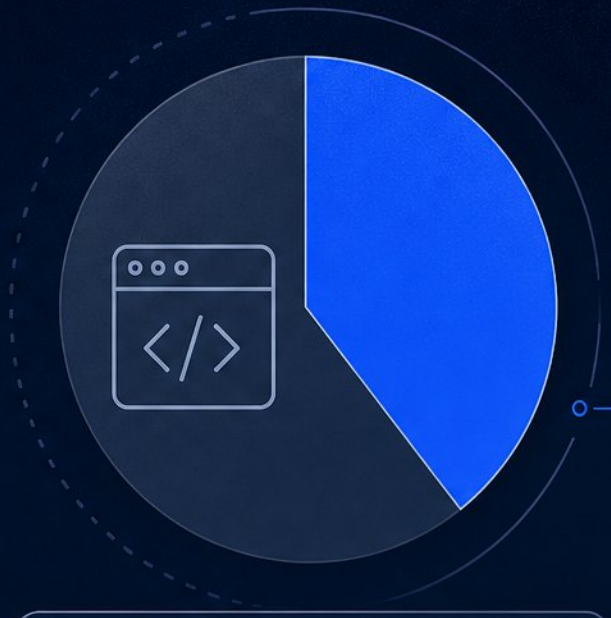
How to Build Applications Using Generative AI: A Practical Workflow

- Course goal: take a GenAI feature from idea to shippable prototype
- Seven stages: problem framing, model selection, prompts, retrieval, tools, evaluation, deployment
- Running demo: a document Q&A assistant with a task-copilot extension
- Built for developers, technical PMs, makers, educators, and prototyping teams
- Core mindset: probabilistic systems need iteration, measurement, and graceful failure
- Cost, privacy, and responsible AI are day-one design inputs

Background:
The 2026 Landscape You Are Building In

- Adoption is mainstream: ~40% of new enterprise apps ship AI features.
- Agents moved from experiment to production infrastructure, now often with write permissions.
- Two-thirds of engineering teams say cost shapes how ambitiously they use AI.
- Open-weight models rarely stand alone: 90%+ of users also call closed models.
- Bottleneck moved from writing code to reviewing, evaluating, and shipping it.
- Trust is falling as usage rises — verification is the scarce skill.

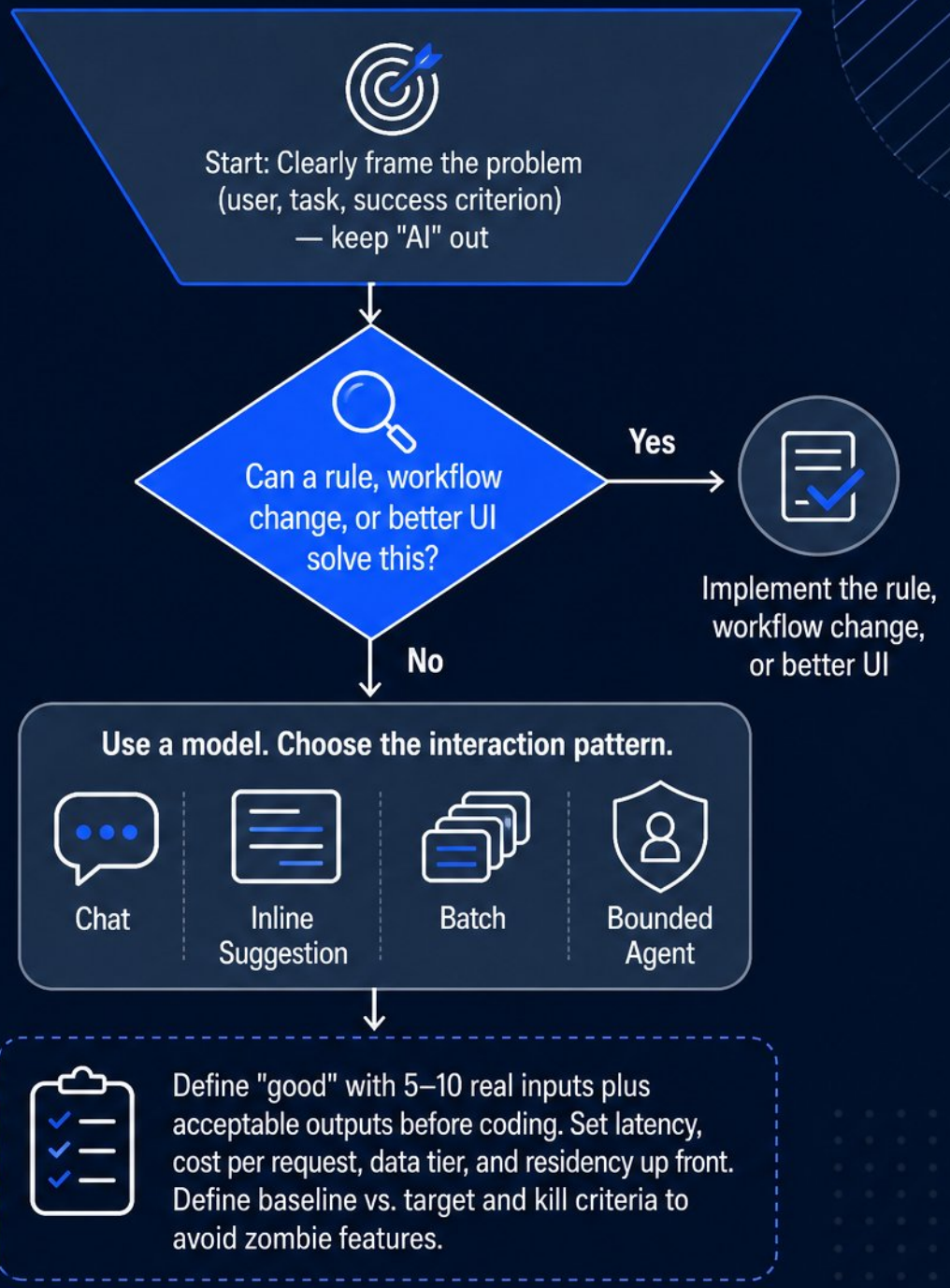


Core Concepts:
What Makes a GenAI App Different

- Traditional software specifies behavior; GenAI apps specify goals, then iterate toward acceptable accuracy
- Four failure modes: stale context, hallucination, format breakage, runaway cost or latency
- Three layers: model reasons, context grounds, execution enforces policy
- The model proposes; deterministic code decides what actually executes
- Three archetypes: copilot (assist), creator (generate), agent (delegate a bounded task)
- Shared vocabulary: tokens, context window, temperature, embedding, retrieval, tool call, eval, drift

Stage 1 — Frame the Problem Before Choosing a Model

- One sentence: user, task, success criterion — keep "AI" out
- Suitability check first: a rule, workflow change, or clearer UI beats a model
- 100% accuracy needs human review in the loop, not better prompting
- Pick the interaction pattern: chat, inline suggestion, batch, or bounded agent
- Define "good" with 5–10 real inputs plus acceptable outputs before coding
- Set latency, cost per request, data tier, and residency up front
- Define baseline vs. target and kill criteria to avoid zombie features



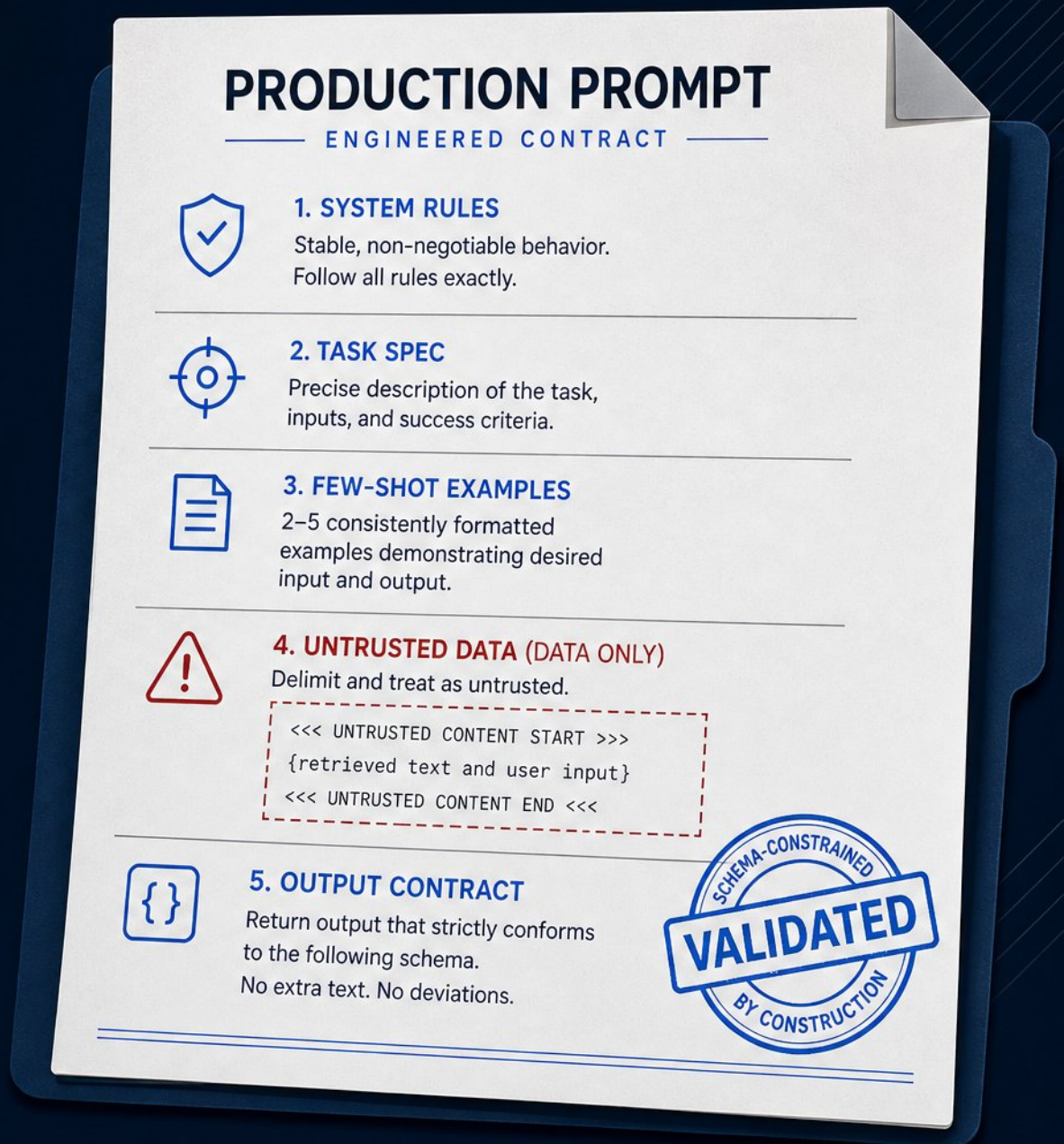
Stage 2 — Choose Models and Access Patterns

- - Model tiers (Sept 2026): ~$5–$10 flagships, $2 mid-tier, $0.20–$0.75 budget
- - Route by task: cheap models for classification, frontier only where evals justify 2.5x cost
- - Cost mechanics: output bills 5–6x input; cache reads ~90% off; batch APIs 50% off both
- - Watch pricing cliffs: long-context thresholds double input; promos expire Jan 1, 2027
- - Design a provider abstraction — a model gateway plus prompt templates
- - Treat providers like utilities: important, replaceable, benchmark against your own evals

	FRONTIER FLAGSHIP	CONTESTED MID TIER	BUDGET TIER
PRICE PER MILLION TOKEN	~$5–$10	$2	$0.20–$0.75
INPUT VERSUS OUTPUT COST RATIO	OUTPUT BILLS 5–6x INPUT	OUTPUT BILLS 5–6x INPUT	OUTPUT BILLS 5–6x INPUT
CACHE DISCOUNT MULTIPLIER	CACHE READS ~90% OFF	CACHE READS ~90% OFF	CACHE READS ~90% OFF
BATCH APIS DISCOUNT	50% OFF BOTH	50% OFF BOTH	50% OFF BOTH

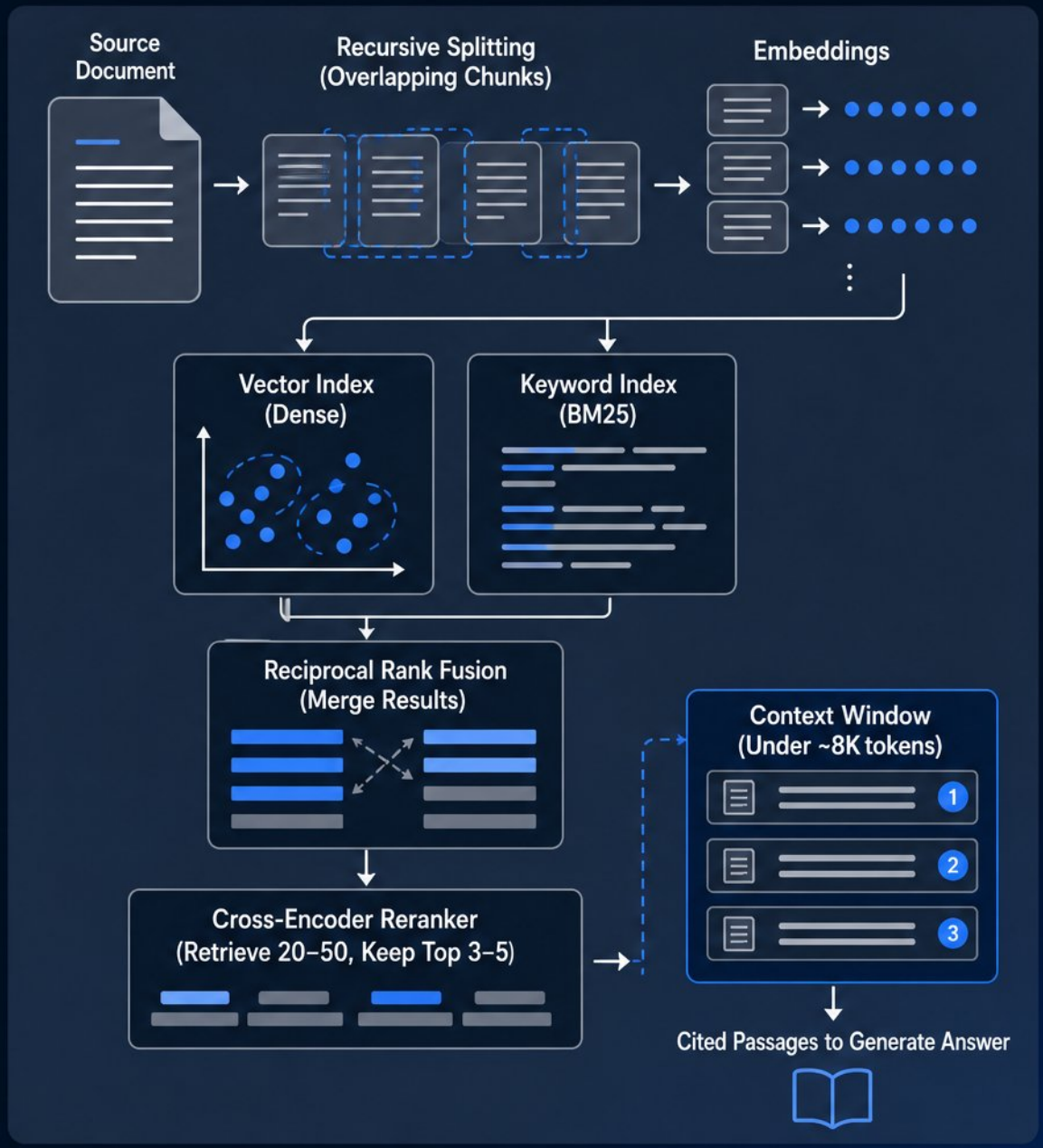
Stage 3 — Prompt Design That Survives Real Users

- Treat the prompt as an engineered contract: stable rules, task spec, output contract
- Constrain output with schema-constrained structured outputs — valid by construction
- 2–5 consistently formatted few-shot examples teach format faster than prose
- Fence untrusted content; retrieved text and user input are data, never commands
- Prompt injection is the top LLM vulnerability — secure in architecture, not wording
- Version prompts with model settings so changes are diffable and rollback-able
- Iterate from failing examples; add them to the eval set, don't rewrite from scratch



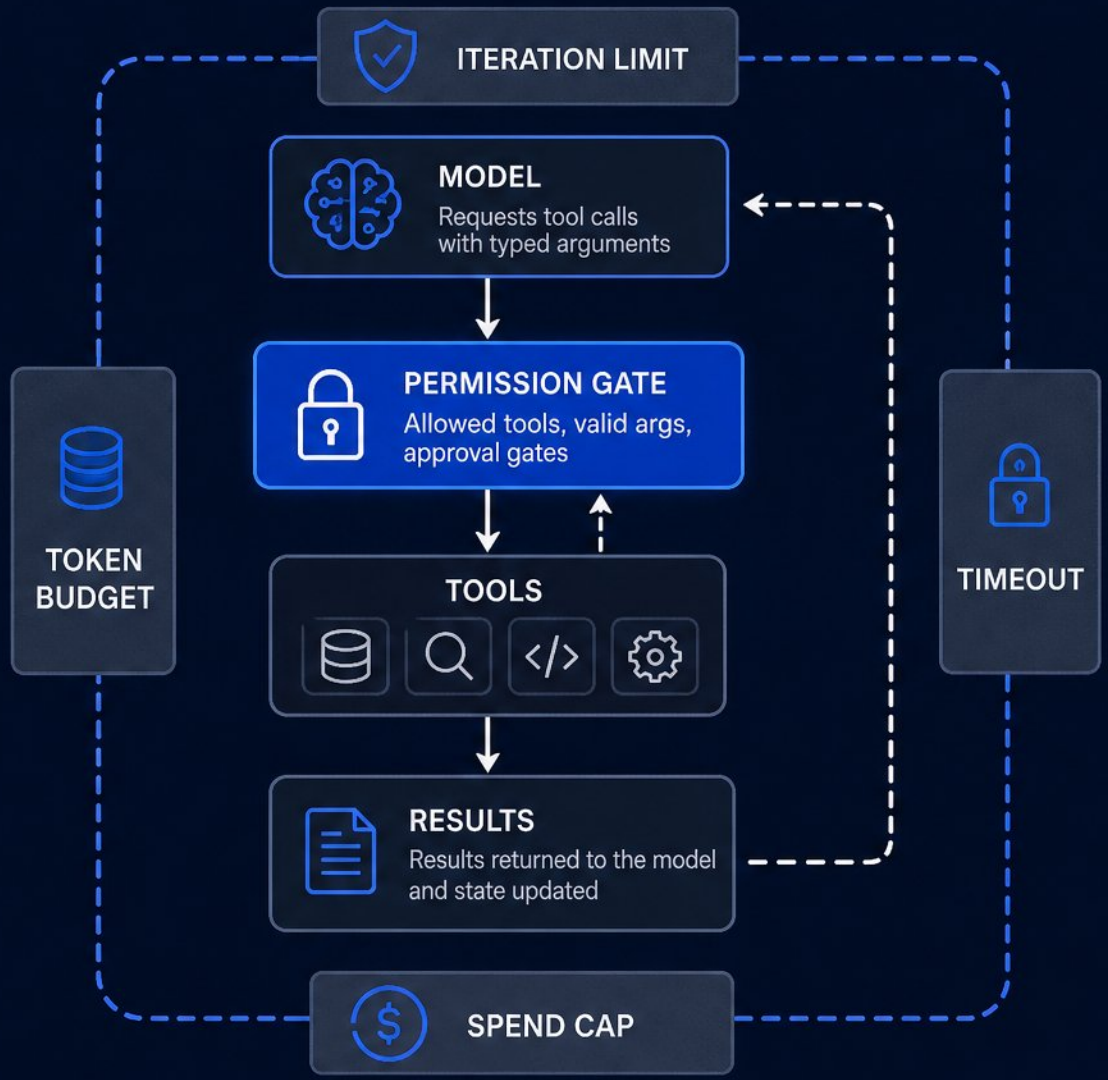
Stage 4 — Grounding with Retrieval and Context

- Retrieval quality is fixed at indexing time: chunking, embedding model, metadata
- Start with recursive splitting ~512 tokens, 10–20% overlap, then tune
- Parent-child chunking: small chunks to retrieve, larger parents to generate from
- Hybrid retrieval: vector plus BM25 merged with reciprocal rank fusion
- Cross-encoder reranker: retrieve 20–50, pass only top 3–5 chunks
- Ground answers with citations; keep context under ~8K tokens
- Evaluate retrieval separately: context precision, context recall, faithfulness



Stage 5 — Tools, Agents, and Orchestration

- Tool calling: the model requests data and actions through typed arguments
- Simplest pattern wins: single prompt → chain → routing → agentic loop
- Compile authority outside the model: allowed tools, valid args, approval gates
- Bound every loop: iteration limits, token budgets, spend caps, timeouts, termination
- Durable, resumable approvals for destructive or high-stakes actions
- Deliberate state: conversation, memory, or re-retrieve each turn
- Trace every prompt, tool call, argument, and result — debugger, eval source, audit log



Stage 6 — Evaluation: Catching Regressions Before Users Do

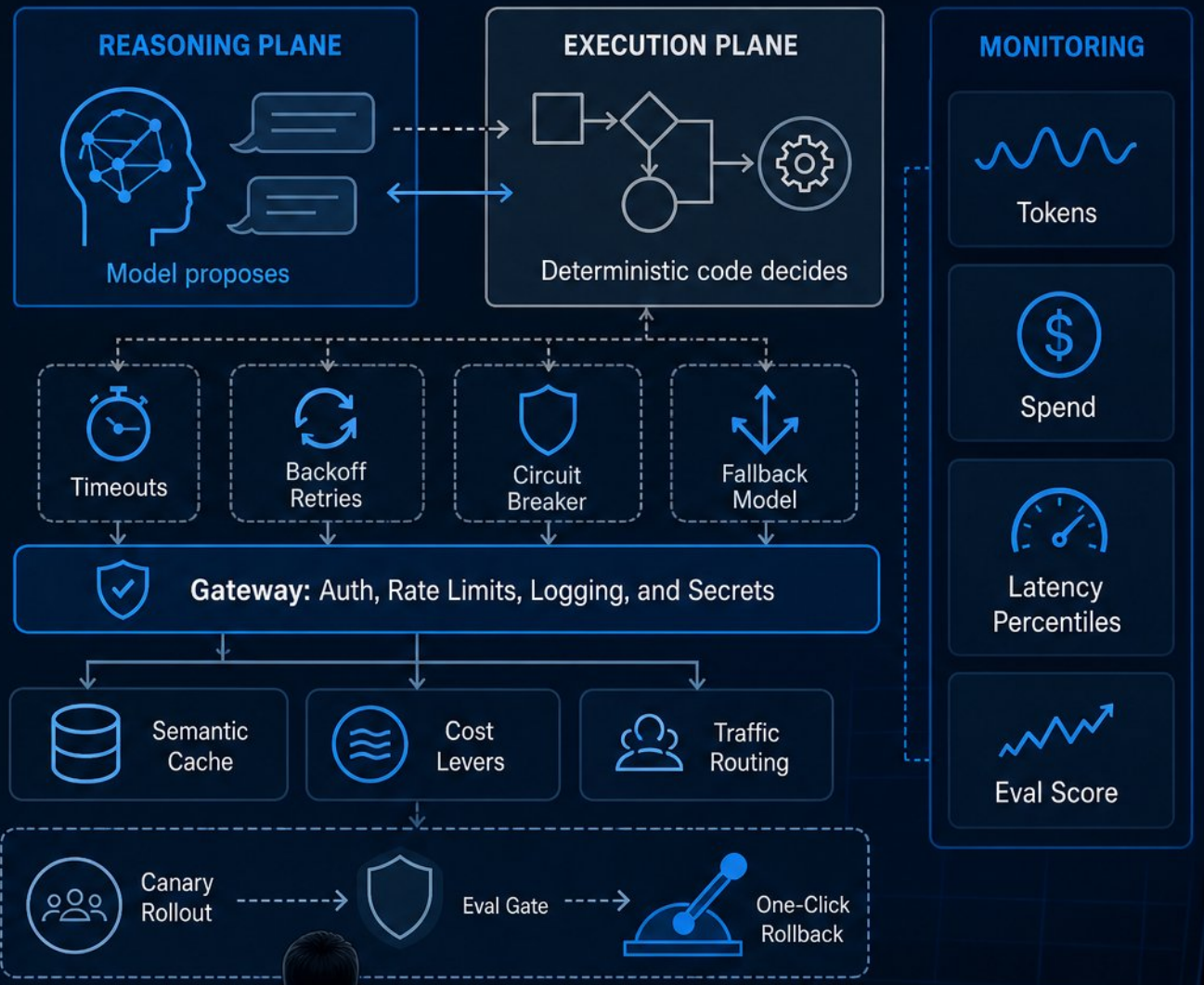
- - Start from failure modes, not a metric list
- - Build a golden set: 50 chosen cases beat 500 random
- - Layer checks: deterministic on 100%, judge on 5–10%, humans on flagged
- - Calibrate judges against human labels; measure with Cohen's kappa
- - Pin judge model and prompt; recheck monthly for drift
- - Gate merges on evals; every incident becomes a test

Judge biases to control:

- - Position — run both orders, aggregate
- - Verbosity, format, self-preference — blind, use a different model family

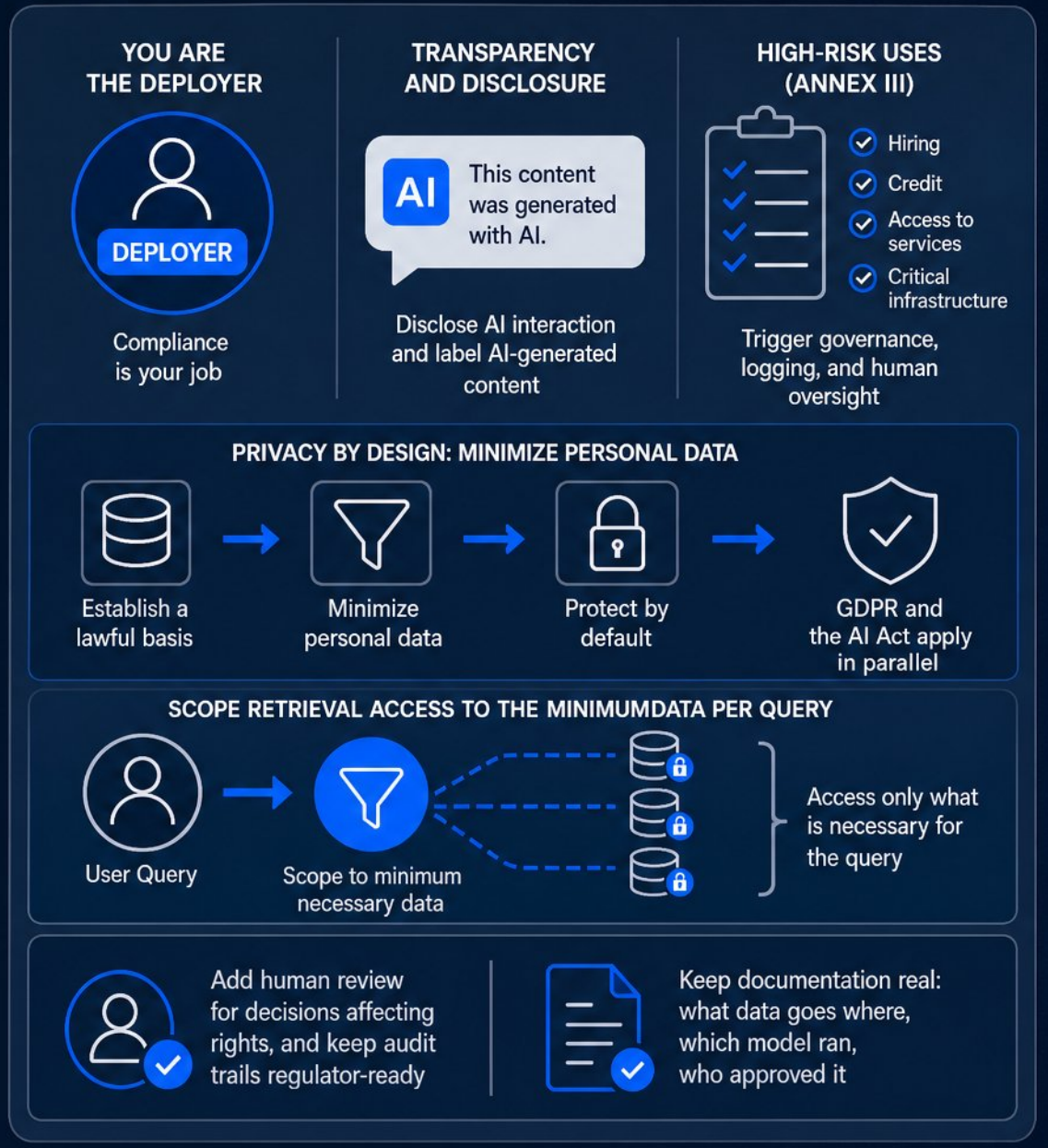
Stage 7 — Shipping, Cost Control, and Monitoring

- Model proposes, deterministic code decides: separate reasoning from execution planes
- Treat every model call as unreliable: timeouts, backoff retries, circuit breaker, fallback model
- Four cost levers: stream tokens, cache stable prefixes, trim prompts, route easy traffic
- Add a semantic cache — one deployment saw 31% hit rate, under 1% incorrect
- Deploy behind a gateway owning auth, rate limits, logging, and secrets
- Monitor tokens, spend, latency percentiles, and eval score to catch drift
- Roll out via canary with eval gate and one-click rollback



Responsible AI, Privacy, and Compliance in Practice

- - If you call a third-party model API, you are a deployer — compliance is your job
- - Disclose AI interaction and label AI-generated content; EU transparency rules in force
- - Annex III high-risk uses like hiring or credit trigger governance, logging, and human oversight
- - GDPR and the AI Act apply in parallel — establish a lawful basis, minimize personal data
- - Scope retrieval access to tlhe minimum data per query; it is a legal obligation
- - Add human review for decisions affecting rights, and keep audit trails regulator-ready
- - Keep documentation real: what data goes where, which model ran, who approved it



Hands-On Lab: The End-to-End Prototype Walkthrough

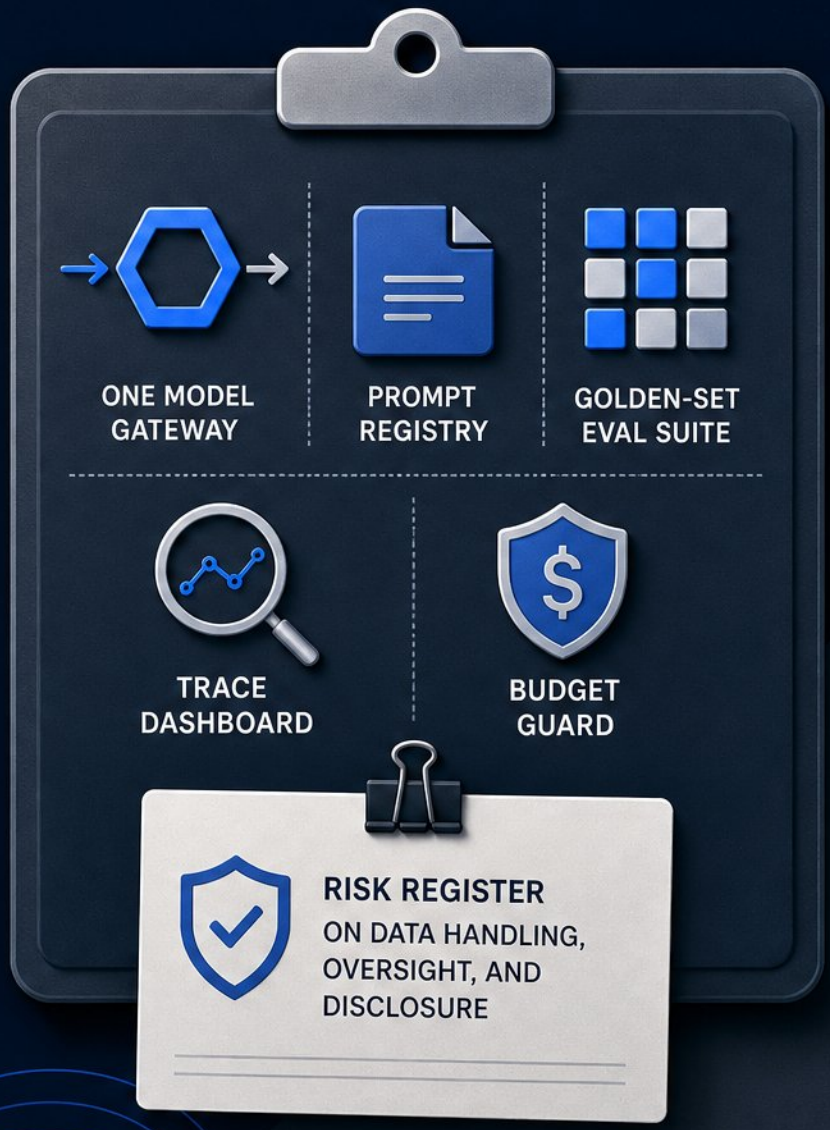
- Walk all seven stages on the running document Q&A demo
- Inspect real artifacts: problem statement, provider abstraction, versioned prompts, chunking choices
- Follow one live trace: query → hybrid retrieval → rerank → citations → approval gate
- Compare eval runs before and after a change; catch the regression on the scoreboard
- Review real incidents: confident answer with no sources, 15s timeout below real latency, a permission that should have been approval
- Reuse the seven-stage checklist as your starter template

Common Pitfalls and the Prototype-to-Product Gap

- Plausible output is not verified output — render provenance visibly.
- Verification, not generation, is where the schedule goes.
- Tiny repairs without shared context cause architectural drift.
- Security and cost are the classic blind spots; abuse becomes a bill.
- Cut scope by half, prove the risky assumption in a thin slice first.

Adapting the Workflow to Your Project and Next Steps

1. Write the problem statement before opening an editor this week
2. Choose first release by suitability test: narrow user, blocked workflow, honest failures
3. Set baseline now: 50-case golden set, cost-per-task and latency targets
4. Adopt compounding practices: structured outputs, versioned prompts, eval gates, kill switch
5. Starter kit: one model gateway, prompt registry, eval suite, trace dashboard, budget guard
6. Keep a short risk register on data handling, oversight, and disclosure



PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/7c698e6e/public