Generative AI 101 Fundamentals by Michaël Bettan

01. Definitions & Scope

What is GenAI?

Generative AI (GenAI) encompasses machine learning models designed to generate novel content (text, images, audio, video, structured data, and executable code) by predicting patterns learned across massive training corpora.

- **LLMs (Large Language Models):** The foundation of text- and code-based GenAI. Built on the **Transformer architecture** and **self-attention mechanisms** to weigh token relevance across long context windows (up to 1M–2M+ tokens). Modern frontier models predominantly use **Sparse Mixture-of-Experts (MoE)** routing—activating only a small subset of specialized expert sub-networks per token so total parameter capacity scales without multiplying per-token inference latency or FLOPs.
- **Multimodal models:** Built from the ground up (*native early-fusion*) to ingest, reason across, and generate multiple modalities (text, images, audio, video, PDFs) inside a unified representation space (*examples: Google Gemini 4 Argon / 3.8 Flash, Anthropic Claude Opus 4.7 / Sonnet 4.6, OpenAI GPT-5 / GPT-4o*).
- **Native Multimodal vs. Modular Perception Pipelines:** Native multimodal ingestion preserves visual layout, audio prosody, and cross-modal spatial relationships without OCR/ASR information loss. However, **modular pipelines** (such as dedicated Speech-to-Text or Document AI Layout Parser stages ahead of an LLM) remain preferable when compliance requires deterministic **PII redaction on intermediate transcripts** before data reaches the foundation model, strict bounding-box coordinate auditability, or ultra-cheap batch pre-filtering.
- **Synthetic media:** The umbrella industry term for AI-generated text, images, audio, video, and tabular/training data—replacing the narrower legacy term "deepfake," which applies only to deceptive likeness synthesis.

02. Standard Workflow

1. **Define Objective:** Specify the business goal, target audience, latency/cost envelope, tone, and structured output schema.
2. **Model Selection:** Match task complexity, modality, and governance constraints to the right model tier and hosting boundary (frontier LLM vs. specialized SLM, SaaS/MaaS vs. dedicated VPC endpoint).
3. **Prompt & Context Engineering:** Craft clear system instructions, schemas, and curated context windows.
4. **Generation & Iteration:** Run baseline evaluations, inspect traces, collect human/automated feedback loops (thumbs up/down, rubric scores), and iterate.
5. **Troubleshooting:** Diagnose lazy completions, hallucinations, context truncation, or refusals by adjusting reasoning effort, decoding parameters, or context structure.
6. **Evaluation & Fact-Checking:** Cross-reference outputs with deterministic checks, autoraters, and human oversight—GenAI is probabilistic, never infallible.
7. **Advanced Customization:** Climb the customization ladder deliberately: **Prompting → RAG / Grounding → Supervised Fine-Tuning (SFT) → Preference Alignment (DPO/RLHF)**.
8. **Finalizing & Editing:** Harmonize voice, verify citations and code dependencies, and apply human editorial judgment.
9. **Deployment & Ethics:** Attach provenance/watermarks (`SynthID`, `C2PA`), enforce runtime guardrails, cite sources, and monitor production telemetry for model or data drift.

03. Ecosystem & Key Players

Selecting the right model family depends on reasoning complexity, multimodal requirements, latency budget, licensing, and data-residency rules:

- **Google (Gemini & Gemma):** Native multimodal Sparse MoE frontier models (**Gemini 4 Argon** for complex multi-step reasoning and agentic coding, **Gemini 3.8 Flash** as high-throughput workhorse, **Gemini 3.8 Live** for sub-second bidirectional voice/video, and **Gemini 3.5 Flash-Lite** for high-QPS routing) integrated across Google Cloud Agent Platform, Search grounding, and Workspace—paired with **Gemma 4** (`E2B`, `E4B`, `26B A4B MoE`, `31B Dense`) open-weight models and DeepMind research (`SynthID`, `Alpha Evolve`).
- **Anthropic (Claude):** Safety-first frontier models (**Claude Opus 4.7**, **Claude Sonnet 4.6**, **Claude Haiku 4.5**) pioneering **Constitutional AI (RLAIF)**—aligning outputs to an explicit written constitution rather than relying solely on ad-hoc human ratings—alongside native **Adaptive Thinking**, 1M-token context engineering, mechanistic interpretability, and computer/tool use.
- **Meta (Llama):** Broad open-weight model ecosystem empowering organizations to fine-tune, quantize, and deploy weights across cloud Model-as-a-Service (MaaS) catalogs, private VPCs, or air-gapped hardware.
- **DeepSeek:** Demonstrated that **Sparse Mixture-of-Experts (MoE)** architectures, multi-head latent attention, and reinforcement-learned reasoning distillation can rival massive proprietary models in math and coding at a fraction of active compute.
- **OpenAI (ChatGPT / GPT-5 / o-series):** Early pioneer of commercial conversational LLMs (`GPT-4`, `GPT-4o`) and inference-time reasoning scaling (`o1` through `o3` / `GPT-5`).

04. Model Selection Matrix: LLM vs. SLM, OSS vs. Proprietary & Self-Hosted vs. SaaS

Architectural exams (especially **Google PAA Section 3.1** and **CCAR-F**) test model selection across **three orthogonal axes**. Never conflate model *size*, *weight licensing*, and *hosting topology*.

Axis 1: Capacity Tier — Frontier LLM vs. Small Language Model (SLM)

Dimension	Frontier LLM (`Gemini 4 Argon`, `Claude Opus 4.7`)	Workhorse Mid-Tier (`Gemini 3.8 Flash`, `Claude Sonnet 4.6`)	Small Language Model / Lite (`Gemini 3.5 Flash-Lite`, `Gemma 4`, `Claude Haiku 4.5`, `Gemini Nano`)
Best for	Ambiguous multi-step planning, cross-document synthesis (1M+ tokens), complex debugging, novel edge cases	80–90% of production agentic tool-calling, RAG synthesis, code generation, customer workflows	Intent routing, schema extraction, PII/safety classification, high-QPS batch tagging, on-device/edge inference
Latency & Cost	Highest TTFT & per-token cost; supports deep System 2 reasoning	Balanced sub-second TTFT and strong reasoning-per-dollar	Ultra-low latency (<150 ms) and 10×–30× lower token cost; fits single GPU or mobile NPU
Failure Mode	Overkill latency and token burn on deterministic or single-step tasks	May miss subtle 6+ hop constraints without explicit decomposition	Brittle on open-ended multi-hop planning or unstated domain assumptions

Axis 2: Weight Licensing — Proprietary Frontier vs. Open-Weight (OSS)

- **Proprietary Models (`Gemini`, `Claude`, `GPT`):** Deliver peak zero-shot reasoning, native 1M+ multimodal context, built-in safety filters, and managed vendor SLAs—at the trade-off of API lock-in and vendor model-retirement lifecycles (requiring version aliases and migration eval suites).
- **Open-Weight Models (`Gemma 4`, `Llama`, `Mistral`, `DeepSeek`):** Provide **weight permanence** (frozen checkpoints never deprecated by a vendor), deep **LoRA / QLoRA / full-parameter fine-tuning**, custom quantization (`FP8` / `INT4`), and portability across clouds or air-gapped environments.
- **Safety Caveat:** Open weights ship without managed platform guardrails; production deployments must wrap inputs and outputs in an external runtime safety layer (such as **Model Armor** or **ShieldGemma 2**).

Axis 3: Hosting & Consumption — Managed SaaS / MaaS vs. Dedicated Self-Hosted VPC

> **EXAM TRAP — OPEN-WEIGHT ≠ SELF-HOSTED GPU OPS**
> Choosing an open-weight model (such as Llama or Gemma 4) does **not** automatically mean provisioning GPUs on Kubernetes. Always evaluate **Managed Model-as-a-Service (MaaS)** serverless endpoints first before paying the 24/7 idle tax of dedicated nodes.

Consumption Mode	Billing Model	Ops Overhead	Choose When
Serverless SaaS / MaaS API (First-party Gemini/Claude or Model Garden MaaS Partner/OSS)	**Pay-As-You-Go per token** (or reserved **Provisioned Throughput** in GSUs for guaranteed latency SLAs)	Zero infra management; instant autoscaling to zero	Spiky or unpredictable traffic, rapid iteration, standard enterprise privacy (zero training on customer prompts, VPC-SC, regional endpoints)
Dedicated Private Endpoint (Managed Model Garden Endpoint)	**Per node-hour (GPU/TPU) — bills 24/7 even at 0 QPS** until undeployed	Managed container serving, manual replica/autoscaling sizing	Strict regulatory mandate prohibiting shared multi-tenant SaaS compute, hosting custom LoRA/fine-tuned open weights, or steady high-duty-cycle volume
Self-Hosted on GKE / Bare Metal (`vLLM`, `JetStream`, `TGI` on GPU/TPU nodes)	**Cluster compute + storage + networking**	Highest: driver patching, KV-cache tuning, autoscaler ops	Full custom serving kernels, multi-LoRA bin-packing across dozens of adapters, or completely air-gapped / sovereign environments

The Multi-Model Routing Cascade & Customization Ladder

1. **Multi-Model Routing Cascade:** Do not send every turn to a frontier model. Place a fast **SLM classifier/router** (`Gemini 3.5 Flash-Lite` or `Haiku 4.5`) in front to handle simple lookups and route standard tool loops to a mid-tier model (`Gemini 3.8 Flash` / `Sonnet 4.6`), escalating to a frontier reasoning model (`Gemini 4 Argon` / `Opus 4.7`) **only** on low confidence, complex planning, or verification failures.
2. **The Customization Ladder:** - Need **fresh, mutable, or citable facts** with per-user ACLs? Use **RAG / Grounding**, **not** fine-tuning (weights freeze stale facts and cannot enforce row-level permissions). - Need **consistent domain tone, terse output formatting, or specialized syntax** (so you can shrink bloated few-shot prompts into a fast SLM)? Use **Supervised Fine-Tuning (SFT)** or **Model Distillation**.

05. System 1 vs. System 2 Reasoning: Inference-Time Compute

Early LLMs operated purely in **System 1** mode—emitting the final answer token-by-token in a single forward pass with fixed compute per token. Modern frontier models add **System 2 reasoning** (*inference-time / test-time compute scaling*): spending variable internal **thinking tokens** to decompose problems, test hypotheses, catch mistakes, and plan between tool calls before emitting user-visible text.

System 1 vs. System 2 Comparison

- **System 1 (Fast / Non-Reasoning or Minimal Effort):** Immediate autoregressive decoding. Lowest Time-to-First-Token (TTFT) and lowest output-token cost. Ideal for classification, entity extraction, translation, straightforward summarization, and low-latency voice turn-taking.
- **System 2 (Deliberative / Extended Reasoning):** Generates an internal chain of thought trained via reinforcement learning before responding and **interleaves reasoning between tool results** (*observe tool error → reason about root cause → choose corrective tool*). Essential for ambiguous multi-constraint planning, root-cause code debugging, theorem/math verification, and policy conflict resolution.

2026 Vendor Reasoning Controls

Capability	Google Gemini (`Gemini 4 Argon` & `Gemini 3.8`)	Anthropic Claude (`Claude Opus 4.7` & `Sonnet 4.6`)
Primary Control Knob	`thinking_level`: `MINIMAL`, `LOW`, `MEDIUM`, `HIGH` (replaces legacy manual token-count `thinking_budget` on Gemini 4)	**Adaptive Thinking** (`thinking: {type: "adaptive"}`) paired with `output_config.effort`: `low`, `medium`, `high`, `xhigh` (long-horizon coding/agents), `max`
Multi-Turn Tool State	Emits encrypted `thought_signatures` alongside tool calls; the agent harness **must pass them back verbatim** on subsequent turns so multi-step reasoning state is preserved	Native **interleaved thinking** between tool calls (`budget_tokens` manual mode is deprecated on 4.6 and removed on Opus 4.7)
Real-Time Voice Bridge	`gemini-3.8-live-extended-thinking` decouples fast conversational turn-taking from deeper background System 2 reasoning	Pair fast voice/UI acknowledgement with asynchronous tool/subagent execution
Sampling Interaction	Keep default sampling settings when `thinking_level` is active to avoid destabilizing reasoning chains	Custom `temperature` / `top_p` / `top_k` (4.7+) and assistant message prefilling (4.6+) are removed; steer behavior via `effort` and `output_config.format` schemas

> **COST & LATENCY WARNING**
> Internal thinking tokens are **billed as output tokens** and delay visible TTFT unless streamed as thought summaries. Never default every microservice call to `HIGH` / `max`—calibrate reasoning effort per route using offline evaluation sets.

06. Prompt Engineering & Frameworks

Prompting Frameworks

Reusable, structured templates designed to build consistent, high-quality prompts instead of starting from scratch:

- **ReAct (Reasoning + Acting):** Alternates between reasoning steps and external tool calls (*Thought: I need live flight inventory → Action: Call Flight API → Observation: 3 seats left*). Foundational loop for AI agents.
- **PACT:** Defines **P**ersona, **A**ction, **C**ontext, and **T**one. Best for marketing, copywriting, and maintaining brand consistency.
- **WISER:** Defines **W**ho (Persona), **I**nstruction (Main task), **S**ubtasks (Decomposition), **E**xamples (Few-shot), and **R**eview (Self-correction). Best for complex, structured single-call workflows.

Core Prompting Techniques

- **Prompt:** A natural-language instruction, schema, and context payload sent to the model. Outputs remain probabilistic—effective engineers specify explicit constraints, edge-case rules, and verification criteria.
- **Iterative Prompting:** A flexible cycle of prompting, evaluating against test cases, and refining instructions until output meets quality bars.
- **Prompt Chaining:** Breaking a complex task into a fixed sequence of smaller prompts where each step's validated output feeds the next step.
- **Meta-Prompts:** Prompts that inspect, critique, or generate other prompts (e.g., *"Analyze these failure traces and propose a tighter system prompt"*).
- **Few-Shot Learning:** Providing 2–5 representative input-output examples inside the prompt to anchor edge-case formatting and classification boundaries.
- **System Prompt:** High-priority meta-instructions defining the model's role, invariants, tool policies, and refusal boundaries.
- **Chain-of-Thought (CoT):** Asking a non-reasoning model to *"think step-by-step"* in visible output tokens to improve logic and math accuracy. *(Note: On native System 2 models with* `thinking_level` *or Adaptive Thinking enabled, avoid rigid step-by-step CoT boilerplate—let native thinking handle internal deliberation.)*
- **Subtractive method:** Best for content creators—start with a broad draft prompt and issue follow-up prompts to whittle away unwanted elements.
- **Additive method:** Best for visual artists and scene builders—start with a foundational composition/background prompt and iteratively layer lighting, details, and textures.

07. How Models Are Trained

Training Phases

1. **Pretraining:** The base model ingests trillions of tokens of diverse web, code, and multimodal data using **self-supervised next-token prediction**, acquiring broad world representations and syntax—but not reliable instruction-following or safety boundaries.
2. **Post-Training (Alignment & Reasoning):** Adapts the base checkpoint for instruction adherence, safety, and multi-step reasoning: - **SFT (Supervised Fine-Tuning):** Trains on curated, high-quality `(instruction, ideal_response)` demonstrations so the model follows user formatting and task structure. - **RLHF (Reinforcement Learning from Human Feedback):** Humans rank candidate responses to train a separate **Reward Model**, which then updates policy weights (e.g., via PPO) toward helpful, harmless outputs. - **DPO (Direct Preference Optimization):** Directly optimizes model weights on `(chosen, rejected)` preference pairs using a closed-form classification loss—bypassing the instability and memory cost of training a separate reward model. - **RLAIF (Reinforcement Learning from AI Feedback / Constitutional AI):** Uses a capable critique model guided by explicit written principles (a constitution) to generate preference labels and self-revisions at scale. - **RLVR (Reinforcement Learning with Verifiable Rewards):** Trains System 2 reasoning models against deterministic verifiers—unit-test suites, compilers, math proof checkers, and sandbox execution—rewarding multi-step chains that arrive at objectively verified solutions.
3. **Model Distillation:** Transfers domain behavior or reasoning traces from a large frontier **"Teacher"** model into a compact **"Student"** SLM, preserving task accuracy at a fraction of serving latency and cost.

08. Advanced Tactics & Troubleshooting

Multi-Model Composition Tactics

- **AI Aggregation:** Combining specialized outputs from multiple distinct GenAI models into one deliverable (e.g., Claude for technical prose, Nano Banana / Imagen for illustrations, Veo for video).
- **Output Stitching:** Running multiple models in parallel on the *same* prompt, then selecting and merging the strongest segments from each response.
- **AI Chaining:** Passing the output of one GenAI model or modality directly as the input prompt/asset to a downstream GenAI tool.
- **Multi-Persona Prompting:** Instructing a single model call to critique a proposal from several distinct stakeholder viewpoints (e.g., *"Evaluate this launch plan as a CISO, a CFO, and an end customer"*).

Troubleshooting Failure Modes

- **The "Finger Problem":** Visual/structural anomalies (such as six-fingered hands in early image generators) caused by ambiguous statistical co-occurrence in training data rather than true physical-world comprehension.
- **Model / Data Drift:** Degradation when real-world facts, APIs, or user behavior diverge from the model's static training cutoff. *Combat via: RAG/search grounding, dynamic context injection, or periodic re-tuning.*
- **Overfitting (Memorization):** Training too narrowly on repeated records so the model regurgitates training text verbatim instead of generalizing. *Combat via: deduplicating training sets, regularization, adjusting sampling, and output similarity/plagiarism filters.*
- **Lazy Models:** Premature truncation, placeholder comments (`// implement rest here`), or unwarranted refusals on long tasks. *Combat via: explicit completeness criteria, chunked output chaining, calibrated* `effort` */* `thinking_level` *, and clear task framing (historically worked around in early chat models via urgency or incentive cues).*

09. Agents & Capability Levels (L0 to L4)

What Makes an Agent

An **AI agent** is a system in which a model decides, step by step, which actions to take toward a goal, observes the results, and continues until the goal is met or a stop condition fires. A chatbot answers; an agent acts.

- **Model:** The reasoning engine that interprets the goal, plans, and selects the next action.
- **Tools:** Functions, APIs, retrieval, and sandboxed code execution that let the model read live state and mutate external systems.

- **Orchestration loop:** Harness code that runs the cycle (*gather context* → *decide* → *act* → *observe*), preserves `thought_signatures` /state, enforces budgets/stop conditions, and routes errors.
- **Context and memory:** Curated working context placed in the window each turn, plus session state and long-term memory stored outside it (see the Context Engineering guide).
- **Guardrails:** Deterministic permission gates, human-in-the-loop approvals, schema validation, and audit logging that keep actions within policy.

```
goal ──► gather context ──► model decides ──► call tool ──► observe result
              ▲                                                    │
              └──────────── repeat until done or stopped ◄─────────┘
```

- **Workflow vs. Agent:** In a **workflow**, your application code fixes the sequence of model calls and tool invocations; in an **agent**, the model chooses the sequence dynamically at run time. Workflows are cheaper, faster, and more predictable; open agent loops are reserved for tasks whose path cannot be known in advance.

Agent Capability Levels (L0 to L4)

Each step up adds a concrete runtime capability—and multiplies token cost, latency, failure modes, evaluation surface, and security exposure.

Level	Adds	Use When	Example	Go Deeper
L0: model only	Nothing beyond one model call with supplied input	The answer depends only on the prompt input and general language skill	Rewrite a station closure notice in plain language; classify a ticket	Prompt Engineering
L1: model with tools	Tool calls and retrieval for live data or a single action	The task requires fresh, private, or exact data, or one external action	Look up live order status; answer from the policy library	ADP agent primitives, MCP, RAG
L2: multi-step agent	Planning across dependent steps with state and managed context	Several steps depend on earlier tool observations	Build a weekly operations report from five sources and flag anomalies	ADP workflow patterns, Context Engineering
L3: multi-agent system	Specialist agents with isolated tools, permissions, and context under a coordinator	Subtasks require distinct IAM scopes, clean context isolation, parallel branches, or an independent verifier	Read-only telemetry diagnostics agent + purchasing agent with bounded spend authority	ADP multi-agent topologies, A2A
L4: self-extending system	Creates or modifies its own tools, prompts, or subagents at run time	Tightly sandboxed research or platform environments with mandatory human review before activation	System that drafts and registers a new data connector inside an isolated sandbox	AgentOps, Responsible AI

KEY RULE

Pick the **lowest level** that solves the task. Climb one level only when an offline evaluation proves the lower level fails, and name the exact capability that justifies the climb.

Choosing a Level

- **Before L0:** If the rule is deterministic (a tax table, a date window check, an ID lookup), write deterministic code and keep the LLM out of the path. 1. **Can one call with the supplied input do it?** Stay at **L0**. Paste the text directly instead of standing up a vector pipeline. 2. **Does it need live, private, or exact data, or one external action?** Move to **L1** with least-privilege tools. 3. **Do several steps depend on each other?** Move to **L2**. Use a **workflow** when the step graph is known in advance; use an open **agent loop** only when steps depend on unpredictable runtime discoveries. 4. **Do subtasks need separate permissions, isolated context windows, parallel execution, or independent verification?** Move to **L3**. Splitting into role-play personas that share identical tools and context is *agent theater*, not justification for L3. 5. **Must the system author new capabilities at run time? L4** is rarely appropriate in production—prefer engineer-built, code-reviewed tools added to L2/L3, and require sandboxed execution plus human approval gates for anything self-generated.

Scenario: Scoping Four Requests at Spokewise Bike Share

A regional bike-share operator asks for "one AI agent" covering four operational requests:

Request	Level	Why Not Lower	Why Not Higher
Draft a rider notice about a station closure	**L0**	*(lowest level)*	Input is supplied; no live lookup or mutation needed
Tell a rider the nearest dock with a charged e-bike	**L1**	Dock availability changes every minute	Single live query; no multi-step planning
Produce the overnight truck rebalancing plan from demand forecasts, weather, and crew shifts	**L2 (workflow)**	Three dependent data sources must be fetched, reconciled, and checked	Step sequence is fixed; one read-only identity suffices

Request	Level	Why Not Lower	Why Not Higher
Run fleet maintenance: diagnose faults, order replacement parts, verify repairs	**L3**	Diagnostics needs read-only telemetry, ordering needs spend authority, repair sign-off must be independent	Connectors are built and reviewed by engineers, not synthesized at run time

- **Outcome:** One vague "agent" request became three focused builds and one multi-agent system, each evaluated at its own capability tier.
- **What the team rejected:** A single L3 "super-assistant" handling all four tasks—which would have exposed purchasing credentials to a public notice drafter and inflated latency on trivial queries.

Common Mistakes & Bridge to Architecture

- **Starting at L3 for an L1 problem:** Multiple persona agents sharing identical tools and context multiply cost and latency without adding capability (*agent theater*).
- **Staying at L0 when facts must be current:** Relying on parametric memory for live inventory or policy state guarantees extrinsic hallucinations; attach retrieval or a tool.
- **Open loops without stop conditions:** Every L1–L4 loop requires explicit `max_steps`, token/cost budgets, and loop-detection circuit breakers.
- **Prompt-only guardrails:** System-prompt wording is a behavioral hint, not a security boundary; irreversible actions require code-level IAM enforcement and confirmation gates.
- **Unreviewed self-modification in production:** Allowing an L4 agent to write and register live tools without sandboxing turns any indirect prompt injection into arbitrary code execution.

If You Chose	Continue in Agentic Design Patterns (ADP) With
L0 or L1	Agent primitives: tool contracts and stop conditions
L2 (workflow)	Workflow patterns: prompt chaining, routing, parallelization, orchestrator-workers, evaluator-optimizer
L2 (agent loop)	Agent primitives and Memory and context
L3	Multi-agent topologies: the agent theater test and handoff schemas
Any level, pre-launch	Anti-patterns and checklist

Credit: The L0–L4 capability progression builds on *Introduction to Agents* (Blount, Gulli, Saboo, Zimmermann & Vuskovic; Google, 2026), while the workflow-vs.-agent distinction follows Anthropic's Building effective agents; scenarios and decision tables here are original.

10. The 5-Layer Agentic Interoperability Protocol Stack

Modern enterprise agent architectures separate **vertical tool plumbing**, **horizontal peer delegation**, **human UI rendering**, **real-time media streaming**, and **cryptographic value transfer** into five distinct protocol layers. Mixing them up is a classic certification trap.

Layer	Boundary	Standard Protocols	Core Primitives & Mechanics	When to Use
Layer 1: Agent ↔ Tools & Data (*Vertical*)	Single agent calling deterministic APIs, databases, or local dev resources	**MCP (Model Context Protocol)**, OpenAPI	`Tools` (model-controlled actions), `Resources` (app-controlled read context), `Prompts`, structured `isError: true` payloads over `stdio` or **Streamable HTTP**	Exposing enterprise CRUD tools, SQL/search connectors, or IDE context to L1+ agents (MCP guide)
Layer 2: Agent ↔ Peer Agents (*Horizontal*)	Cross-team, cross-framework, or cross-vendor opaque agent delegation	**A2A (Agent2Agent Protocol)**	Discovery via `/.well-known/agent-card.json` or **Agent Registry**; stateful `Task` lifecycle (`submitted → working → input-required → completed`), `Artifacts`, streaming SSE	Delegating goals across organizational or runtime boundaries without exposing internal prompts, memory, or tool credentials (A2A guide)
Layer 3: Agent ↔ Rich Human UI (*Generative UI*)	Streaming interactive frontend components and bidirectional user state	**MCP UI**, **AG-UI**, **A2UI**	**AG-UI:** bidirectional client-agent event stream (state diffs, tool progress, HITL interrupts). **A2UI:** declarative JSON widget payloads rendered natively by host UIs. **MCP UI:** sandboxed interactive views returned by MCP tools	Replacing walls of raw Markdown with live seat pickers, approval cards, charts, and editable forms without letting the LLM emit raw unsanitized HTML/JS
Layer 4: Agent ↔ Real-Time Streams (*Live Media*)	Sub-second full-duplex voice, video, and screen sharing with barge-in	**Gemini Live API** over **WebSockets / WebRTC**	Continuous audio/video frame streaming, native **Voice Activity Detection (VAD)** / interruption (*barge-in*), server-side session resumption	Live voice concierges, camera-guided field diagnostics, and real-time tutoring (terminate heavy media at the edge, then invoke L1/L2 backend tools via MCP/A2A)
Layer 5: Agent ↔ Commerce & Value (*Payments*)	Verifiable human spend authorization & autonomous API micropayments	**AP2 (Agent Payments Protocol)** & **x402**	**AP2:** cryptographically signed **Mandates** (`Intent Mandate` for bounded autonomous rules; `Cart Mandate` for explicit human item/price approval) separating shopper, merchant, and credentials provider. **x402:** reviving `HTTP 402 Payment Required` for instant machine-to-machine per-request API/data micropayments	Booking travel, executing B2B procurement with non-repudiable spend proof (`AP2`), or paying per API call/dataset query autonomously (`x402`)

ARCHITECTURAL RULE OF THUMB

- **MCP is vertical** (an agent's *hands* reaching into deterministic tools/data). - **A2A is horizontal** (peer *colleagues* collaborating as opaque black boxes via `AgentCard`s). - **A2UI / AG-UI / MCP UI** govern the **human glass**, **Live API / WebRTC** governs **real-time media**, and **AP2 + x402** govern **money and cryptographic consent**.

11. RAG & Context Augmentation

RAG (Retrieval-Augmented Generation) bypasses a model's static training cutoff and reduces extrinsic hallucinations by grounding synthesis in external, permission-checked enterprise data at query time.

```
User Query ⟶ Embed / Hybrid Search ⟶ Retrieve & Rerank Chunks (ACL-filtered) ⟶ Inject Grounded Context ⟶ LLM Synthesizes Cited Answer
```

- **How it works:** User submits a query → System runs **hybrid retrieval** (dense vector embeddings for semantic meaning + sparse lexical BM25 for exact SKU/error codes) against an ACL-filtered index → Cross-encoder reranker selects top passages → Retrieved evidence is injected into the prompt → The LLM synthesizes an answer grounded *strictly* in cited passages (see the RAG guide).
- **Vector Database:** Specialized storage indexing high-dimensional embedding vectors (via HNSW or ScaNN approximate nearest-neighbor graphs) so retrieval matches **semantic similarity** rather than brittle exact-keyword overlap alone.

12. Risks, Ethics & Vulnerabilities

Core Risks & Attack Surfaces

- **Hallucinations:** Plausible-sounding but fabricated or contradicted statements. *Combat via: grounded RAG, citation verification, autoraters, and human-in-the-loop review.*
- **Extrinsic hallucination:** The model invents facts absent from the provided source context (e.g., fabricating a clause or API flag that nowhere appears in the document).
- **Intrinsic hallucination:** The model directly contradicts facts explicitly stated in the provided source context (e.g., summarizing a stated "24-month contract term" as "12 months").
- **Prompt Injection (Direct & Indirect):** Attackers embed adversarial instructions inside user input (*direct*) or untrusted external content such as emails, web pages, MCP tool outputs, or PDFs (*indirect / zero-click exploit*) to hijack agent tool calls or exfiltrate context.
- **Jailbreaking:** Adversarial framing (role-play personas, multi-turn escalation, cipher encodings) aimed at bypassing safety alignment to elicit restricted outputs.
- **Data Poisoning & Slopsquatting:** Attackers inject backdoored examples into training/RAG corpora (*poisoning*), or register malicious packages on PyPI/npm matching package names that coding LLMs frequently hallucinate (*slopsquatting*).
- **Memorization & Privacy Leakage:** Models can verbatim regurgitate PII, secrets, or copyrighted passages over-represented in training splits.
- **Detection & Content Provenance:**
- **Watermarking (`SynthID`):** Imperceptibly biases token sampling distributions (text) or frequency-domain pixel/audio signals (images, video, audio) so AI-generated media remains statistically detectable even after cropping or compression—without providing an edit history.
- **C2PA (Coalition for Content Provenance and Authenticity):** Attaches **cryptographically signed tamper-evident metadata manifests** (*Content Credentials*) recording capture origin, model edits, and custody chain.
- **Automation Bias:** The human cognitive failure of rubber-stamping fluent AI recommendations without independent verification—especially hazardous in clinical, legal, financial, and production deployment workflows. *Rule of thumb: AI drafts and proposes; accountable humans or deterministic policy gates verify.*

Ethics, Quality & Human Oversight

- **Human-in-the-Loop (HITL):** Calibrate human review to **action reversibility and blast radius**—automate low-risk reversible reads, require explicit human approval before irreversible state mutations or external communications.
- **Bias & Fairness:** Models inherit demographic and historical skews from pretraining corpora; audit disaggregated subgroup metrics and test counterfactual prompts before deployment.
- **Copyright & Plagiarism:** Mitigate verbatim reproduction via deduplicated corpora, recitation filters, grounded citations, and avoiding prompts that request living artists' or authors' proprietary styles.

13. Glossary

- **Tokens:** Sub-word units (roughly ~3–4 English characters per token) processed by tokenizers. Context limits, throughput, and API billing (including hidden System 2 thinking tokens) are metered in tokens.
- **Context Window:** Maximum token budget a model can attend over in a single turn (`input + retrieved context + tool schemas + thinking tokens + visible output`). Content evicted past the window is forgotten unless persisted in external state/memory.
- **Parameters:** Learned numerical weights defining neural connections inside dense or Sparse MoE layers.
- **Temperature:** Sampling hyperparameter scaling logit probabilities before token selection. Low (`0-0.3`) concentrates probability mass on the likeliest tokens for extraction/classification (though GPU floating-point non-determinism and MoE batching mean even `0` is not strictly deterministic); high (`0.8-1.5`) flattens probabilities for creative brainstorming. *(Note: Several 2026 reasoning models lock sampling defaults and expose `thinking_level` or `effort` instead.)*
- **Fine-Tuning (SFT / LoRA):** Updating all weights or lightweight Low-Rank Adaptation (`LoRA`) adapter matrices on curated domain examples to specialize style, schema adherence, or task behavior.
- **SLMs (Small Language Models):** Compact, parameter-efficient models (typically 1B–30B active parameters, such as `Gemma 4` , `Gemini 3.5 Flash-Lite` , or `Gemini Nano`) engineered for sub-200ms latency, low serving cost, or on-device/edge execution.
- **MaaS (Model-as-a-Service):** Consuming open-weight or partner models (e.g., Claude, Llama, DeepSeek) via managed, per-token serverless cloud endpoints without provisioning dedicated GPU nodes.
- **`thinking_level` / Adaptive Thinking:** Inference-time System 2 compute controls— `thinking_level` (`MINIMAL` .. `HIGH`) + `thought_signatures` on Gemini, and `thinking: {type: "adaptive"}` + `output_config.effort` (`low` .. `max`) on Claude.
- **Custom GPTs / Gems / Projects:** Lightweight workspace wrappers combining persistent system instructions, curated reference files, and pre-bound tools on top of a foundation model.
- **Digital Twins:** Dynamic virtual simulations of physical assets, factories, or operational networks used for scenario testing and predictive maintenance.
- **Agent Capability Level (`L0` – `L4`):** Scoping ladder from a single model call (`L0`), tool-augmented call (`L1`), multi-step workflow/agent (`L2`), and multi-agent system (`L3`) to self-extending runtime systems (`L4`).
- **Orchestration Loop:** Harness runtime executing *gather context* → *decide* → *act* → *observe* while enforcing step budgets, `thought_signatures` , and guardrails.
- **Workflow vs. Agent:** Code-fixed step sequence (*workflow*) vs. model-directed runtime control flow (*agent*).
- **5-Layer Agentic Protocol Stack:** `MCP` (L1 tools/data), `A2A` (L2 peer agents), `MCP UI / AG-UI / A2UI` (L3 generative human UI), `Live API / Web RTC` (L4 real-time audio/video streams), and `AP2 + x402` (L5 cryptographic spend mandates & HTTP 402 micropayments).