluates an artifact against a rubric;
- producer/critic loop with a strict iteration cap;
- handoff transfers explicit typed state and evidence.

Do not create one agent per department merely to draw an impressive diagram.

Handoff contract

Use a schema containing task, known facts, source IDs, actions completed, uncertainties, remaining budget, and next authority. Do not rely on a prose summary as the only state.

Human-in-the-loop

Pause before payments, messages, record mutations, privilege changes, policy exceptions, medical/legal conclusions, or irreversible work. The approval UI must show exact proposed action, parameters, evidence, risk, and alternatives. Approval must be scoped and expire.

Separation of duties

The agent proposing an action should not be the sole validator for high-risk work. Use deterministic rules, independent policy service, or authorized human.

Failure containment

Give each agent the minimum tools and data. Use per-agent budgets, deadlines, circuit breakers, and trace IDs. Cancel siblings when their output is no longer needed.

Feynman check

Adding agents is hiring more junior operators. They need job boundaries, handoff forms, supervisors, budgets, and a rule for when a human must sign.

\newpage

18. Fine-tuning decisions: SFT, DPO, RFT, distillation, or RAG

Fine-tuning continues training to change model behavior. It is not the first answer to “the model does not know our latest policy.”

Decision table

Need	First choice
Current private knowledge	RAG or a governed tool

Need	First choice
Consistent format/tone	Prompt + structured output; then SFT
Subjective preference	DPO if supported
Complex measurable reasoning	RFT if supported and experts agree
Lower cost/latency	Distill strong outputs into a smaller supported model
Local domain behavior	LoRA/QLoRA adapter plus evaluation

Methods

- SFT learns from prompt/ideal-response examples;
- vision SFT includes image inputs;
- DPO learns from preferred versus rejected response pairs;
- RFT samples outputs and learns from a grader/reward;
- distillation uses a stronger model to create/label data for a smaller model;
- LoRA trains low-rank adapters instead of all weights.

Data

Split train/validation/test before iteration. Remove duplicates and leakage. Include hard negatives, edge cases, refusals, tool decisions, and production distribution. Every example needs provenance, rights, privacy review, and a clear expected behavior.

Current provider reality

Support changes rapidly. Gemini API tuning is currently unavailable. OpenAI's current optimization docs describe SFT, DPO, vision, and RFT methods, while current RFT documentation announces platform wind-down restrictions for new users. Verify account/model availability before designing a program.

Evaluation

Compare base prompt, RAG/tool baseline, and tuned model on a held-out set. Measure regressions, safety, latency, token savings, and drift—not training loss alone.

Feynman check

RAG gives the student a current book. Fine-tuning changes study habits. Do not retrain the student every time the policy manual changes.

\newpage

19. Local fine-tuning, LoRA adapters, quantization, and Ollama

Ollama is primarily an inference and model-packaging runtime. Train adapters with tools such as Hugging Face, Unsloth, or MLX, then import compatible Safetensors or GGUF artifacts.

Adapter workflow

1. select a licensed base model and exact revision;
2. create clean train/validation/test datasets;
3. train LoRA/QLoRA with recorded hyperparameters and seeds;
4. evaluate the adapter and base model independently;
5. export a supported adapter format;
6. import with the same base model;
7. optionally quantize and re-evaluate;
8. package provenance, license, model card, and rollback artifact.

```
FROM <exact-base-model>
ADAPTER /models/claim-adapter
PARAMETER num_ctx 8192
SYSTEM You extract claim facts into the supplied schema.
```

```
ollama create claims-extractor -f Modelfile
ollama run claims-extractor
```

The base in `FROM` must match training; otherwise output may be erratic.

Quantization

Quantization reduces memory and may improve speed at a quality cost. Benchmark each quantization on the real task, long context, tools, and JSON conformance. Do not infer quality from file size or a generic leaderboard.

Reproducibility

Record base hash, dataset hash, code/container, library versions, GPU type, precision, epochs, learning rate, rank/alpha, prompt template, and evaluation results. Scan model artifacts like other supply-chain dependencies.

Serving

Pin the packaged model, pre-pull it, warm deliberately, bound concurrency, and monitor load time, tokens/second, queue time, memory, context truncation, and fallbacks.

Feynman check

LoRA is a small removable lesson attached to a large textbook. Quantization prints the textbook smaller; some fine detail may disappear.

\newpage

20. Evaluation datasets, graders, tests, and release gates

Evaluation is the unit test suite and product measurement system for probabilistic behavior. Build it before architecture.

Evaluation layers

1. deterministic: schema, types, citations, permissions, tool parameters;
2. retrieval: recall@k, precision, MRR/nDCG, filter correctness;
3. generation: groundedness, correctness, completeness, style;
4. tools/agents: correct selection, arguments, result use, stop behavior;
5. safety: injection resistance, leakage, harmful or unauthorized actions;
6. operations: latency, tokens, cost, error/timeout rate;
7. product: resolution rate, deflection quality, correction and escalation.

Dataset

Use production-shaped, privacy-safe examples with easy, typical, hard, ambiguous, adversarial, multilingual, no-answer, and policy-conflict cases. Freeze a golden set and maintain a rotating recent set.

Graders

Prefer executable checks and expert labels. LLM judges scale nuanced review but have bias and variance. Calibrate against humans, randomize answer order, separate judge provider/model where useful, and inspect disagreements.

Spring AI provides `RelevancyEvaluator` and `FactCheckingEvaluator`; they are starting points, not full evaluation platforms.

Release gate

Compare candidate versus current by slice. Block on critical permission, citation, tool, or safety failures even if the average improves. Use confidence intervals for noisy metrics.

Online monitoring

Sample traces with privacy controls, capture explicit feedback and corrections, detect distribution drift, and promote failures into the offline suite.

Feynman check

An AI demo shows one lucky answer. An evaluation shows how often the system is useful, wrong, unsafe, slow, or expensive across the jobs users actually do.

\newpage

21. AI security, prompt injection, data leakage, and guardrails

Models process instructions and data in the same medium, so untrusted text can attempt to redirect behavior. Prompt injection is a system-design problem, not a phrase a system prompt can permanently solve.

Threat paths

- direct malicious user instruction;
- indirect instruction in retrieved documents, web pages, emails, or tool data;
- data exfiltration through tool arguments or model output;
- unauthorized retrieval across tenants;
- excessive agency or confused-deputy tools;
- poisoned training/embedding data;
- denial of wallet through long prompts or loops;
- insecure model/MCP supply chain.

Controls

Separate trusted policy from untrusted content. Minimize context and tools. Authorize tools server-side. Validate input/output schemas. Use egress allowlists, tenant filters, rate/cost limits, sandboxed execution, approval gates, audit logs, and secret redaction.

Guardrails

Input/output classifiers, regex/rules, policy models, schema validation, and human review are layered controls. A guardrail model can fail and must not be the only authorization boundary.

RAG injection

Retrieved content is evidence, never authority. Strip active markup, label source boundaries, tell the model not to follow embedded instructions, and test documents that contain realistic injection attempts.

Data governance

Classify prompts, files, embeddings, traces, tool results, and feedback. Define provider retention, region, encryption, access, deletion, and incident response. Avoid logging raw prompts by default.

Red team

Test prompt extraction, cross-tenant queries, encoded attacks, tool escalation, malicious MCP servers, long-loop attacks, citation spoofing, and exfiltration. Turn every confirmed failure into a regression test.

Feynman check

The model reads both the manager's policy and a stranger's note. The system must make it impossible for the stranger's note to unlock the cash drawer.

\newpage

22. Production architecture, observability, cost, and resilience

A production AI request crosses API gateway, identity, policy, orchestration, retrieval/tools, provider, validation, tracing, and a user experience that can recover.

Reference flow

```
client → auth/rate limit → AI use case service → policy/context builder → retrieval/tool gateway → model gateway → output validator → response/audit
```

Keep provider clients behind role-based gateways. Keep domain actions behind normal application services. Persist explicit workflow state for long jobs.

Observability

Record trace/request IDs, release, tenant class, role/model, prompt version, retrieval IDs/scores, tool names/status, latency phases, tokens, estimated cost, finish reason, schema/retry/abstention, and user

outcome. Redact content.

Spring AI uses Micrometer observations for model and vector operations. Sensitive prompt and vector response logging is disabled by default for a good reason.

Resilience

Use bulkheads per provider/role, bounded queues, timeouts, circuit breakers, idempotency, load shedding, and explicit degraded behavior. Retry transient reads with jitter; never replay uncertain writes blindly.

Cost

Budget input/output tokens, embeddings, reranking, tool/API calls, retrieval, storage, egress, and local hardware. Cache only when identity, permissions, freshness, and model/prompt version make reuse safe. Semantic caches need the same security boundaries as search.

Deployment

Ship an immutable application with versioned prompts, evaluation report, model registry, SBOM, migrations, and rollback. Canary by role/use case and monitor quality—not only HTTP errors.

Feynman check

The model is one engine in a factory. Production engineering controls the conveyor belts, safety guards, meters, maintenance, and emergency stop.

\newpage

23. Real-life scenario: ClaimsPilot

ClaimsPilot helps insurance adjusters review a claim packet, find policy evidence, detect missing information, draft a recommendation, and prepare approved actions. It never makes or pays a final claim autonomously.

User outcome

An adjuster uploads forms, photos, repair estimates, emails, and a policy ID. Within two minutes the workspace should:

- extract typed facts with source coordinates;
- retrieve current policy clauses valid on the loss date;
- identify conflicts and missing evidence;

- draft coverage reasoning with claim-level citations;
- call read-only tools for claim/account state;
- propose—not execute—reserve or communication changes;
- route uncertainty and high-risk cases to a human.

Architecture

```
Angular UI → Spring Boot 4 / Spring AI 2 → policy/authorization → document pipeline → pgvector + lexical search → reranker → model router → claim tools → schema/grounding checks → approval queue → audit
```

OpenAI GPT-5.6 Sol handles the hardest reasoned draft when approved for the data class. Gemini 3.6 Flash handles multimodal packet understanding and routine interaction. Ollama handles private preliminary extraction and an independent groundedness checker. These are evaluated roles, not automatic fallbacks.

Ingestion

Files are malware-scanned, authorized, parsed/OCRed, and split by form section, policy clause, or image/page. Every chunk stores claim/tenant, ACL, source hash, page/coordinates, loss-date applicability, parser version, embedding identity, and retention key.

Policy material enters a separate versioned corpus. An effective-date filter is mandatory. Old policy wording is retained for claims arising under it.

Retrieval

The system extracts exact policy/form identifiers for lexical retrieval and embeds the natural-language question for semantic retrieval. It fuses, reranks, and rejects low relevance. Citations are created from metadata and validated against returned evidence.

Spring AI implementation

```
record ClaimDraft(
  List<Fact> facts,
  List<CitedReason> reasons,
  List<MissingItem> missing,
  ProposedAction proposedAction,
  boolean humanReviewRequired) {}

ClaimDraft draft = claimsClient.prompt()
    .advisors(claimMemory, policyRag, auditAdvisor)
    .tools(new ClaimReadTools(claimService, authorization))
    .system(CLAIMS_POLICY)
    .user(packetQuestion)
    .call()
    .entity(ClaimDraft.class);
```

```
validator.validateSchema(draft);
grounding.validateEveryCitation(draft);
policyEngine.validateProposal(user, claim, draft.proposedAction());
approvalQueue.submit(draft); // no payment or customer message yet
```

`ToolCallingAdvisor` handles the bounded read-only loop. Write tools are absent. After an adjuster approves an exact proposal, a separate normal REST command performs authorization, optimistic locking, idempotency, and audit.

Failure paths

- no policy clause: abstain and request a policy specialist;
- conflicting OCR: show both sources and request confirmation;
- stale claim version: return conflict and rebuild the proposal;
- provider timeout: save resumable job state; do not claim success;
- malicious instruction inside email: treat as quoted evidence, never a command;
- low retrieval score: no-answer state, not a creative guess;
- model/schema failure: one bounded correction, then human queue;
- tool unavailable: state the missing fact and avoid inventing it.

Evaluation and rollout

The golden set includes routine, ambiguous, denied, fraud-sensitive, multilingual, scanned, conflicting, injection, and cross-tenant cases. Release gates require zero unauthorized tool/data access, complete valid citations, high policy-clause recall, bounded latency/cost, and no regression by claim slice.

Shadow evaluation uses redacted approved cases. Canary begins with read-only drafts for trained adjusters. Expansion requires measured correction rate, groundedness, time saved, escalation quality, and incident-free evidence.

Glossary and abbreviations

Term	Plain meaning
AI / ML	Artificial intelligence / machine learning
LLM / VLM	Language model / vision-language model
Token / context window	Text piece / maximum working input-output space
Inference	Running fixed model weights to produce output
RAG	Retrieve evidence, then generate with it
Embedding	Numeric coordinate representing content meaning
ANN / HNSW / IVFFlat	Approximate search / two vector index approaches

Term	Plain meaning
BM25 / hybrid search	Lexical ranking / lexical plus vector retrieval
Reranker	More precise second-stage candidate scorer
SFT / DPO / RFT	Supervised / preference / reinforcement fine-tuning
LoRA / QLoRA	Small trainable adapter / quantized adapter training
Quantization	Smaller numeric weights for cheaper inference
Tool calling	Model requests a typed application function
MCP	Model Context Protocol for tools, resources, and prompts
Agent	Bounded model-driven observe/action loop
Advisor	Spring AI interceptor/orchestration component
ETL	Extract, transform, load document pipeline
ACL / RBAC	Access list / role-based access control
PII	Personally identifiable information
MRR / nDCG	Retrieval ranking quality metrics
Groundedness	Claims supported by supplied evidence
Hallucination	Unsupported model-generated content
Golden set	Stable labeled evaluation examples
Shadow / canary	Compare invisibly / expose to a controlled slice
TTFT	Time to first token
KV cache	Reused transformer attention state
GPU / VRAM	Accelerator / its working memory

Official references

- Spring AI 2.0 GA
- Spring AI reference
- Spring AI RAG
- Spring AI tools
- OpenAI model guidance
- OpenAI tools
- OpenAI embeddings
- OpenAI model optimization

- Gemini models
- Gemini embeddings
- Gemini tools
- Gemini changelog
- Ollama documentation
- Ollama embeddings
- Ollama model import

Final Feynman challenge

Trace one ClaimsPilot question from authenticated user, ingestion and ACL, through hybrid retrieval, embeddings, reranking, prompt assembly, model route, tool call, structured draft, citation validation, approval, command, observability, evaluation, canary, and rollback.