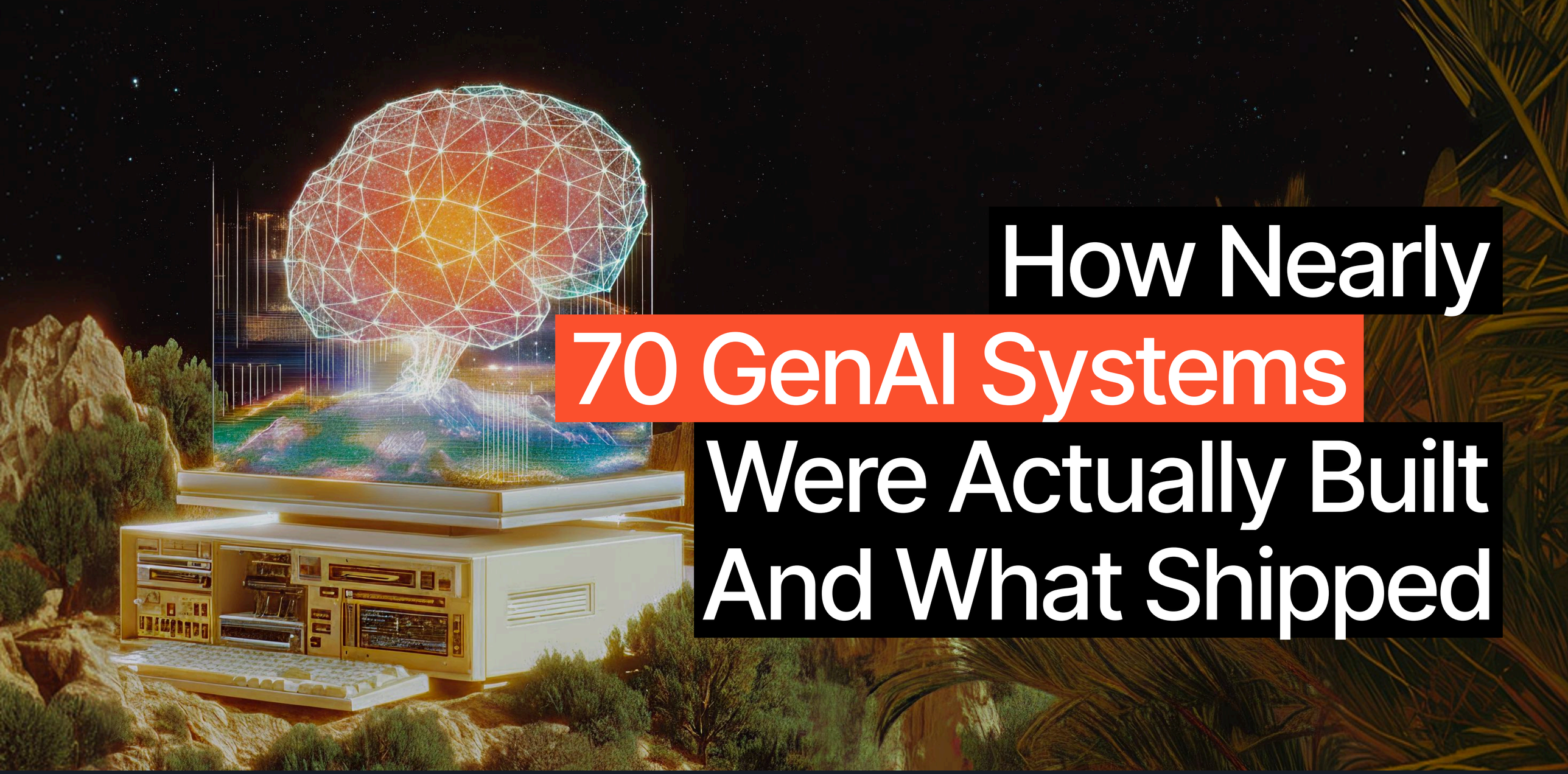


How Nearly 70 GenAI Systems Were Actually Built And What Shipped

Technical implementation guide
March 2026

Zürich
Switzerland

Artifact

aws

GenAI Zürich

A technical implementation guide for engineers, ML engineers, and solution architects — based on architecture reviews, stack analyses, and expert feedback from real-world GenAI systems across Switzerland and Europe

What We Found Under the Hood

Strip away the pitch decks and the business cases, and a pattern emerges from nearly 70 GenAI systems built across healthcare, insurance, agriculture, manufacturing, education, and public services: the systems that reached production invested more in what surrounds the model than in the model itself.

Consider an industrial sourcing platform that needed to match buyer requirements to supplier capabilities across hundreds of thousands of profiles. Generic semantic search returned too much noise.

They trained embeddings on industrial part taxonomies and built a semantic parser that decomposes nested Bills of Materials into machine-readable task graphs. Matching accuracy jumped from 25% to 70%, validated across multiple enterprise pilots. The investment wasn't a better foundation model — it was a domain-specific data layer that general-purpose AI couldn't replicate.

Or consider a pharmaceutical company that needed AI to talk directly to patients at risk of discontinuing treatment — a use case where a single wrong statement could cause real harm. Their solution: a primary conversational model that handles the dialogue, flanked by specialist safety models that monitor every response in real time and intervene before anything clinically inappropriate reaches the patient.

Think of it as a co-pilot architecture where the safety layer has veto power. The system was vetted by over 7,000 licensed clinicians before deployment. In testing, 100% of patients found the calls helpful and accurate. The breakthrough wasn't the model — it was the architectural decision to separate conversation from safety verification.

Production GenAI, it turns out, is 20% model and 80% everything around it — data pipelines, terminology layers, feedback loops, monitoring, and integration into existing workflows.

Start with your domain's taxonomy — the terms, hierarchies, and relationships that your industry uses but that general-purpose models don't understand.

Architecture Patterns That Actually Ship

Real-World Example

An oncology AI company building custom multimodal foundation models trained pathology, radiology, and genomics encoders from scratch, then combined them with LLM reasoning through patient representation embeddings. Their finding — state-of-the-art pathology performance with 100x less training data — was published at a top medical imaging conference. Deployed at a leading European cancer institute with sovereign infrastructure and ISO 27001 certification.

The moat isn't the model architecture; it's the clinical data pipeline and hospital deployment infrastructure that took years to build.

A sovereign AI platform for regulated Swiss enterprises illustrates the full stack: open-source inference, a document processing pipeline that preserves table structures and hierarchies where standard parsers fail, a vector store for retrieval, and a PII firewall scanning every prompt before it reaches the model. An "expert-asking pattern" detects knowledge gaps, escalates to human experts via Slack or Teams, and feeds verified answers back into the knowledge base — the organization's knowledge grows organically with every interaction.

What this means for you

Start with your domain's taxonomy — the terms, hierarchies, and relationships that your industry uses but that general-purpose models don't understand. Train or fine-tune embeddings on this domain data. Build a structured knowledge layer that sits between your raw data and the LLM. The teams that did this — whether in oncology, industrial procurement, insurance, or agricultural marketing — consistently reported the largest accuracy gains. Only then select the foundation model, and expect to swap it later.

The systems deployed in regulated industries shared a pattern that most projects missed entirely: validating outputs before generation, not after.

Pre-generation validation eliminates the trust problem

Real-World Example

An insurance group built a hybrid term-first architecture for multilingual corporate translation. A deterministic terminology layer fuzzy-matches input against a curated database and pins mandatory legal and brand terms before the LLM generates anything.

The model then operates within a stateful orchestration graph — style, verify, explain — with each step constrained. Thousands of production requests, sub-10-second turnaround, near-perfect success rates. Regulated content that legal teams trust, because the dangerous terms were never hallucinated in the first place.

The pattern extends beyond terminology. A pharmaceutical patient support system uses a constellation of specialist models that shape the conversation space continuously — not filtering outputs after generation, but constraining what the primary model can say in real time. A public sector platform limits GenAI to a supporting role while transparent, rule-based logic handles legally binding decisions.

What this means for you

Identify the terms, facts, or constraints in your domain where hallucination is unacceptable. Build a deterministic pre-generation layer that locks these before the LLM runs.

Post-generation filtering catches errors;
pre-generation validation prevents them.

The projects with the strongest adoption metrics didn't build standalone AI tools. They embedded AI into existing workflows so deeply that using the AI became the default path, not an optional add-on.

Deep workflow integration creates adoption — and defensibility

Real-World Example

A healthcare documentation startup built an ambient listening pipeline that feeds into report generation, billing code creation, and document analysis — all integrated into the consultation workflow from pre-visit to post-visit. Within 16 months: dozens of paying customers, hundreds of active users, and a physician testimonial noting the system had become part of daily practice.

The key architectural decision was horizontal and vertical integration across the entire consultation workflow, not a standalone transcription tool.

A digital commerce company integrated a RAG-based knowledge assistant directly into the chat application their customer service agents already used every day. Ticket resolution time dropped by multiple hours within two months.

The key decision: plugging the AI into the existing tool rather than building a separate interface. Adoption rose steadily because there was no context switch — the AI met users where they already worked.

What this means for you

Map your target user's existing workflow tool-by-tool.

Identify the moment where they currently context-switch to find information or make a decision. Place your AI at that exact point.
The systems that achieved the highest adoption all shared this pattern: zero workflow disruption.

The most technically ambitious systems in the dataset used multi-agent architectures. The ones that shipped kept each agent's scope ruthlessly narrow.

Multi-agent orchestration works — when scopes are narrow

Real-World Example

An agricultural marketing platform deployed five specialized agents orchestrated via LangGraph with Model Context Protocol connecting to live MarTech data sources. Each agent has one job: orchestrate, research, plan, create, or analyze. A gateway routes across multiple foundation models, making the system model-agnostic. In production across 10+ markets on four continents within months of MVP.

An enterprise AI assistant for a major ERP migration used multi-agent orchestration where specialized agents coordinate dynamically.
Its breakthrough capability: a Pathfinder agent that views the user's screen in real-time via vision models and provides context-aware step-by-step navigation through complex ERP interfaces.

On go-live day: over 1,100 unique users, 3,000+ real-time interactions, 100M+ tokens processed.

What this means for you

Design each agent with a single, well-defined capability.
Use an orchestrator agent to route — not to reason. Connect agents to live data sources via MCP or equivalent protocols rather than passing context through prompts. The systems that failed with multi-agent architectures tried to build general-purpose reasoning across agents; the ones that shipped gave each agent a narrow tool and let the orchestrator compose.

What actually shipped: the architecture stack

Where Most Teams Had Room to Improve

Missing eval pipelines

The single most frequent technical gap across the entire dataset: no systematic evaluation infrastructure. Teams built impressive systems but had no automated way to measure whether outputs were correct, consistent, or improving over time. Expert reviewers flagged this in the majority of projects — not as a minor gap, but as the primary technical risk.

The counter-examples stood out sharply. A translation platform ran a blind evaluation: professional translators assessed hundreds of segments across multiple language pairs, comparing their system against the market leader. The result — nearly twice as many perfect translations — was rigorous, reproducible, and became the centerpiece of their enterprise sales narrative. Their continuous improvement loop, where every human correction feeds back into client-specific models, means the eval data compounds: each customer interaction makes the system measurably better and harder to replicate.

Before your first deployment, define your eval suite. For RAG systems: retrieval precision, answer faithfulness, and hallucination rate, measured against a gold-standard dataset you build manually from your domain. For generation systems: blind human evaluation with disclosed methodology and sample sizes. For agents: task completion rate, tool-call accuracy, and escalation appropriateness. Automate these as CI/CD gates. The teams that had eval infrastructure could iterate faster, because they knew what was improving and what was regressing.

Define your eval suite

Thin API wrappers without proprietary data advantages

A recurring pattern: teams that built competent applications by orchestrating existing APIs but without any proprietary data layer, domain-specific training, or technical moat that couldn't be replicated in a weekend. The expert feedback was consistent: "What is genuinely novel about the AI elements versus standard recommendation algorithms?"

A voice-based health screening startup built the opposite: over a million voice checks powering model robustness across 25+ conditions, with fully owned IP including patents, CE medical device certification, and language-agnostic acoustic features that generalize across languages. The data flywheel — not the model — is the defensible asset.

Ask yourself: if a major model provider ships this feature tomorrow, does my system still have value? If the answer is no, you need a proprietary data layer — domain-specific embeddings, a knowledge graph encoding relationships that don't exist in public datasets, a human-in-the-loop feedback pipeline that compounds quality over time, or a regulatory compliance layer specific to your jurisdiction.

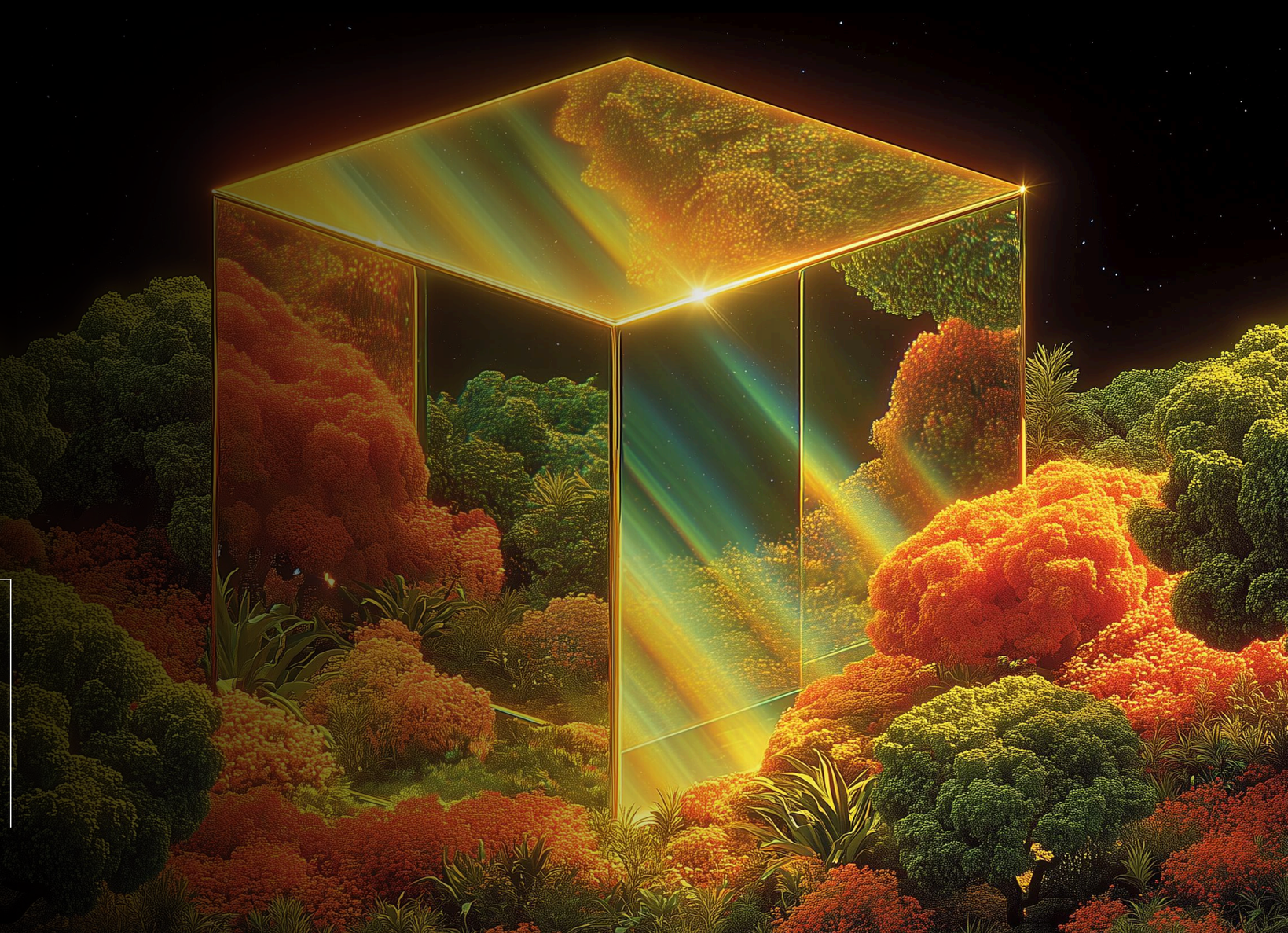
Ask yourself: does my system have value?

Governance as afterthought in regulated domains

Projects handling patient data, financial documents, and personal information frequently had no documented data flow diagrams, no explicit retention policies, and no audit trails — even when operating in industries where these are regulatory requirements.

The systems that got this right built governance into the architecture from day one. An insurance company processing legal cases built DSG[1] compliance, traceable outputs linked to source files, gold-standard evaluation sets, and data drift monitoring into the system before the first user touched it. Outputs are assistive; legal experts make all decisions. These weren't compliance checkboxes — they were architectural decisions that accelerated adoption because stakeholders trusted the system.

Three architectural components, implemented before your first pilot: a PII detection layer in the inference pipeline that sanitizes inputs before they reach the model, full trace observability logging every prompt, retrieval, and generation step, and a human-in-the-loop review interface that captures corrections as structured data feeding back into your eval suite.



Systems with built-in governance scale faster due to trust

[1] OpenAI, *The State of Enterprise AI*, 2025 Report

Platform ambition before proof-of-value

A pattern that cut across startups and enterprises alike: building broad, multi-capability platforms before proving that any single capability delivered measurable value. One project combined a knowledge graph, a digital twin, a podcast studio, and a data marketplace — all in a single platform. The expert feedback was direct: pick the single feature that generated the strongest reaction and build a focused go-to-market around it.

Ship one agent, one pipeline, one workflow — and instrument it to prove value before building the next. A digital marketing company's approach is the template: start with a single product, prove a 25% sales increase within three months, then use that evidence to justify the broader architecture. The speedboat earned the trust; the platform followed.

The Technical Checklist

From validate to scale

01
Validate

Define your eval suite before writing code.

Build a gold-standard dataset of 50–100 examples from your domain.

Establish baselines by measuring current human performance on the same tasks.

Identify the terms, facts, or constraints where hallucination is unacceptable and design your pre-generation validation layer.

02
Build

Invest in your domain data layer first: embeddings trained on your taxonomy, a knowledge graph encoding your domain relationships, a structured pipeline that preserves document hierarchy.

Build PII detection, trace observability, and human-in-the-loop review infrastructure before your first user touches the system. Design each agent with a single, narrow scope.

Embed into existing workflow tools rather than building standalone interfaces.

03
Ship

Deploy to one team, one workflow, one use case. Instrument everything: adoption, quality, efficiency, and cost.

Run your eval suite as a CI/CD gate on every model or prompt change.

Capture human corrections as structured training data.

04
Validate

Use measured outcomes from your first deployment to justify expansion.

Extract reusable components — the strongest enterprise in the dataset synthesized hundreds of use cases into a handful of reusable AI components, cutting development cost dramatically.

Add model-agnostic routing to avoid vendor lock-in. Monitor for data drift and quality regression continuously.

05
Start where you are

If you're starting fresh: build your eval dataset and baseline measurements.

If you're in pilot: add trace observability and a human review interface.

If you're in production: implement automated eval as a CI/CD gate.

If you're scaling: extract reusable components and add model-agnostic routing.

The checklist is a loop, not a ladder.

About This Guide

This guide was produced as part of the GenAI Zürich 2026 awards program. The analysis is based on nearly 70 project submissions across three award tracks — Rising Innovators, Impact Achievers, and Enterprise Transformers — each independently scored and reviewed by a panel of senior industry experts. All company names and identifying details have been anonymized.

How this guide was created

The analysis and initial drafting of this guide were produced using an agentic AI workflow, with expert jury members reviewing and commenting on each version. Their feedback was fed back into the process, making this a human-in-the-loop collaboration between AI and domain experts.

Kiro CLI, an agentic AI development tool by AWS, served as the orchestration environment. Custom Agent SOPs (Standard Operating Procedures from the Strands Agents open-source project) defined the end-to-end workflow — from ingesting all submissions and expert scores, to cross-referencing judge feedback, identifying patterns across tracks, and generating the final text.

The underlying model was Anthropic Claude (Opus 4.6).
For an introduction to this agentic tooling, visit goagentic.ch

Companion

This guide is the technical companion to the “GenAI Business Guide”, which covers where to invest, how to measure, and how to govern GenAI initiatives. The two guides share the same anonymized examples and pattern taxonomy; this guide goes deeper on architecture, data pipelines, and stack choices.

Expert jury

Marc Stampfli
Business Director Switzerland
@ NVIDIA

Stefan Ravizza
Co-Founder
@ Artifact

Dr. Efi Pylarinou
Global Fintech & Tech Thought Leader
@ GrowFin

Claudia Pletscher
Entrepreneur, Board Member & Innosuisse Innovation
@ Council

Sacha Ghiglione
AI & Strategy Executive

Paul Meyrat
AI Officer
@ Stadt Zürich

Anne-Marie Buzatu
Executive Director
@ ICT4Peace Foundation

Dr. sc. ETH Michael Hardegger
Lead Data Scientist
@ Digitec Galaxus AG

Dr. Sina Wulfmeyer
Chief Data Officer
@ Unique AI

Dr. Jan Kerschgens, EMBA
Innovation & DeepTech Leader in AI

Daniel Dobos
Research Director
@ Swisscom

Alex Dean
Manufacturing Practice Lead
@ Unit8

Rea Selina Wenk
CDO
@ CSS

Dr. Denis Samuylov
Co-Founder
@ GenAI Zürich

Expert jury and guide production

Christoph Schnidrig
Head of Technology
@Amazon Web Services
Orchestrated and iterated the agentic AI workflow that produced this guide

Michael Wegmüller
Co-Founder
@Artifact
Acted as lecturer and editorial advisor