<epam>

AI FACTORY HANDBOOK

MAY 13, 2025

Contents

1. Introduction

1.1. Purpose

The **AI Factory** is an organizational and technical framework designed to consistently deliver AI-driven products that generate business value, while effectively managing risks and optimizing costs. It integrates data, models, infrastructure, human expertise, and operational processes into a cohesive system, enabling the scalable development and operation of AI solutions.

The AI Factory concept is a natural evolution of the **Data Factory** paradigm – where data products are created through managed pipelines and quality controls. As AI relies fundamentally on data, the AI Factory builds upon these foundations, adding specialized capabilities for model lifecycle management, ethical oversight, and adaptive service delivery. It extends beyond technical integration to include clear roles, governance structures, and organizational alignment necessary to scale AI effectively.

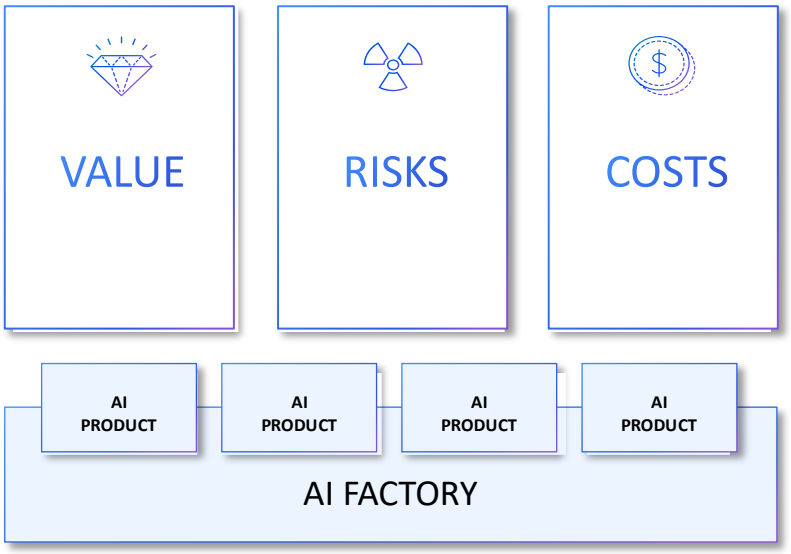


Figure 1. AI Factory Framework: Balancing Value, Risks, and Costs.

In essence, an AI Factory provides a framework to consistently deliver **AI products** (AI-enabled software services) while managing the value they generate, the risks they entail, and the costs of implementation. We recognize that AI initiatives must be evaluated across three critical dimensions: Value, Risk, and Cost.

- **Value:** AI-powered products are intended to drive tangible business value – increasing revenue, reducing costs, improving customer satisfaction, or creating new opportunities. The AI Factory framework enables faster iteration and more reliable delivery of AI solutions, allowing quick testing and refinement of approaches. (Comment: Clarified that the AI Factory enables value delivery, even if it doesn't directly create value on its own.)
- **Risk:** AI projects carry inherent risks, including data privacy concerns, algorithmic bias, model drift, security vulnerabilities, and regulatory compliance issues. The AI Factory framework emphasizes proactive risk identification, mitigation, and ongoing monitoring throughout the AI lifecycle. This includes establishing clear governance structures and ethical guidelines from the start.
- **Cost:** AI initiatives can be expensive, requiring significant investment in data infrastructure, talent, and compute resources. The AI Factory framework focuses on cost-efficient implementation via standardized processes, reusable components, and optimized resource allocation. Adopting the framework enables iterative, finely granular, data-driven management of both infrastructure and talent spending, ultimately promoting the culture of scalable and evolving AI operations.

While our aim is to deliver AI products that provide significant business value, the AI Factory's primary focus is on de-risking AI initiatives and optimizing their implementation costs. By addressing potential pitfalls early and streamlining development, we can confidently pursue AI opportunities while keeping investments under control and minimizing negative outcomes.

The outcome of a functioning AI Factory is a portfolio of AI products – software-based services comprising data, AI models, infrastructure, and user interfaces.

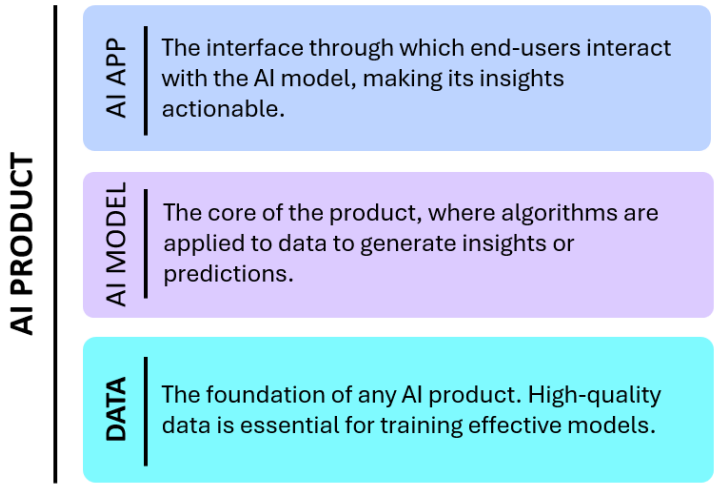


Figure 2. AI Product Structure.

These AI products can span multiple categories, and over time an AI Factory may expand from focusing on a single product category to a broader portfolio of services.

1.2. Target Audience

This handbook serves as a practical guide for EPAM professionals involved in the planning, execution, and oversight of building and operating AI platforms and/or developing AI-intensive products. It draws on collective experience and insights from numerous EPAM AI practitioners and projects, providing a common framework and vocabulary for AI initiatives.

The handbook is specifically geared toward the following roles, helping each to achieve success in AI Factory projects:

- Delivery Managers: Provides a framework for effectively managing AI projects and understanding the unique challenges of AI development. It helps in scoping projects, identifying and mitigating risks, and ensuring predictable timelines and cost-efficient resource allocation. It also aids in communicating progress and value to stakeholders.
- Solution Architects: Offers a blueprint for designing the foundation for robust, scalable AI solutions. It incorporates best practices for data processing, model development and deployment, operations and ongoing maintenance. Architects can use this handbook to get familiar with the common AI platform and AI product patterns, typical stakeholders and their concerns, make better informed decisions on the relevant system quality attributes, technology selection criteria, improve their enterprise context and integration points understanding.
- Technology Consultants: Equips consultants with knowledge and tools to advise clients on AI adoption strategies and to assess project feasibility using the AI Factory framework. It provides a common methodology for discussing AI initiatives with clients and internal teams, ensuring alignment and understanding across stakeholders.

- **Engineering Leads:** Serves as a guide for leading engineering teams in developing and deploying AI solutions. It outlines best practices for MLOps, generative AI techniques, and AI delivery processes, enabling leads to ensure code quality, maintainability, and adherence to standards throughout the project lifecycle.

1.3. Guiding Principles

Our AI Factory is guided by foundational principles that shape how teams collaborate, how technology is deployed, and how value is delivered. These principles ensure consistency across teams, alignment with business objectives, and scalability of AI capabilities across the enterprise.

1.3.1. Strategic Foundations

Defines the why and what of the AI Factory – ensuring business relevance, ethical integrity, and human-centered design.

1.3.1.1. Business Alignment

- Value-driven prioritization. Align all AI initiatives with measurable business outcomes.
- Transparent economics. Apply FinOps for cost visibility and continuous optimization.
- Outcome-first mindset. Focus on delivering impact, not just technical novelty.

1.3.1.2. Ethical & Responsible AI

- Responsible AI by default. Ensure fairness, interpretability, and safety from the start.
- Regulatory and values alignment. Comply with legal and ethical standards.
- Human augmentation over replacement. Enhance – not displace – human capabilities.
- Domain context preservation. Retain business context throughout the AI lifecycle.

1.3.2. Ways of Working

Describes how teams collaborate, iterate, and evolve AI solutions together.

1.3.2.1. People & Culture

- Skills diversity. Build cross-functional teams with technical and domain expertise.
- AI democratization. Enable self-service AI access for varying skill levels.
- Continuous learning culture. Promote ongoing development of AI literacy.

1.3.2.2. Agile Delivery

- Iterative delivery: Deliver value in small, focused increments.
- Adaptive planning: Adjust based on feedback and changing needs.
- Blueprint-guided execution: Align increments with the AI Factory vision.

1.3.3. Operational Excellence

Focuses on consistency, automation, and robust feedback in building and deploying AI solutions.

1.3.3.1. Operational Discipline

- Standardized workflows. Apply consistent processes across the AI lifecycle.
- Automation-first mindset. Minimize manual tasks through automation.
- Feedback amplification. Create tight loops from production to development.

1.3.4. Platform & Architecture

Defines the enabling capabilities and infrastructure of the AI Factory.

1.3.4.1. Technology Architecture

- Component modularity. Build interoperable, reusable capabilities.
- Scalability. Ensure horizontal scaling for community and demand.

- Vendor-agnostic design. Prefer open standards over proprietary lock-in.

1.3.4.2. Platform Capabilities

- DevOps
  - CI/CD. Automate build, test, and deployment across the Factory.
  - Infrastructure as Code (IaC). Manage with version-controlled configurations.
  - Observability. Monitor infrastructure and AI Product performance.
- DataOps
  - Data as a product. Define ownership, quality, and SLAs.
  - Metadata-driven architecture. Ensure discoverability and lineage.
  - Automated data quality. Validate data across the lifecycle.
- ModelOps
  - Model lifecycle governance. Govern from ideation to retirement.
  - Continuous training and evaluation. Monitor for drift and update models.
  - Reproducibility guarantee. Recreate experiments across environments.

1.3.5. Knowledge & Reusability

Captures how information is leveraged, reused, and shared within the AI Factory.

1.3.5.1. Knowledge-Driven AI

- Enterprise knowledge leverage: Capture and utilize institutional knowledge.
- Metadata and governance: Maintain traceability and discoverability.

1.3.5.2. FAIR Principles

- Ensure data and models are Findable, Accessible, Interoperable, and Reusable across the enterprise.

1.4. How to Use This Handbook

This handbook outlines a structured approach to building AI-intensive products, addressing the common challenges enterprises face in AI adoption. Historically, applied AI projects have suffered from unpredictable timelines, escalating costs, inconsistent quality, and unmanaged risks. These challenges often outweigh potential benefits, leading to project failures and hindering the realization of AI's strategic value. By following the AI Factory framework detailed in this handbook, readers can systematically de-risk AI projects and streamline their delivery. Each section of the handbook corresponds to critical components of an AI Factory operating model – from high-level strategy and business alignment to deep technical best practices. Readers are encouraged to use this document as a reference guide and toolkit:

- For planning: Identify required capabilities, deliverables, and checkpoints before starting an AI initiative.
- For execution: Apply the outlined frameworks (MLOps, generative AI techniques, validation strategies) to structure the development process.
- For governance: Leverage guiding principles and best practices to maintain quality, manage risk, and measure value throughout the project lifecycle.

By understanding the concepts and applying the recommendations herein, teams can improve coherence in their approach to AI projects and increase the likelihood of delivering AI solutions successfully, on time, and within budget.

Note: ***The AI field is evolving rapidly, particularly in technology domains such as large language models, cloud-based AI services, and automation frameworks. Readers of this handbook are strongly encouraged to conduct their own research and stay informed about the latest tools, standards, and breakthroughs.***

---

2. Understanding the AI Factory

An **AI Factory** can be visualized as the end-to-end system that enables an organization to reliably and efficiently develop, deploy, and manage AI products at scale. In this section, we walk through an example of building an AI solution, use it to illustrate evolving **value streams**, and then show how these value streams align with the **organizational capabilities** required for successful AI adoption.

2.1. Example of AI Product Creation

To ground the concept of an AI Factory in something tangible, let's consider a straightforward AI product example: a Customer Service Chatbot for automating routine support inquiries. This Chatbot:

1. **Retrieves FAQs and historical tickets** from a curated data source.
2. **Uses an AI model** – for instance, a fine-tuned LLM – to interpret user questions.
3. **Generates relevant responses** and routes complex cases to human agents.

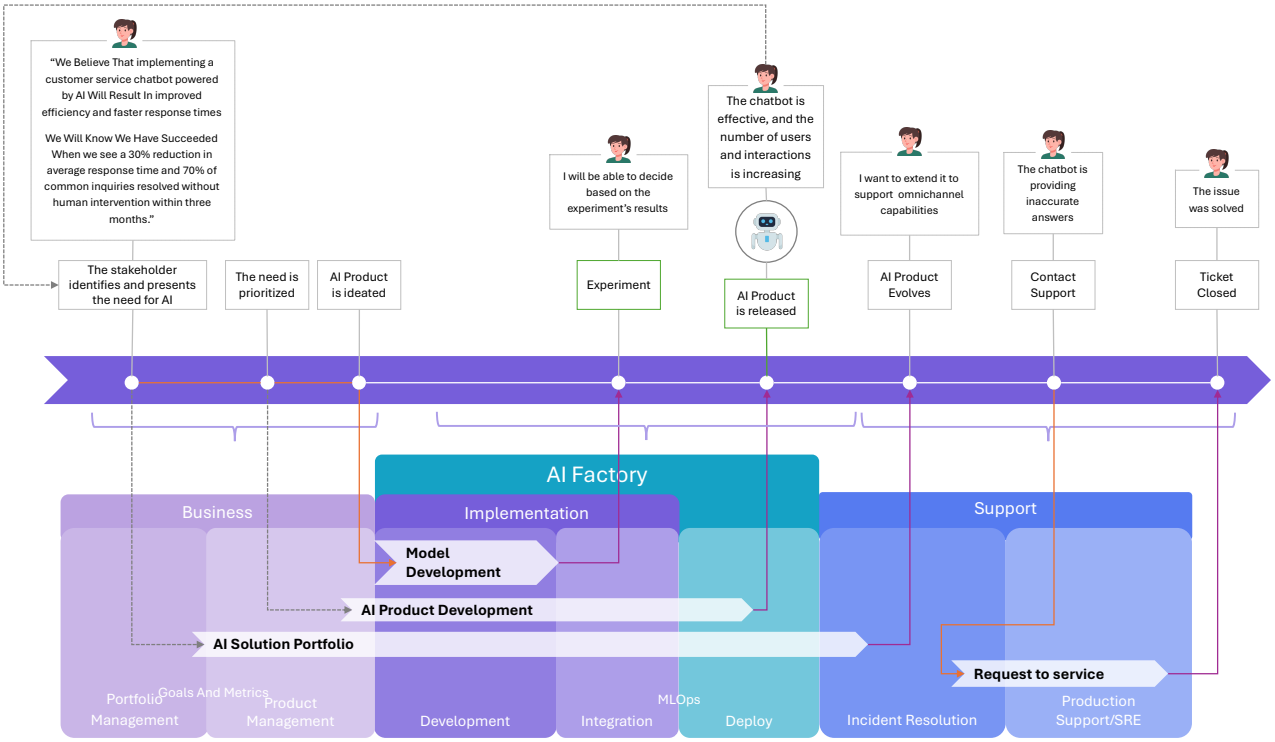


Figure 3. Stakeholders' value generation flow and Development Value Streams.

From ideation to deployment, building this AI product typically involves:

- **Model experimentation and training** (initial prototyping, data labeling, evaluating prototypes).
- **Productization** (packaging the model as a service, integrating it into existing customer support systems).
- **Continuous improvement** (monitoring live performance, adding new FAQs to the knowledge base, retraining the model as user needs evolve).

Although this example is relatively small in scope, it helps illustrate how technical components (data ingestion, AI model training, integration and monitoring) must align with business processes (budget, compliance, success metrics, user adoption) for the initiative to deliver real value.

2.2. AI Development Value Streams

Within an AI Factory, value streams are sequences of activities that transform ideas into tangible AI-driven outcomes. As organizations mature their AI capabilities, these value streams expand from **narrow, experimental model development** to full-blown **AI product creation** and finally to **managing a portfolio of multiple AI solutions**.

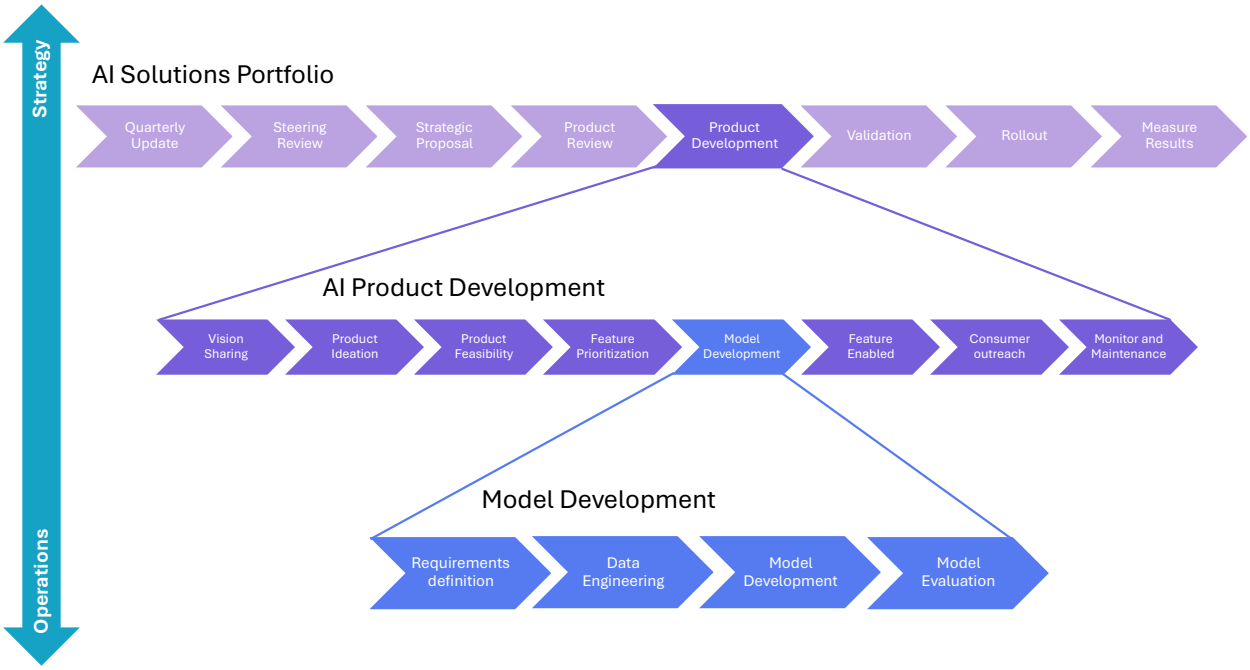


Figure 4. AI Development Value Stream progression.

1. Development Value Stream: Model Development

- Focus: Experimentation, prototyping, and feasibility checks.
- Outcome: A validated model or proof of concept ready to become part of an AI product.
- Example: Testing multiple chatbot ML architectures with historical tickets to confirm that a classification model can reliably route inquiries.

2. Development Value Stream: AI Product Development

- Focus: Taking a promising model and turning it into a stable, scalable product.
- Outcome: A production-grade AI service or application delivering measurable value.
- Example: Packaging the Chatbot model with user authentication, conversation flow logic, and analytics so it can be deployed into a customer support environment.

3. Development Value Stream: AI Solutions Portfolio

- Focus: Managing multiple AI solutions—often across different domains or business units—and managing them under a unified strategy.
- Outcome: A robust portfolio of AI products that provide organization-wide benefits, monitored and improved via standard governance.
- Example: Expanding beyond the Chatbot to include predictive maintenance, personalized marketing recommendations, and other AI products under a common governance, monitoring, and funding model.

Each of these value streams builds on the prior one. Early successes in **Model Development** feed into structured **AI Product Development**, which in turn becomes part of a broad **AI Solutions Portfolio**. Together, they reflect the growing maturity of both the technical platforms (data pipelines, MLOps, deployment) and the organization's processes (governance, budgeting, portfolio management).

2.3. Progressive Development of Organizational Capabilities

To succeed in any of these value streams, an enterprise must develop or refine a set of **organizational capabilities**. These go beyond technology and include decision-making, operating models, risk management, and more. The AI Factory Organizational Capability Map outlines the core competencies an organization needs.

Model Development

Engagement Stage

- Proof of Concept
- Experimental AI

Value

The ability to make decisions based on the experiment's results. The hypothesis can be confirmed, partially confirmed, or incorrect.

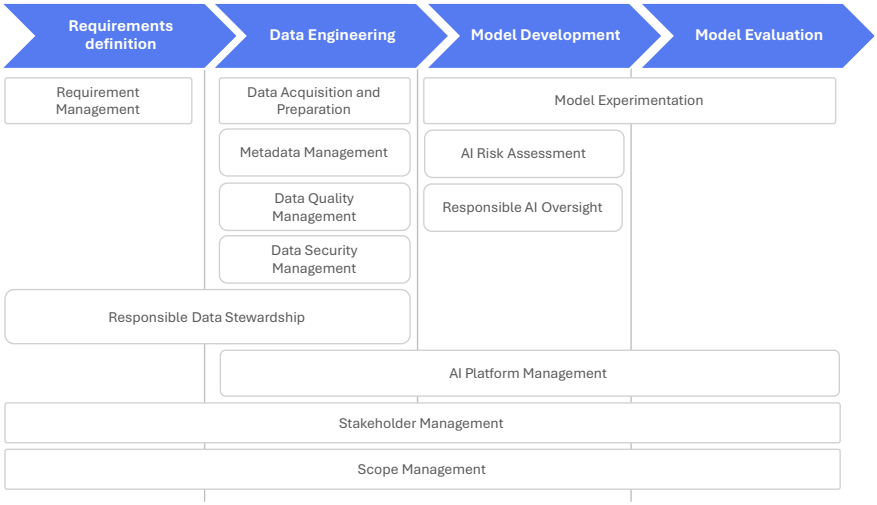


Figure 4. Model Development Organizational Capabilities.

AI Product Development

Engagement Stage

- MVP
- Single AI Product

Value

The AI product or new feature is ready to use and produces the measures required to evaluate the outcome.

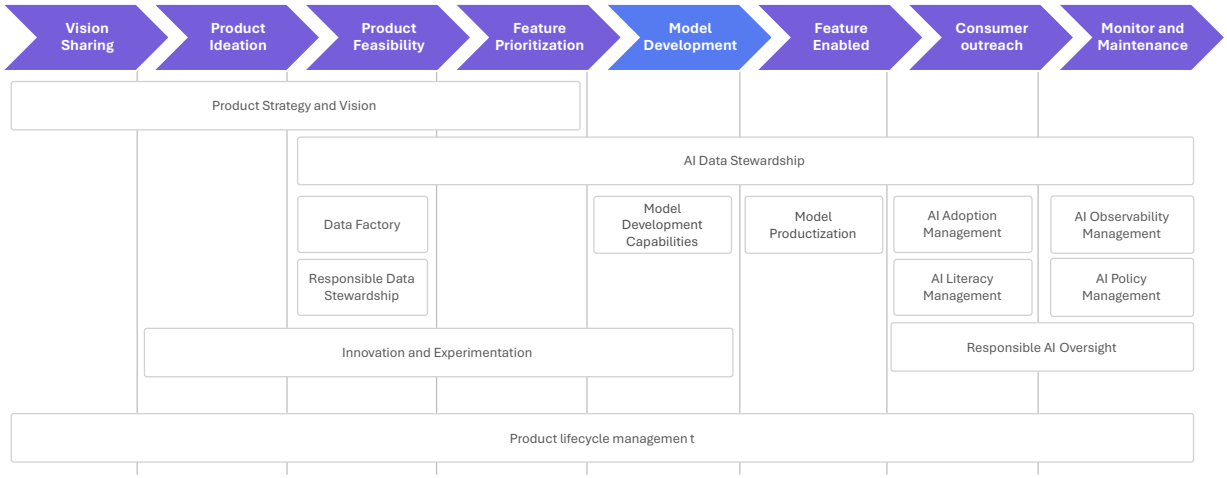


Figure 5. Product Development Organizational Capabilities.

AI Solutions Portfolio

Engagement Stage	Value
• Repeatable AI • Multiple AI Products • AI at Scale	Realization of the use cases hypothesis

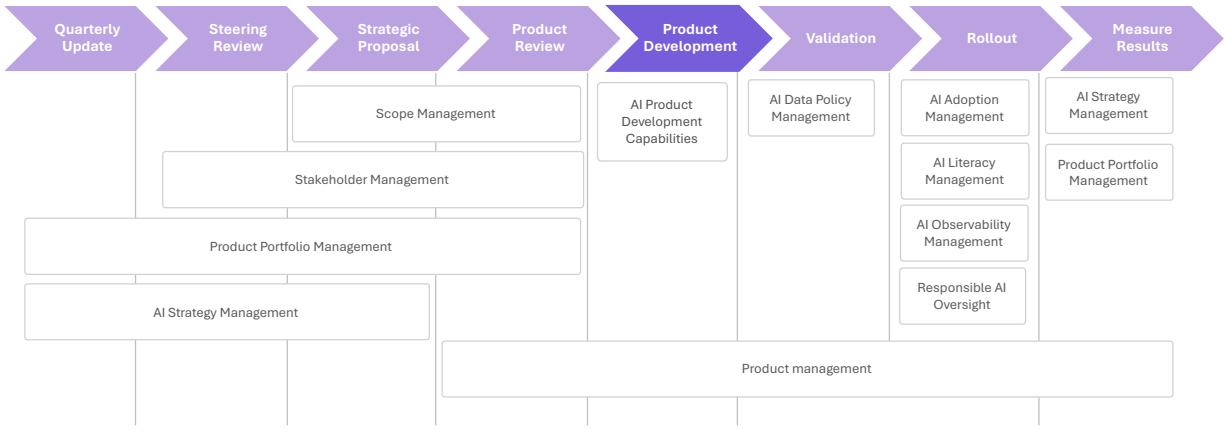


Figure 6. AI Solutions Portfolio Organizational Capabilities.

2.4. Organizational Capabilities Map

The capability map is a framework resulting from the grouping and rationalizing of organizational capabilities for each AI DVS level (See Figure 7). It will drive the definition of operational processes. As a framework, it aims to set up a baseline for the internal abilities and competencies required to implement AI products.

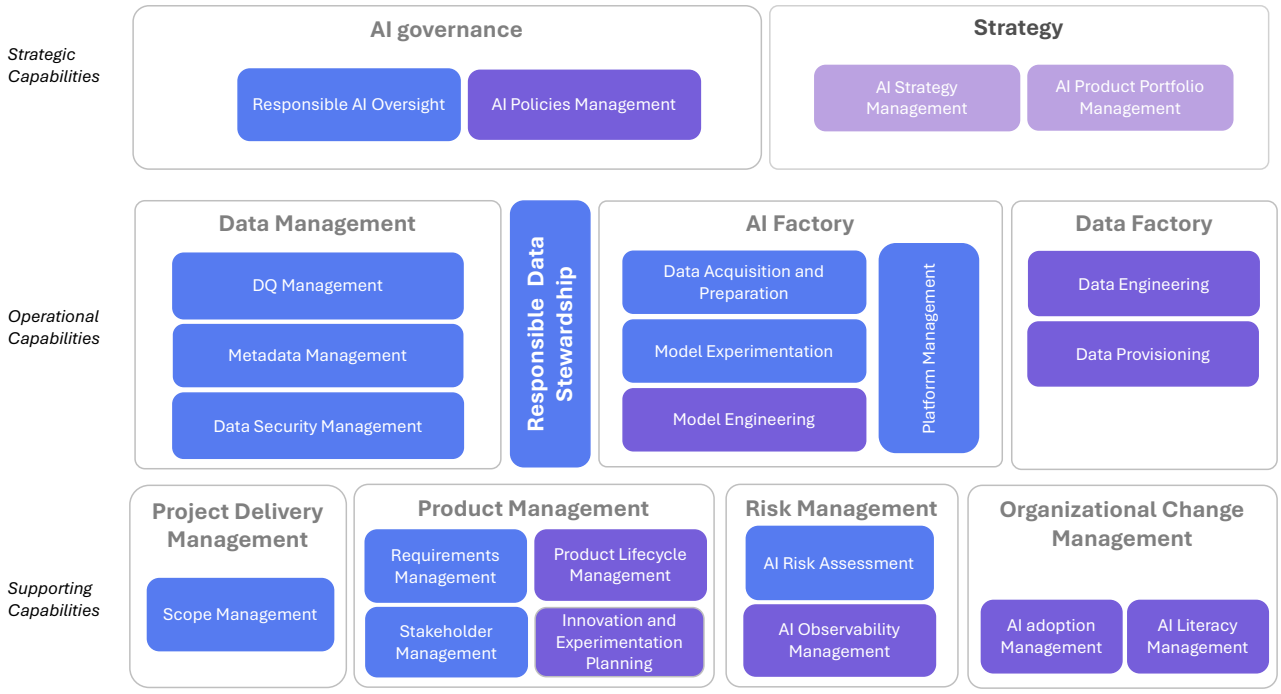


Figure 7. AI Factory Organizational Capability Map.

- Strategic: Capabilities needed for the long-term future and business scalability.
- Operational: Capabilities necessary for the primary value chain for AI products.

- Supporting: Capabilities required to operate or enable but not directly related to creating the value.

Below is a breakdown of these organizational capabilities:

Level 0	Level 1	Description
AI Governance	Responsible AI Oversight	Represents the ability to develop and enforce ethical guidelines, principles of fairness, transparency, and accountability for AI systems.
	AI Policy Management	Represents the ability to enact, enforce, and monitor policies and rules on the data used to train the models
Strategy	AI Strategy Management	The ability to develop and maintain the strategic plan and overall direction of the AI Factory.
	Product Portfolio Management	The ability to articulate a business problem, prioritize it, and ideate a feasible solution.
Data Management	Data Security and Privacy Management	The ability to ensure compliance with relevant regulations and safeguard information according sensitivity levels
	Data Quality Management	Implement AI model training and inference processes to ensure data quality, accuracy, completeness, and consistency.
	Metadata Management	The ability to provide business, technical, and operational context of the data assets.
Responsible Stewardship	Responsible Data Stewardship	The ability to ensure that data assets used to train models are collected, used, shared and disposed mitigating the ways that data can produce harm, and addressing how it can redress structural inequalities
AI Factory	Data Acquisition and Preparation	The ability to collect and deliver data for algorithm development and implementation. It considers cleaning and transformation necessary to ensure data quality and consistency.
	Model Experimenting	Capacity to evaluate, select, train, and test the AI model
	Model Productization	The ability to integrate and deploy the model ready for consumption by end users
	AI Platform Management	Capacity to develop and implement the architecture roadmap outlining the evolution of the AI platform
Data Factory	Data Provisioning	Represents the ability to create or acquire data in a repeatable manner
	Data Engineering	The ability to deploy data products
Organizational Change Management	AI Literacy Management	The ability to create a culture that critically evaluates AI, understands its capabilities and limitations, effectively communicates and collaborates with AI systems, uses AI as a tool in various contexts, and understands privacy and safety-related issues
	AI Adoption Management	The ability to integrate AI into the business workflow and operations to help with the tasks, make things easier, and drive innovation
Project Delivery Management	Scope Management	The ability to collect requirements and plan, define, and control the scope of a project
Product Management	Stakeholder Management	It represents the capacity to identify, organize, and communicate with stakeholders to manage their expectations.
Product Management	Innovation and Experimentation Planning	The ability to plan the experiments and measure the results
	Requirement Management	The ability of translating market and customer needs into clear, actionable requirements
	Product Lifecycle Management	The ability to manage a product from inception to retirement, including updates, feature enhancements, and changes
Risk Management	AI Observability Management	The ability to define and enforce standards to prevent the model degradation
	AI Risk Assessment	Represents the ability to identify AI-related risks and create a mitigation plan

2.5. Technology Blueprint

The AI Factory Technology Blueprint is a capability-driven reference architecture of the platform needed to build and run AI products at scale. It provides a structured, high-level view of the fundamental abilities (the "what") an organization must have to achieve its AI objectives. Figure 4 illustrates the blueprint's structure and legend.

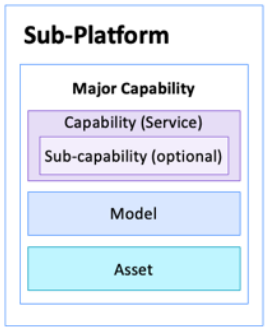


Figure 4. AI Factory Blueprint legend.

Below is the structure of the AI Factory blueprint from broad to detailed:

- **AI Factory (overall platform):** hosts all sub-platforms, capabilities, assets, and models that together deliver AI products.
- **Sub-platforms:** distinct functional areas within the AI Factory, each grouping related major capabilities.
- **Major Capabilities:** high-level groupings of functionality or domains critical to the platform's objectives (within a sub-platform).
- **Capabilities (Services):** specific services or functions under a major capability.
- **Sub-capabilities:** optional, more granular functions within a capability (if needed).
- **Assets:** data or resources (structured data, unstructured content, knowledge artifacts) produced or consumed by the platform.
- **Models:** AI/ML models developed and managed on the platform.

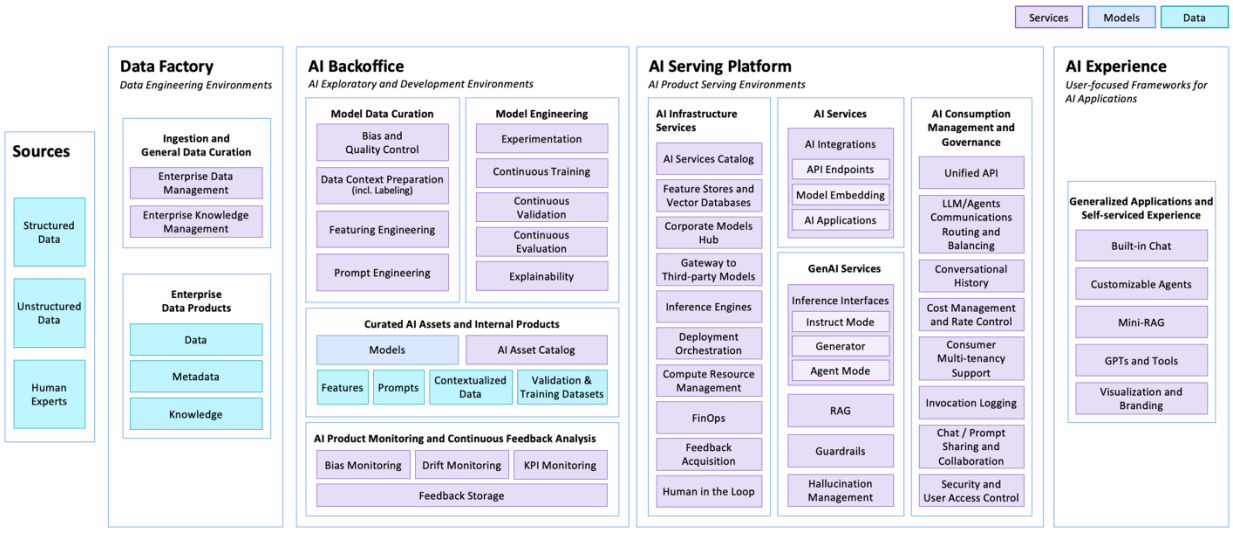
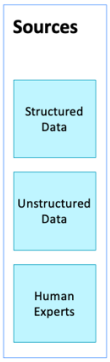


Figure 5. AI Factory Blueprint.

2.5.1. Sources

Utilizing various sources of data constitutes a fundamental capability within the AI Factory Platform, supplying the raw materials that fuel the entirety of the AI Factory's operations. The data extracted, processed, and plays a major role in shaping the training and performance of AI models. These sources can be either structured or unstructured, represent real-world observations or people's knowledge, and can originate from a variety of different environments. As such, the Sources of Data serve as the grounding bedrock of the AI Factory, supplying the essential information throughout the AI model lifecycle.



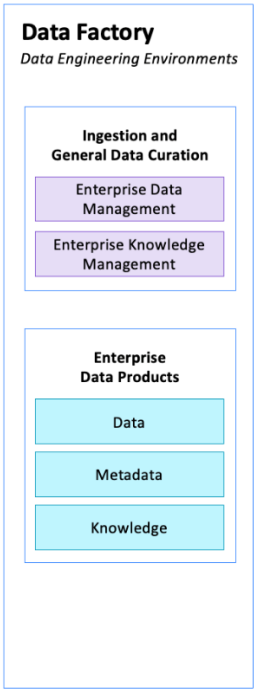
Structured Data sources such as data lakes, data marts based on relational databases or file-based data storage systems provide factual information in well-defined formats.

Unstructured Data presents a rich source of information in the form of text documents, images, audio, and more. Emerging largely from corporate knowledge bases and document repositories, these sources have recently become particularly relevant in the context of Language Model (LLM) utilization.

Human Experts—possessing specialized knowledge and experiential insights—form another vital data source. The knowledge obtained from them can be captured and stored in various forms such as texts, knowledge graphs, ontologies, or taxonomies. This human-centered data enriches the AI Factory Platform's ability to learn from, mimic complex human-like understanding and decision-making processes.

2.5.2. Data Factory

The Data Factory focuses on Data Engineering Environments, managing the flow of data and knowledge into the AI Factory. It ensures data meets data management standards and is packaged into Enterprise Data Products.



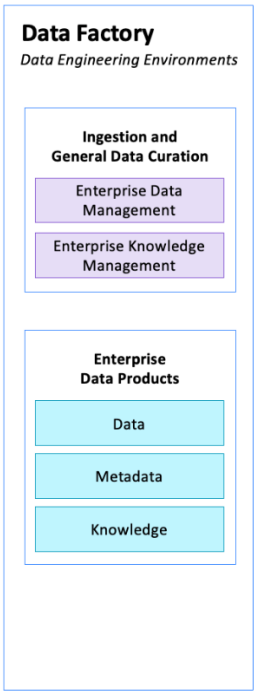
Ingestion and General Data Curation (Major Capability) focuses on bringing raw data into the platform, cleaning, standardizing, and ensuring its proper curation for further consumption within the downstream AI Backoffice. The outputs are stored As Enterprise Data Products.

Enterprise Data Management (Capability) includes a range of processes and tools designed to manage data effectively. Its focus areas include:

- Data Integration. Extracting and combining data from different sources to provide a view of data supporting business functions e.g., data processing, data warehousing, data lakes.
- Data Governance. Definition of policies and procedures to ensure data quality, consistency, and security.
- Data Quality Management. Profiling, cleansing, and monitoring data to ensure accuracy, completeness, reliability, and relevance.
- Metadata Management. Collecting and managing "data about data," such as definitions, lineage, and relationships.
- Data Security and Privacy. Protecting data from unauthorized access and ensuring compliance with data privacy regulations.
- Big Data Management. Handling large data volumes with scalable storage, distributed computing, and big data frameworks.
- Data Lifecycle Management. Managing data from creation to deletion, ensuring proper archiving, retention, and disposal according to regulations.

Enterprise Knowledge Management (Capability) includes processes and tools to capture, store, manage, and share knowledge. Focus areas include:

- **Knowledge Capture and Codification.** Identifying and documenting tacit knowledge from human experts and converting it into explicit knowledge.
- **Knowledge Repositories.** Implementing centralized storage for knowledge assets.
- **Search and Retrieval.** Enabling quick, easy access to knowledge through indexing and search engines.
- **Content Management.** Organizing and managing content throughout its lifecycle including version control, approval workflows, content curation.



The **Enterprise Data Products (Major Capability)** is a set of value generating information assets—namely Data, Metadata, and Knowledge—each structured and maintained according to enterprise quality standards. Groups of curated assets owned by specific business domains are considered Enterprise Data Products, serving as trustworthy, interoperable sources for the AI Backoffice (model training/validation) and for additional context in the AI Serving Platform.

Data (Asset)

- Structured or semi-structured data sets.

Metadata (Asset)

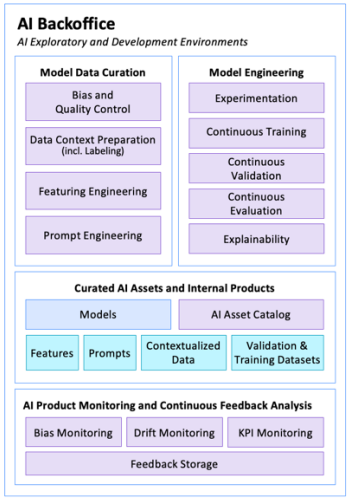
- "Data about data": cataloging information, lineage, ownership details.

Knowledge (Asset)

Domain-specific or expert-driven unstructured insights.

2.5.3. AI Backoffice

The **AI Backoffice** provides Exploratory and Development Environments for AI, primarily for model training, validation, and curation of AI-related assets. This is the primary space for hosting analytical community of Data Scientists, Machine Learning Engineers, Data Analysts, and Quality Assuring Engineers.



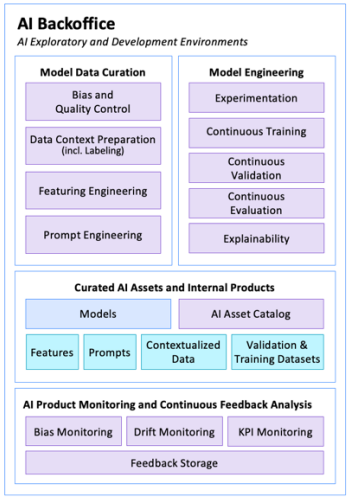
Model Data Curation (Major Capability) provides the back office with the services for taking the Enterprise Data Products from the Data Factory and applying context and feature engineering techniques. Produced features, contextualized data, LLM prompts – supply both AI engineering and serving as *Curated AI Assets and Internal Products*.

Bias and Quality Control (Capability) facilitates detection and elimination of bias in data, addressing target-population misrepresentation, measurement inaccuracies, or historical inequalities. By employing data cleansing, augmentation, and rigorous validation checks, this capability ensures data integrity and fairness for AI models.

Data Context Preparation (Capability) ensures data is properly understood (by both models and human experts), structured, and enriched with necessary contextual information to make it suitable for AI model training, evaluation, and validation. Includes labeling tasks and the injection of additional context as needed.

Feature Engineering (Capability) enables creating, transforming, and selecting variables (features) to boost model performance. Leverages domain knowledge and data-manipulation techniques to generate or refine features, making them more relevant and predictive for downstream modelling and model serving.

Prompt Engineering (Capability) supports the creation, testing, and optimization of input prompts for LLMs. Ensures that language models produce high-quality, contextually accurate outputs aligned with specific business objectives.



Model Engineering (Major Capability) encompasses the end-to-end model development cycle, including experiments, training, and evaluation, and provides tooling for continuous delivery for artificial intelligence models.

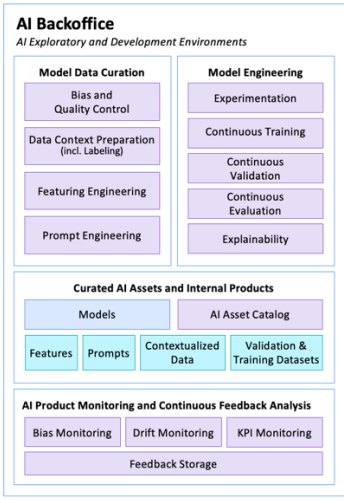
Experimentation (Capability) supports trying out different model architectures, hyperparameters, and feature and label sets. Tracks model experiments to identify best-performing approaches.

Continuous Training (Capability) automates recurrent or streaming data updates for models, ensuring they remain current over time.

Continuous Validation (Capability) executes validation checks with new or hold-out data sets. Allows engineering of performance degradation or concept drift detection.

Continuous Evaluation (Capability) allows implementation of monitors for model metrics e.g., precision, recall, F1, or business KPIs in near real time. Feeds back into new experimentation cycles as needed.

Explainability (Capability) generates insights into how models arrive at predictions (SHAP, LIME, attention maps). Vital for regulated industries or high-stake decisions.



Curated AI Assets and Internal Products (Major Capability) a range of AI-ready Assets and Models for both internal consumption and end-user serving.

Models (Model) Machine Learning and Deep Learning models, fine-tuned LLMs, or specialized AI components like models. Versioned, with associated metadata about training runs and performance metrics.

AI Asset Catalog (Capability) registry for all available models, feature sets, curated data subsets, and prompt templates.

Features (Asset) reusable feature sets.

Prompts (Asset) reusable templates or instructions for Generative AI tasks..

Contextualized Data (Asset) domain-specific or scenario-specific subsets of data ready for modeling (e.g., labels).

Validation & Training Datasets (Asset) curated data sets used for iterative training/validation in the AI Backoffice.

AI Product Monitoring and Continuous Feedback Analysis (Major Capability) captures performance, bias, or drift signals from production, feeding back into new development cycles.

Bias Monitoring (Capability) watches for emerging biases in production predictions

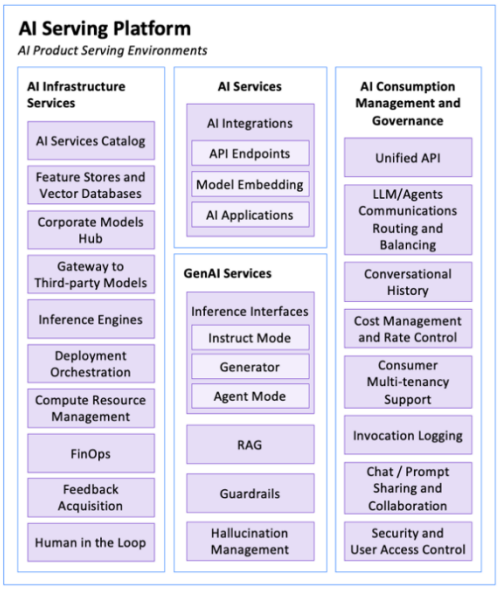
Drift Monitoring (Capability) detects data-distribution changes or concept drift that degrade model accuracy.

KPI Monitoring (Capability) tacks business/operational metrics to measure the real-world impact of model outputs.

Feedback Storage (Capability) centralized repository for user or system feedback, which in turn can retrigger model updates.

2.5.4. AI Serving Platform

External AI Product Serving Environments, focusing on deployment, scalability, and runtime management of AI solutions.



AI Infrastructure Services (Major Capability) support smooth deployment, management, and scaling of AI models curated in the AI Backoffice. The major capability fosters continuous feedback loops for model enhancement and adaptation, ensuring efficient performance of AI assets in production. Enabler for continuous delivery.

AI Services Catalog (Capability) central directory of AI microservices, facilitating reuse and discoverability.

Feature Stores and Vector Databases (Capability) provide online/offline feature storage for consistent feature retrieval including vector databases for semantic search and nearest-neighbor queries (e.g., for LLM embeddings).

Corporate Models Hub (Capability) enforces version control, security, and compliance checks, serves as a central location for approved or production-ready models.

Gateway to Third-party Models (Capability) orchestrates inference requests to external providers (e.g., public LLM APIs), manages authentication, usage metering, and rate limiting.

Inference Engines (Capability) provide execution for real-time or batch scoring, cost and performance optimizations for throughput and latency with GPU/TPU/other acceleration options.

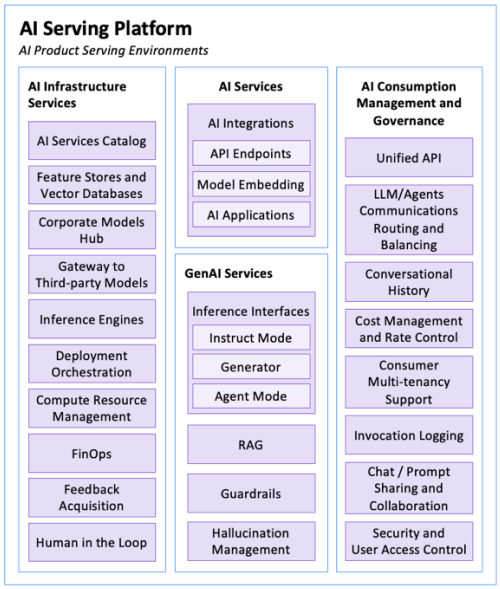
Deployment Orchestration (Capability) is the handling of the last component of Continuous Delivery for ML (alongside with CT from Backoffice). The automated CI/CD pipelines for containerized AI services. Handles canary or blue-green deployments of new model versions.

Compute Resource Management (Capability) provisioning and scaling of compute clusters (Kubernetes, Apache Spark). Control over hardware utilization and cost on the enterprise scale of the development and production.

FinOps (Capability) culture enables financial governance for AI workloads (cost transparency, chargebacks), ensures budgets and usage constraints are respected.

Feedback Acquisition (Capability) mechanisms to capture real-time usage data, user feedback critical for iterative improvement.

Human in the Loop (Capability) mechanisms that route uncertain model decisions to human reviewers for final adjudication (or labeling).



AI Services (Major Capability) offers various integration avenues for AI models, leveraging *Shared AI Infrastructure Services*. It acts as the conduit between AI models and real-world applications, ensuring accessibility, discoverability, and operational consistency of AI services while providing various *AI Integration* Modes.

AI Integrations (Capability) enables AI model to be served and used in production:

- API Endpoints. AI model is wrapped in a service that can be deployed independently of the consuming applications. Under the hood the service can do runtime predictions on demand or serve predictions pre-calculated on schedule, event trigger, or new model release.
- Model Embedding. AI model artifact is a dependency that is ready to be built and packaged within a software application or service.
- AI Applications. A mode of serving a full-stack web application (or similar) with built-in AI model within the AI Factory.

GenAI Services (Major Capability) address Generative AI models specifics.

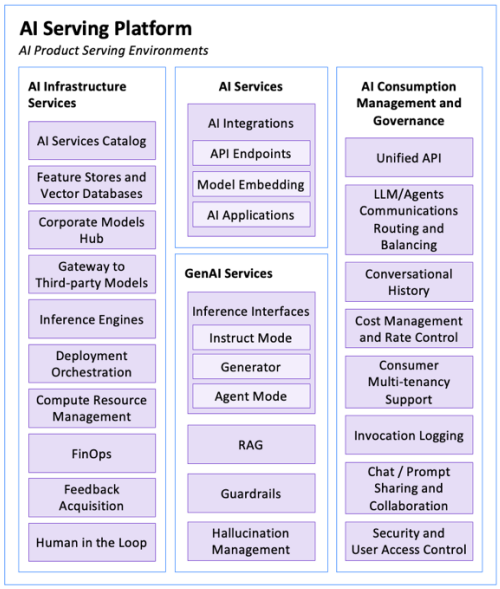
Inference Interfaces (Capability) offer a versatile inference interface with multiple consumption modes:

- Instruct. Single-turn direct Q&A, instructions, or clarifications.
- Generator. Open-ended or creative text production.
- Agent. Multi-step, tool-using, or API-calling orchestration.

RAG (Capability) combines generative AI model with real-time data retrieval.

Guardrails (Capability) help to filter or transform model outputs.

Hallucination Management (Capability) detects and mitigates incorrect or fabricated responses.



AI Consumption Management and Governance (Major Capability) primarily facilitates and governs interactions between AI models and customer-facing applications, offering extensive management controls. Ensures seamless information flow, cost management, security, and compliance.

Unified API (Capability) provides a single point of entry for multiple AI services, enabling standardized request routing and versioning.

LLM/Agents Communications Routing and Balancing (Capability) can dynamically direct requests to the most suitable LLM provider, self-hosted instance, or agent, optimizing performance and cost, and providing higher service availability.

Conversation History (Capability) maintains session context for chatbots, ensuring continuity of complex multi-turn conversations.

Cost Management and Rate Control (Capability) implements per-user or per-application usage throttling, cost attribution, and quotas.

Consumer Multi-tenancy Support (Capability) separates data, usage policies, and resources across different business units or external customers.

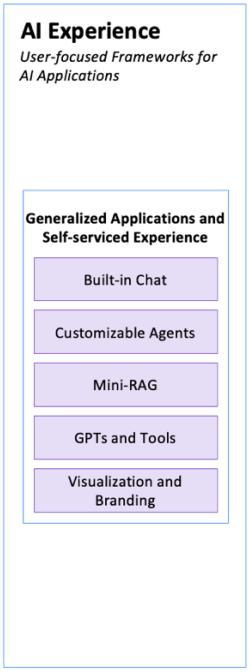
Invocation Logging (Capability) records calls to AI services for audit, debugging, or analytics.

Chat / Prompt Sharing and Collaboration (Capability) allows teams of end-users to share curated prompts or conversation transcripts safely, with version tracking.

Security and User Access Control (Capability) provides authentication, authorization, and encryption for all AI service endpoints, integration with enterprise identity providers (SSO, LDAP, OAuth)

2.5.5. AI Experience

Focused on End-User Experience (UX), User-Centered Frameworks for consuming AI—via web interfaces, chatbots, agent builders, or integrations with advanced analytics portals.



Generalized Applications & Self-Serviced Experience (Major Capability)

Built-in Chat (Capability) provides native to the Enterprise chat applications (Web, MS Teams, Slack etc.) that allow real-time interaction with LLMs or specialized models.

Customizable Agents (Capability) enable user-friendly tools to create or tailor domain-specific AI agents with minimal coding.

Mini-RAG (Capability) lightweight retrieval-augmented generation frameworks for smaller no-code use cases or prototypes.

GPTs and Tools (Capability) pre-integrated toolsets that harness GPT-style models for summary, classification, or creative tasks.

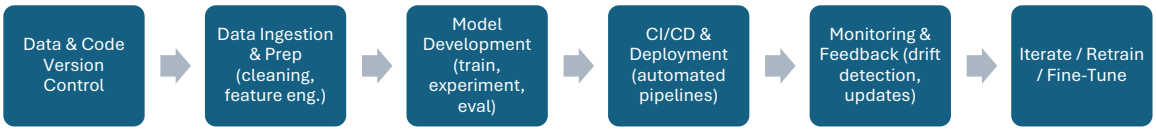
Visualization and Branding (Capability) branding layers, templates, or front-end components for end-user interactions with AI.

3. AI Engineering Best Practices

3.1. MLOps Framework

MLOps (Machine Learning Operations) is the engineering backbone of the AI Factory, ensuring that AI models – whether classical ML or Gen AI – move from development to production reliably, efficiently, and at scale. It combines DevOps principles (automation, continuous integration, continuous delivery) with machine learning best practices (data and model versioning, validation, monitoring).

Below is a conceptual diagram of a typical MLOps lifecycle, illustrating key stages and feedback loops:



3.1.1. Purpose and Role of MLOps

In the AI Factory, MLOps serves as the *assembly line* that translates experimental models into **production-grade AI services**, ensuring:

- **Faster Time to Value:** Automated processes reduce friction, enabling frequent iterations and deployments.
- **Better Risk Management:** Monitoring, version control, and governance mitigate risks such as data drift, model bias, or compliance breaches.
- **Cost Efficiency:** Shared infrastructure, reusable pipelines, and resource orchestration help optimize cloud or on-premise costs.
- **High Reliability:** Standardized validation steps and CI/CD ensure that only models meeting quality thresholds go live.

3.1.2. Lifecycle Stages

Although details differ between classical ML and GenAI projects, most pipelines share the same high-level stages:

3.1.2.1. Data Ingestion & Preparation

- **Core Activities:** Data collection, cleaning, feature engineering (e.g., classical numeric features or text embeddings), and data validation.
- **Common Tools & Frameworks:**
  - Apache Airflow, Azure Data Factory, AWS Glue, dbt – for orchestrating data pipelines.
  - Great Expectations, Deequ – for data quality checks and schema validation.
- **Design Considerations:**
  - Ensure scalability for large volumes (e.g., GenAI corpora).
  - Maintain a robust metadata/catalog system to enable traceability and governance.

3.1.2.2. Model Development & Training

- **Core Activities:** Experimentation, hyperparameter tuning, model evaluation. For GenAI, this might include fine-tuning large language models or training stable diffusion networks.
- **Common Tools & Frameworks:**
  - MLflow, Kubeflow, SageMaker, Vertex AI – for tracking experiments, packaging models, and orchestrating training.
  - PyTorch, TensorFlow, Hugging Face Transformers – for model implementation.
- **Design Considerations:**

- Use version control (Git, DVC) for both data and model artifacts.
- Factor in hardware acceleration (GPUs, TPUs) and distributed training if large-scale or GenAI modeling is required.

3.1.2.3. Continuous Integration & Deployment (CI/CD)

- Core Activities: Automated testing, containerization, versioned releases, and environment consistency.
- Common Tools & Frameworks:
  - Jenkins, GitLab CI, GitHub Actions, Azure DevOps, Argo CD – for DevOps pipelines.
  - Docker, Kubernetes – for containerization and orchestration.
- Design Considerations:
  - Define quality gates (e.g., performance thresholds, bias checks) that must pass before deployment.
  - Support blue-green or canary deployment strategies to safely roll out new model versions.

3.1.2.4. Deployment & Integration

- Core Activities: Exposing the model as a scalable service, or embedding it into applications. For GenAI, may also include specialized inference paths (e.g., prompt engineering).
- Common Tools & Frameworks:
  - TorchServe, TensorFlow Serving, NVIDIA Triton, vLLM – for serving ML models.
  - API Gateways (e.g., Kong, AWS API Gateway) – to manage external access.
- Design Considerations:
  - Balance latency vs. throughput requirements.
  - For GenAI, incorporate RAG or caching stchchatrategies if needed for more efficient inference.

3.1.2.5. Monitoring & Maintenance

- Core Activities: Observing live performance, detecting data drift or concept drift, collecting user feedback, and triggering retraining or fine-tuning.
- Common Tools & Frameworks:
  - Prometheus, Grafana, Evidently AI, Fiddler AI – for metrics and drift detection.
  - PagerDuty, Opsgenie – for alerting on anomalies or critical failures.
- Design Considerations:
  - Implement real-time logging/monitoring dashboards to catch issues promptly.
  - For GenAI, monitor potential hallucinations or harmful content via content moderation pipelines.

3.1.2.6. Governance & Compliance

- Core Activities: Maintaining audit trails, ensuring ethical AI guidelines are followed, verifying data privacy measures, and guaranteeing reproducibility.
- Design Considerations:
  - Incorporate role-based access control (RBAC) to manage who can deploy or retrain models.
  - Maintain an AI Model Registry with notes on training data lineage, intended use, and risk classification.

3.1.3. Common Pitfalls and Lessons Learned

Implementing MLOps is a journey, and many organizations encounter similar pitfalls along the way. Being aware of these common challenges can help teams avoid them or quickly mitigate issues. Below are some frequently reported pitfalls from real-world MLOps projects, along with lessons learned:

- Siloed Teams and Lack of Collaboration: One primary pitfall is when data scientists, engineers, and IT/operations work in isolation without continuous communication. This often leads to mismatches – for example, data scientists build a model that cannot be deployed on the target infrastructure, or operations teams deploy a model without understanding its limitations. Lesson learned: Encourage a culture of collaboration from day one. Hold regular cross-functional meetings and design reviews so that everyone understands the end-to-end pipeline. Many successful teams embed ML engineers with data scientists or create “ML product squads” to ensure tight

feedback loops. This addresses the classic "throwing over the wall" issue and leads to models that are easier to productionize.

- **Neglecting Data Quality and Versioning:** A very common pitfall is underestimating the importance of data management. Models get trained on data that is not representative of production (garbage in, garbage out), or data changes over time but the team doesn't keep track of which version was used for which model. This can lead to unreproducible results and mysterious model failures. **Lesson learned:** Implement data validation and versioning as part of the pipeline (the **Data as Code** principle).
- **Inadequate Model Monitoring and Governance:** Some organizations deploy a model and then fail to monitor its performance or compliance, resulting in issues going unnoticed. For instance, a model's accuracy might slowly degrade, or it might start exhibiting bias after some time in production, but without monitoring, the problem is only caught after causing damage (e.g., customer complaints or regulatory attention). **Lesson learned:** Treat monitoring and governance as first-class citizens. Establish a monitoring system that tracks metrics and drift and set up alerts for anomalies. Also, put governance policies in place: e.g., require periodic model audits, create dashboards for key metrics like bias and accuracy, and schedule regular reviews of all production models.
- **Insufficient Automation (Manual Processes):** MLOps aims to automate the ML lifecycle, yet a pitfall is leaving too many steps manual (such as manually copying data, retraining ad-hoc, or manually deploying models). This not only slows down iteration but introduces human error. **Lesson learned:** Invest time in automating from the beginning. Even if scripts and automation take initial effort, they pay off in consistency. One lesson is to use Infrastructure-as-Code and Pipeline-as-Code early – teams that rely on manual notebook runs often struggle to reproduce results or scale the process. By automating, say, a nightly retraining job or an automated test suite for models, teams have more confidence to experiment quickly, knowing the safety nets are there.
- **Overfitting to Offline Metrics:** Teams sometimes focus on optimizing a model for offline validation metrics, only to find it doesn't translate to business value or real-world performance (**"model myopia"**). For instance, a model could achieve high accuracy on a validation set but still make unacceptable errors in production (perhaps because the validation set wasn't truly representative of future data, or because the real cost of certain errors wasn't captured by the metric). **Lesson learned:** Include business context in evaluation – use relevant metrics and test scenarios. One best practice is to perform a pilot or A/B test with the model in a shadow mode to gather real-world performance data before fully launching it. Additionally, engage stakeholders to define what success looks like beyond just technical metrics (e.g., user satisfaction, revenue impact). This pitfall underscores the importance of cross-functional input in the evaluation stage and possibly maintaining **human-in-the-loop** checks for critical decisions initially.
- **Scaling and Performance Issues in Production:** Another lesson comes when models that performed well in the lab falter at scale – maybe the inference latency is too high under production load, or memory usage is excessive. This is often due to neglecting **"non-functional" requirements** during development. **Lesson learned:** Include engineering considerations (like latency, memory, CPU/GPU utilization) as part of model validation. For example, a team deploying an image recognition model learned that the model was too slow on their edge devices; they had to scramble to optimize it. Since then, they incorporated a step in the pipeline to measure inference time and resource usage for each candidate model and set thresholds to decide if a model is production-worthy. It's also wise to design for scalability – using asynchronous processing, batching predictions, or smaller model alternatives if needed.
- **Lack of Clear Ownership and Process:** Sometimes the biggest pitfall is organizational – if it's unclear who is responsible for the model after it's deployed, things can fall through the cracks (e.g., nobody retrains it when needed, or issues linger without action). Also, without a defined MLOps process, ad-hoc approaches can lead to confusion (multiple people deploying different versions, etc.). **Lesson learned:** Define roles (as earlier section detailed) and a clear workflow. Many organizations create an **"ML operations" playbook or handbook** that outlines how a model moves from development to production, who signs off at each stage, and what checks are mandatory. Having this in place ensures everyone knows their responsibility.

Each of these pitfalls carries a lesson: MLOps is not just about software tooling, but about **people, process, and technology working together**. Adopting MLOps incrementally, learning from failures, and iterating on the pipeline is normal. In fact, many organizations do a post-mortem after major issues (a practice borrowed from Site Reliability Engineering's blameless post-mortems) to improve their MLOps practices. Over time, these lessons translate into robust conventions and automations in the MLOps framework – for example, after suffering from siloed teams, a

company might mandate that a representative from each role is involved in every ML project from the start, or after experiencing data drift issues, a team might integrate a drift detection library and alerts into the standard pipeline.

By being mindful of these common challenges and proactively addressing them, teams can significantly increase the success rate of moving ML projects from ideation to real-world impact. MLOps, done right, becomes a virtuous cycle: each model deployment teaches how to strengthen the pipeline and collaboration for the next one, leading to faster and safer innovation in AI products.

3.2. Generative AI Best Practices

3.2.1. Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) is a technique that helps large language models create better, more current text by pulling in real-time data. Instead of relying on what they learned during training, these models can look up fresh information from external sources like databases or documents as they generate responses. This approach makes the answers more accurate and timelier and helps prevent "hallucinations," where a model might otherwise produce inaccurate or made-up details.

The RAG framework sees the process in two primary phases:

- **Retrieval:** The model first does a retrieval phase, searching outside sources to compile relevant details directly related to the user's question.
- **Generation:** The technique works and weaves this recently acquired information together in the generating phase to create a cogent, contextually rich, and informed response.

This dynamic approach makes RAG highly effective for use cases requiring real-time information retrieval, such as open-domain question answering, customer support automation, or enterprise search. Additionally, RAG offers advantages over techniques like fine-tuning, where models are retrained on new data, or prompt engineering, which focuses on optimizing the input query for better results.

RAG can take many forms of architecture. The next few next sections describe a few of the most popular RAG architectures.

3.2.1.1. Simple RAG

The RAG's simplest form: Given a query, the language model searches and pulls relevant documents from a designated database and subsequently creates a response utilizing the acquired material. Working with a smaller, more stable dataset that doesn't call for regular updates or sophisticated processing makes this arrangement perfect.

Workflow:

- **User Query:** The user submits a question or prompt.
- **Information Retrieval:** The model searches a fixed database for relevant documents.
- **Response Generation:** The model constructs a well-informed and fact-based response using the retrieved information.

Use Case: Simple RAG is perfect for FAQ systems or customer service bots where responses must be factually accurate. However, the scope of information is limited to a known collection of documents, like a product manual or knowledge base.

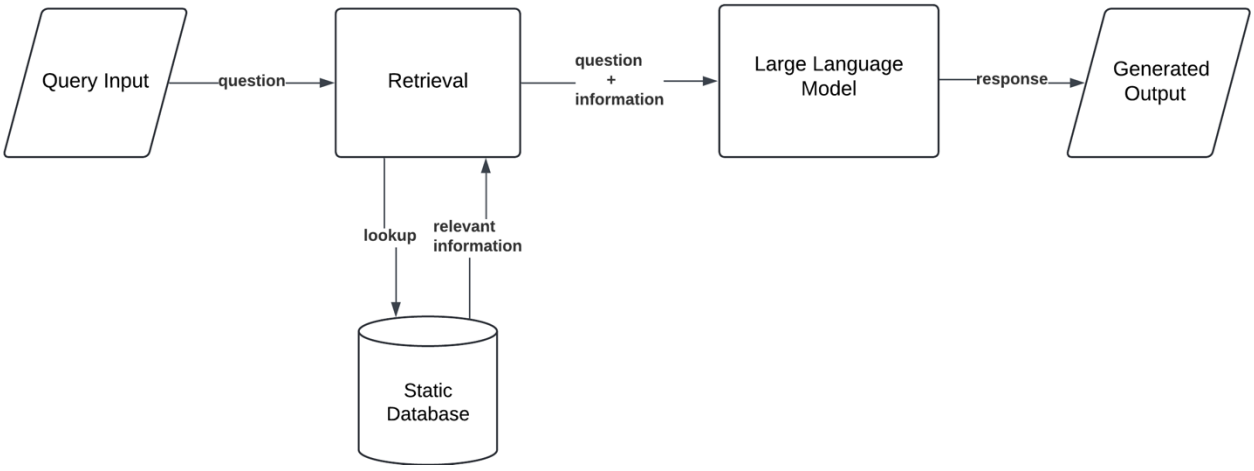


Figure 6. Simple RAG.

3.2.1.2. Simple RAG with Memory

Simple RAG with Memory presents a storage component enabling the model to retain information from past interactions. This addition increases its potency for activities requiring contextual awareness across several searches or ongoing dialogues. Simple RAG can help you do this with prompt caching.

Workflow:

- Query Input: The user submits a query or prompt.
- Generation: The model generates a response by combining retrieved documents with the stored memory.
- Memory Access: The model retrieves past interactions or data stored in its memory.
- Document Retrieval: It searches the external database for new relevant information.

Use Case: This design is beneficial for chatbots in customer service, where continuous encounters require the model to recall user preferences or past issues. In customised recommendations, when previous data increases the relevancy of responses, it also helps.

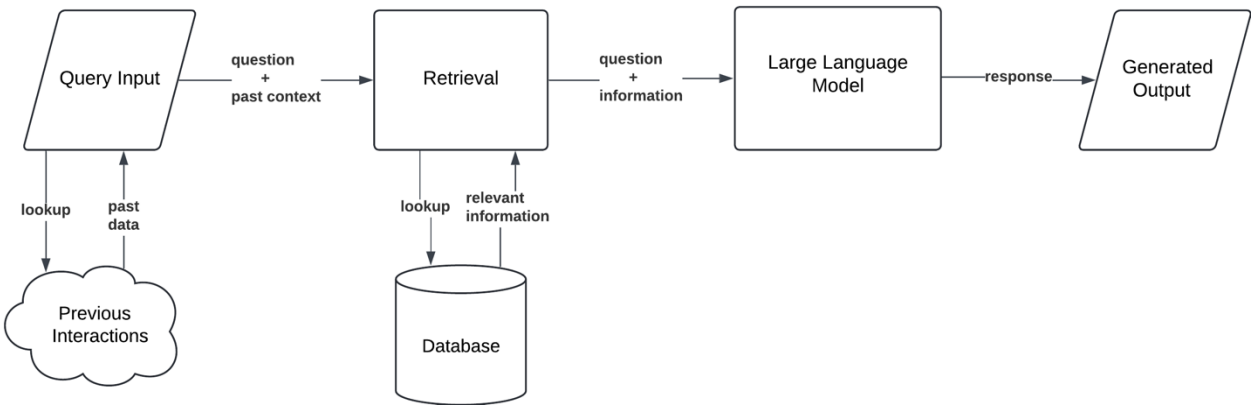


Figure 7. Simple RAG with Memory.

3.2.1.3. Branched RAG

Branched RAG determines which particular data sources should be searched depending on the input, allowing a more flexible and effective data retrieval method. Branched RAG examines the query and chooses the most pertinent source(s) to access information instead of asking for all the accessible sources.

Workflow:

- Query Input: The user submits a prompt.
- Branch Selection: The model evaluates multiple retrieval sources and selects the most relevant one based on the query.
- Single Retrieval: The model retrieves documents from the selected source.
- Generation: The model generates a response based on the retrieved information from the chosen source.

Use Case: Branching RAG is perfect for complicated searches needing specialized knowledge, including legal tools or multidisciplinary research, where the model must select the best information source without aggregating pointless data from many sources.

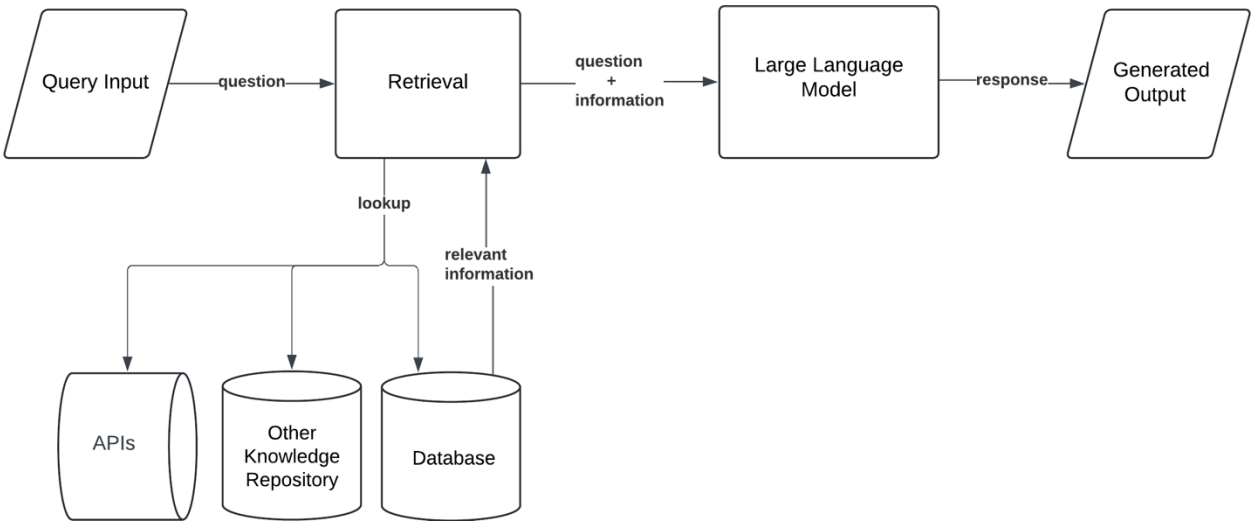


Figure 8. Branched RAG.

3.2.1.4. Hypothetical Document Embedding (HyDE)

One unusual RAG version, Hypothetical Document Embedding (HyDE), creates hypothetical documents depending on the query before collecting relevant information. HyDE initially generates an embedded representation of what an ideal document might look like given the query rather than just retrieving records from a database. It then guides retrieval using this fictitious paper, enhancing the findings' relevance and quality.

Workflow:

- Query Input: The user provides a prompt or question.
- Hypothetical Document Creation: The model generates an embedded representation of an ideal response.
- Document Retrieval: The model retrieves actual documents from a knowledge base using the hypothetical document.
- Generation: The model generates an output based on the retrieved documents, influenced by the hypothetical document.

Use Case: Research and development benefit notably from HyDE since questions may be imprecise, and obtaining data based on ideal or hypothetical solutions helps hone difficult responses. HyDE also relates to the development of creative materials when more flexible, innovative results are required.

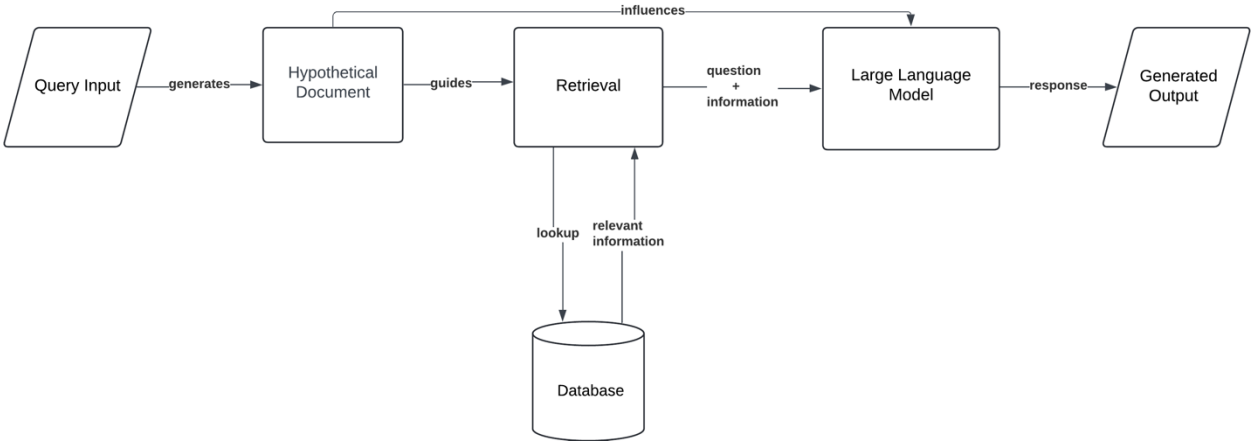


Figure 9. Hypothetical Document Embedding (HyDE).

3.2.1.5. Adaptive RAG

Adaptive RAG is a dynamic implementation whose retrieval approach changes depending on the kind or degree of the query. Unlike fixed models, which follow a single retrieval path independent of the question, adaptive RAG can change its strategy in real-time. For more complicated searches, it may access several data sources or use more advanced retrieval algorithms, while for basic queries, it may obtain documents from a single source.

Workflow:

- **Query Input:** The user submits a prompt.
- **Adaptive Retrieval:** The complexity of the query determines whether the model should change the retrieval technique or obtain documents from one or several sources.
- **Generation:** The model optimizes the retrieval process for every particular query by processing the obtained data and producing a customized answer.

Use Case: Enterprise search systems benefit from adaptive RAG since the type of searches could differ significantly. It guarantees effective handling of simple and sophisticated searches, optimizing the speed and depth balance.

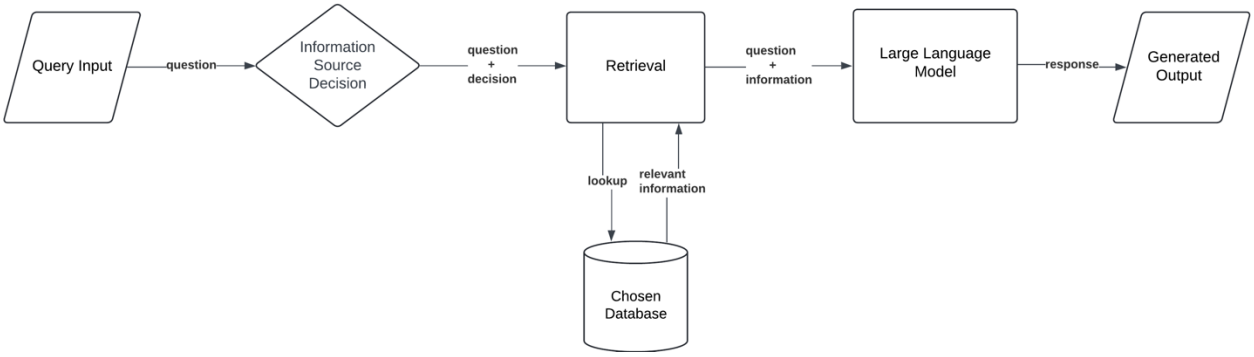


Figure 10. Adaptive RAG.

3.2.1.6. Corrective RAG (CRAG)

Corrective RAG (CRAG) uses a self-reflection or self-grading mechanism on retrieved documents to raise the relevance and accuracy of produced answers. Before entering the generating stage, CRAG critically assesses the quality of the acquired data, unlike conventional RAG models. The algorithm assesses each document for relevance and divides obtained documents into "knowledge strips." Imagine the first retrieval falls short of a threshold of relevance. In such a scenario, CRAG starts extra retrieval actions, including web searches, to guarantee it produces the report using the best available information.

Workflow:

- **Query Input:** The user submits a query or prompt.
- **Document Retrieval:** The model retrieves documents from the knowledge base and evaluates their relevance.
- **Knowledge Stripping and Grading:** The retrieved documents are broken down into "knowledge strips" — smaller information sections. Each strip is graded based on relevance.
- **Knowledge Refinement:** Irrelevant strips are filtered out. If no strip meets the relevance threshold, the model seeks additional information, often using web searches to supplement retrieval.
- **Generation:** Once a satisfactory set of knowledge strips is obtained, the model generates a final response based on the most relevant and accurate information.

Use Case: Corrective RAG is perfect for applications that require great factual accuracy, such as legal document preparation, medical diagnosis support, or financial analysis, where minor errors can have significant repercussions. **Evaluating models** such as CRAG requires considering the consistency of error correction and retrieval accuracy to guarantee the best performance in sensitive areas.

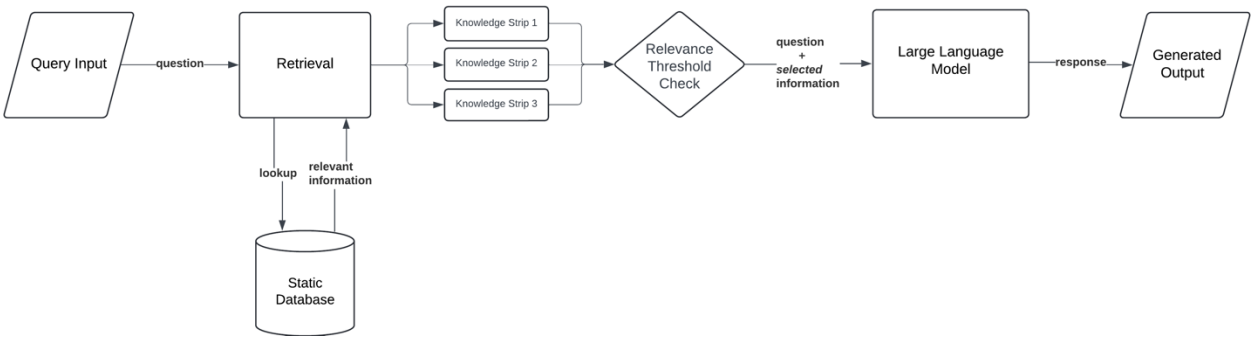


Figure 11. Corrective RAG (CRAG).

3.2.1.7. Self-RAG

Self-RAG presents a self-retrieval technique that lets the model create retrieval queries on its own during generation. Unlike conventional RAG models, which are based just on the user's input, Self-RAG may iteratively optimize its retrieval searches as it develops material. Especially for complicated or changing searches, this self-guided technique improves the relevancy and quality of the information.

Workflow:

- Query Input: The user submits a prompt.
- Initial Retrieval: The model retrieves documents based on the user's query.
- Self-Retrieval Loop: The model finds information gaps during generating and generates fresh retrieval searches to access more data.
- Generation: The model generates a final response, iteratively improving it by retrieving further documents as needed.

Use Case: In exploratory research or long-form content creation, when the model must dynamically generate new information as the answer changes to guarantee thorough and accurate findings, Self-RAG is quite successful.

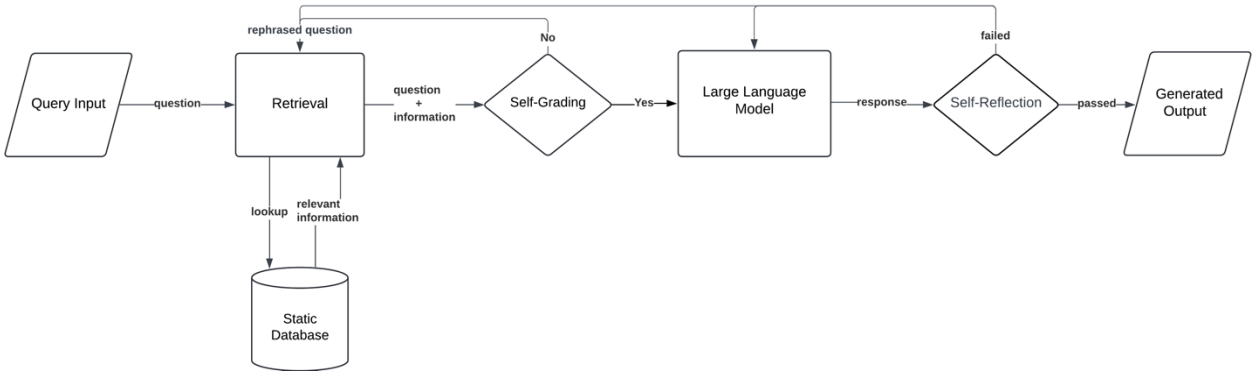


Figure 12. Self-RAG.

3.2.1.8. Agentic RAG

Agentic RAG presents a more independent, agent-like behavior in the retrieval and creation process. Under this implementation, the model functions as an "agent" capable of completing challenging, multi-stage tasks using proactive interaction with several data sources or APIs to compile knowledge. Agentic RAG distinguishes itself by letting Document Agents assign themselves to every document and coordinate their interactions via a meta-agent. This system makes more complex decision-making possible, helping the model decide which retrieval techniques or outside systems to interact with, depending on the query complexity.

Workflow:

- **Query Input:** The user submits a complex query or task.
- **Agent Activation:** The model activates multiple agents. Each Document Agent is responsible for a specific document and can answer questions and summarize information from that document.
- **Multi-step Retrieval:** The Meta-Agent manages and coordinates the interactions between the various Document Agents, ensuring the most relevant information is retrieved.
- **Synthesis and Generation:** Combining the outputs of the separate Document Agents, the meta-agent creates a complete, cogent response based on the cumulative knowledge acquired.

Use Case: Agentic RAG is perfect for tasks like automated research, multi-source data aggregation, or executive decision support, where the model needs to pull together and synthesize information from various systems autonomously.

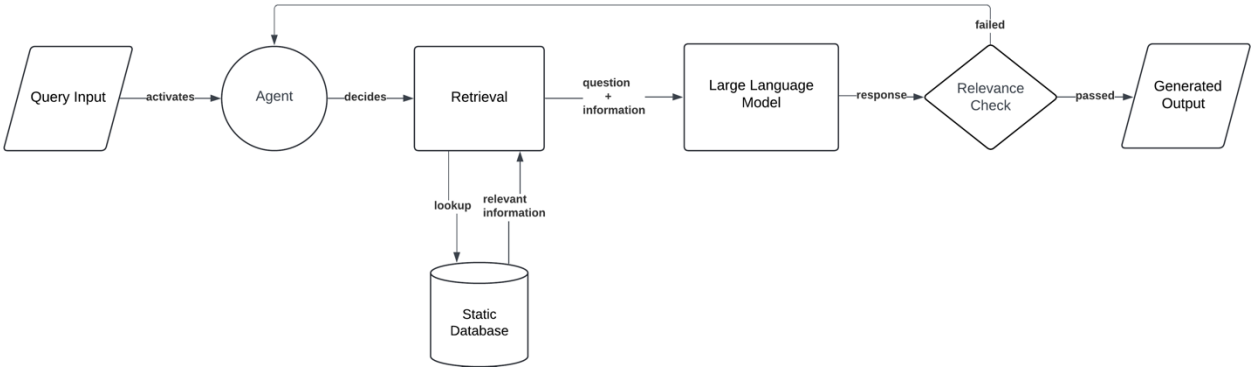


Figure 13. Agentic RAG.

3.2.2. Optimization Techniques

3.2.2.1. Prompt Caching

As LLMs grow more powerful, they also become more resource intensive. Bigger models require more computing power, leading to higher costs and slower response times. This has been a long-standing challenge for AI applications.

Prompt caching offers an innovative solution that stores and reuses responses to repeated prompts. Instead of processing the same request repeatedly, AI systems can pull from a cache of past responses, dramatically cutting down on processing time and cost. This technique is beneficial for handling long prompts that contain repetitive elements, like system instructions or frequently used queries.

Why Prompt Caching Matters

Major AI providers like OpenAI and Anthropic rely on prompt caching to optimize their services, with reported benefits such as:

- Up to 90% cost reduction.
- Up to 80% lower latency.

Prompt caching minimizes redundant processing, making it feasible to provide LLMs with additional background knowledge and structured responses while maintaining efficiency.

How Prompt Caching Works

OpenAI's Approach

OpenAI automatically enables prompt caching for prompts 1024 tokens or longer, ensuring faster response times for frequently used queries. Here's how it works when you send an API request:

1. **Cache Lookup:** The system checks if your prompt's beginning (prefix) is already stored.
2. **Cache Hit:** If found, the cached result is used, significantly reducing processing time and cost.
3. **Cache Miss:** If no match is found, OpenAI processes the full prompt and stores its prefix for future use.

How long do prompts stay cached?

- Typically, 5–10 minutes of inactivity.
- During off-peak times, caches may last up to one hour.

Best Practices for Prompt Caching according to OpenAI:

- Keep static content at the start and place dynamic or retrieved information (such as in RAG) at the end.
- Monitor cache hit rates, latency, and token usage to fine-tune your caching strategy.
- Leverage off-peak hours to increase cache hits since evictions (cache deletions) happen more frequently during busy periods.
- Avoid unnecessary changes to the prefix to prevent frequent cache evictions.

Anthropic's Approach

Anthropic follows a similar caching process but offers more control. Instead of automatically caching prompts, Anthropic allows users to define up to 4 cache breakpoints. This means you can decide which sections of your prompt should be cached using the cache control parameter.

How it works:

1. The system checks if the prefix of a previously used prompt is stored.
2. The cached response is used if found, reducing processing time.
3. If not found, the full prompt is processed, creating a new cache entry.

Anthropic refreshes cached content every time it's accessed and stores it for 5 minutes before automatic removal.

Prompt Caching in Other Systems (CacheGPT, GPTCache, etc.)

Other frameworks, such as GPTCache, take a slightly different approach:

- User input is transformed into a vector embedding (a mathematical text representation).
- A similarity search is performed on the cache.
- If a match (cache hit) is found, the cached response is used.
- If no match (cache miss) occurs, the LLM generates a new response and stores it for future use.

This method ensures that similar requests are handled efficiently, reducing unnecessary processing.

Prompt caching is a powerful optimization tool for LLM applications, making AI models faster and more cost-effective. Whether using OpenAI, Anthropic, or other caching frameworks, understanding how to structure prompts and leverage caching effectively can lead to substantial performance gains.

	Uncached Input Tokens	Cached Input Tokens	Output Tokens
GPT-4o			
gpt-4o-2024-08-06	$2.50	$1.25	$10.00
GPT-4o fine-tuning	$3.75	$1.875	$15.00
GPT-4o mini			
gpt-4o-mini-2024-07-18	$0.15	$0.075	$0.60
GPT-4o mini fine-tuning	$0.30	$0.15	$1.20
o1			
o1-preview	$15.00	$7.50	$60.00
o1 mini	$3.00	$1.50	$12.00

Figure 14. Input token savings with Prompt Caching on OpenAI. Source: OpenAI.

Model	Base Input Tokens	Cache Writes	Cache Hits	Output Tokens
Claude 3.5 Sonnet	$3 / MTok	$3.75 / MTok	$0.30 / MTok	$15 / MTok
Claude 3 Haiku	$0.25 / MTok	$0.30 / MTok	$0.03 / MTok	$1.25 / MTok
Claude 3 Opus	$15 / MTok	$18.75 / MTok	$1.50 / MTok	$75 / MTok

Figure 15. Prompt caching pricing with Claude. Source: Anthropic.

3.2.2.2. Late Chunking: A Cost-Effective Solution for Long-Context Retrieval in RAG

RAG systems handling long-context retrieval often face a tradeoff between precision and computational efficiency. Late chunking, introduced by JinaAI, offers a middle ground between naive chunking (low cost but reduced precision) and late interaction/ColBERT (high precision but expensive).

What is ColBERT and Why Does it Matter?

Before diving into late chunking, it's important to understand the alternative: ColBERT (Contextualized Late Interaction over BERT). Traditional RAG systems often use naive chunking, but for highly precise retrieval, ColBERT offers a more granular approach. ColBERT uses token-level embeddings to compare query tokens with document tokens without pre-pooling embeddings into a single vector. This allows for a very fine-grained comparison, capturing subtle semantic relationships. However, this comes at a significant cost: it requires storing embeddings for every token in a document, leading to massive storage overhead and expensive query-time computation. Late chunking offers a way to achieve much of the precision benefit of ColBERT without the prohibitive costs.

RAG systems handling long-context retrieval often face a tradeoff between precision and computational efficiency. The Late chunking concept offers a middle ground between naive chunking (low cost but reduced precision) and late interaction/ColBERT (high precision but expensive).

What is Late Chunking?

Traditional vs. Late Chunking Approaches

Approach	Process	Key Drawback
Naive Chunking	Chunk first, then embed each chunk separately	Loses cross-chunk context
Late Interaction (ColBERT)	Embed each token, then compare query tokens with document tokens	Massive storage and compute costs
Late Chunking	Embed the full document first, then chunk embeddings	None significant

Late chunking reverses the traditional embedding and chunking order. Instead of chunking a document into smaller pieces and then embedding each chunk, late chunking first embeds the entire document into a single vector representation. Then, this document embedding is divided into smaller, contextually enriched segments. This ensures that each chunk retains contextual awareness from the entire document. It enables long-context retrieval while keeping storage and computational costs manageable.

Why is Late Chunking Beneficial for RAG?

A. Solves Context Loss from Naive Chunking

- Naive chunking breaks documents into independent sections, leading to loss of references between them.
- Late chunking preserves cross-chunk relationships, ensuring queries retrieve coherent and complete answers.

B. Reduces Storage and Compute Costs vs. Late Interaction (ColBERT)

While ColBERT retains per-token embeddings for query-time comparison, it requires massive storage overhead.

Approach	Total Vectors Stored (for 100,000 documents)	Storage Required
Naive Chunking	800 million	~2.46 TB
Late Interaction (ColBERT)	~1.6 million	~4.9 GB
Late Chunking	~1.6 million	~4.9 GB

Late chunking achieves similar storage savings as naive chunking but maintains contextual integrity.

When to Use Late Chunking in RAG?

Scenario	Best Approach
Short documents (≤1,000 tokens)	Naive Chunking
Long documents with structured data	Late Chunking
High precision retrieval, unlimited storage	Late Interaction (ColBERT)
Enterprise RAG needing cost-efficiency	Late Chunking

3.2.3. Workflows

GenAI workflows are orchestrated sequences of prompts, model calls, and tool uses that break complex tasks into manageable steps. Instead of relying on a single monolithic prompt, a workflow chains together multiple LLM interactions (and possibly other operations) with explicit logic controlling the flow. This structured approach brings predictability and easier debugging, as each step is well-defined. Common patterns for GenAI workflows include the **augmented LLM** as a building block and various chaining or routing techniques to handle complexity.

3.2.3.1. Augmented LLM

An "augmented LLM" is a large language model enhanced with retrieval (for external data), tooling (for taking actions), and memory (for referencing past events).

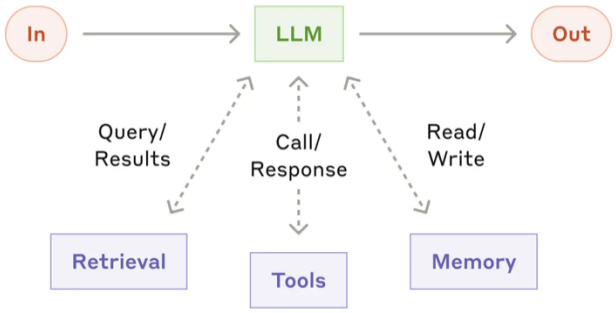


Figure 16. Augmented LLM.

This model is then wrapped in an interface that allows an agent to decide:

- Which tool to call (e.g., external API, database, or local code execution).
- How and what to store as "memory" or retrieve for later steps.
- Whether to revise, retry, or generate a final output.

This augmented LLM concept underpins many advanced workflows and agent systems (see Agentic AI below). Several frameworks (Amazon Bedrock's AI Agents, LangChain's LangGraph, GUI builders like Rivet and Vellum) provide abstractions to implement augmented LLMs easily. However, using simple API calls with custom logic is often preferred for transparency and control. Several frameworks—such as Amazon Bedrock's AI Agent Framework, LangGraph, Vellum, and Rivet—offer abstractions for these augmentations. Often, however, straightforward API calls with a bit of custom logic provide greater transparency and ease of debugging. For illustrative agentic patterns, refer to resources like the Anthropic Cookbook.

3.2.3.2. Prompt Chaining

Breaks the task into sequential prompts, with each step receiving output from the previous step. Allows insertion of checks or "gates" in between.

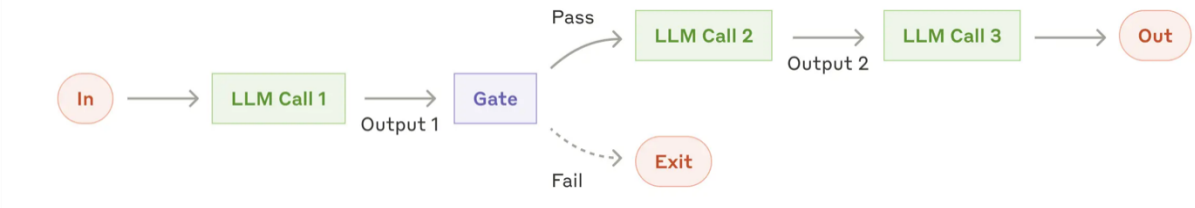


Figure 17. Prompt Chaining.

Typical Use Cases: Content generation in multiple steps (outline → draft → refine); multi-step Q&A where each step filters or transforms information.

3.2.3.3. Routing

Classifies or routes requests to different LLM prompts or tools based on the input category. This avoids “one-size-fits-all” prompts and allows specialized handling of certain requests.

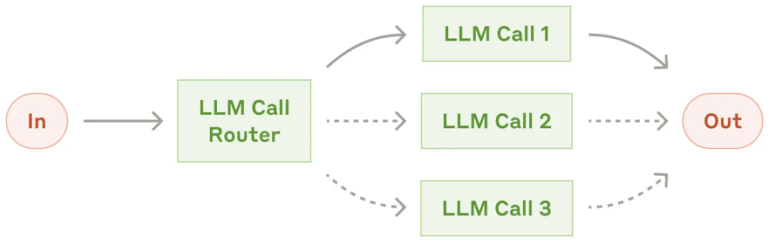


Figure 18. Routing.

Typical Use Cases: Customer support queries directed to specialized flows; routing simple queries to smaller models and complex ones to more capable LLMs.

3.2.3.4. Parallelization

Splits the work so multiple LLM calls run at once, then combines outputs. This can include repeating the same task multiple times for “voting” or dividing a complex task into smaller pieces (“sectioning”).

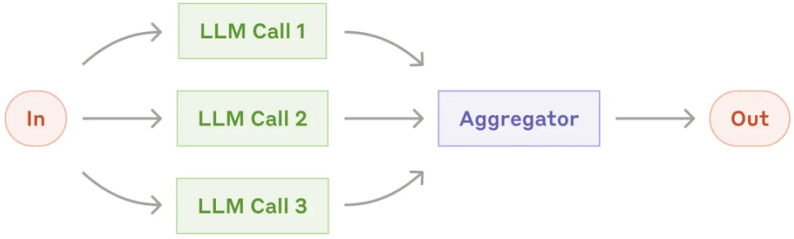


Figure 19. Parallelization.

Typical Use Cases: Faster evaluations by distributing subtasks; using multiple LLM calls to cross-check answers for reliability; code reviews with separate checks.

3.2.3.5. Orchestrator

An “orchestrator” LLM delegates subtasks to specialized “worker” LLMs, then merges the results. Helpful when you cannot predefine how many subtasks will be needed.

Typical Use Cases: Large-scale code edits (where file count and changes are dynamic); multi-document retrieval and summarization tasks.

3.2.3.6. Evaluator–Optimizer

Uses a loop of “draft” LLM calls followed by “review” LLM calls that provide feedback or critique. The drafting model then iterates until it produces a final solution.

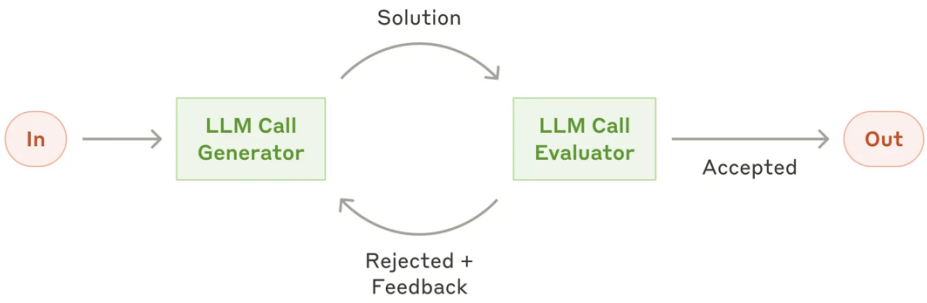


Figure 20. Evaluator-Optimizer.

Typical Use Cases: Complex outputs that benefit from iterative refinement, such as literary translation, systematic research, or multi-step problem solving.

3.2.3.7. Memory

Workflows with a memory component explicitly maintain state across multiple turns or sessions beyond the LLM's immediate context window. This can be as simple as carrying a "scratchpad" of the conversation so far, or as advanced as using an external vector database to store and retrieve long-term knowledge.

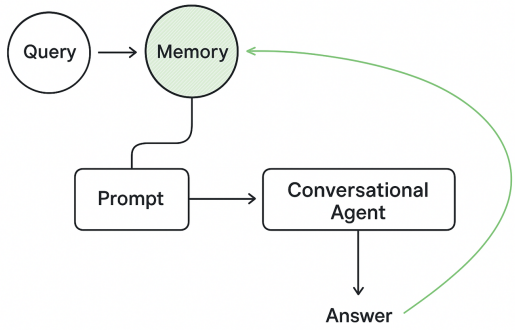


Figure 21. Memory.

Typical use cases: long-running dialogues or processes where earlier information must be remembered after many turns (exceeding what the prompt can hold); personal assistants that learn user preferences over time; any scenario where the AI needs to refer back to something said or learned in previous interactions. Memory can be implemented by summarizing old context, retrieving relevant facts from a knowledge base, or logging interactions to feed back in when needed. The challenge is ensuring the memory is kept up-to-date and doesn't introduce outdated or irrelevant information.

3.2.3.8. Human-in-the-Loop

Incorporating human judgment at key points of a workflow can greatly enhance quality and safety. In a human-in-the-loop design, the workflow will pause for a person to review, correct, or approve the AI's output before proceeding to the next step or finalizing the result.

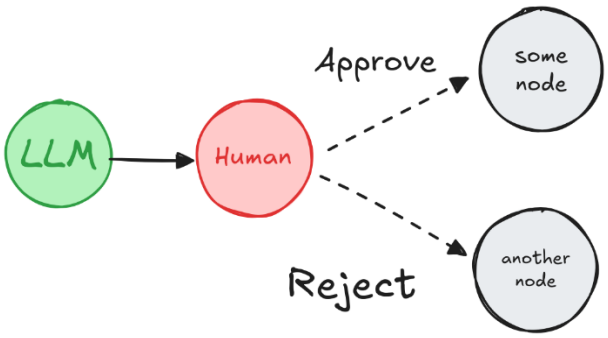


Figure 22. Human-in-the-loop.

Typical use cases: content generation processes where a human editor reviews each stage (for example, an AI writes a draft, a human edits or approves it, then the AI continues refining); decision-support systems in high-stakes fields (medical, legal, financial) where a human expert must validate the AI's intermediate conclusions; moderation pipelines where AI flags content and humans make the final decision. Human-in-the-loop workflows ensure oversight and allow for corrective feedback, combining AI speed with human judgment.

GenAI workflows like the above provide structured orchestration of LLM capabilities. They shine when the solution path can be anticipated or constrained by design. By carefully designing chains, routes, or loops, practitioners can leverage LLMs for complex tasks while mitigating errors step by step. These workflow patterns are also building blocks

for more agentic systems, where an AI dynamically chooses its own sequence of actions. Next, we explore *agentic AI*, which grants the model more autonomy in planning and tool use.

3.2.4. Tools

3.2.4.1. OpenAPI Tools and Plugins

One proven method to enable tool use is leveraging the OpenAPI specification standard to expose external APIs to the LLM in a controlled way. OpenAPI (formerly Swagger) is a format for describing RESTful APIs. In an LLM context, an API's OpenAPI spec can serve as a *guidebook* for the model, detailing what endpoints exist and how to call them. For example, OpenAI's ChatGPT plugin system uses an OpenAPI spec (plus a manifest) to let the model safely interact with third-party services. The language model "reads" the API description and, if the user's request is relevant, the model can choose an appropriate endpoint and ask the tool to execute it. The LLM only knows about the API what is defined in the spec, which allows developers to expose just the parts of a service they want the model to access.

By 2024, a wide range of plugins had been created using this approach – from travel planning and shopping to data analysis – showcasing how *OpenAPI-described tools* can greatly extend an AI's capabilities. Developers integrating LLMs into applications can adopt the same pattern for internal APIs: document the endpoints and use the model's function calling features to let it invoke those operations. Best practices include keeping the API spec concise (to fit into prompt context), providing clear examples of usage in the description, and putting guardrails on execution (like timeouts or quotas for tool calls). This approach allows complex operations (database lookups, calculations, transactions) to be performed by classic software components, triggered by natural language intents from the LLM. It bridges the gap between free-form text and programmatic actions. Many LLM development frameworks now support plugin-style integrations; for instance, LangChain offers an OpenAPI tool wrapper to automatically parse a spec and enable an agent to call those endpoints.

3.2.4.2. Model Context Protocol (MCP)

The Model Context Protocol (MCP) is a newer open standard introduced by Anthropic in late 2024 and designed to standardize interactions between LLM applications and external data sources or tools.

MCP establishes a *client–server architecture* for AI context: AI applications act as clients that can discover and retrieve data from any number of MCP servers which expose data or services. In practical terms, an organization can stand up **MCP servers** for its resources (documents, databases, SaaS tools), and any AI assistant or agent that implements an **MCP client** can query those resources in a standardized way. This removes the need for custom integration code every time you want to give an AI access to a new system – instead of writing a bespoke connector, you adhere to the MCP standard.

This structure allows LLMs to access and utilize external resources efficiently. Integrating MCP into existing architectures offers several benefits:

- **Standardized Integration:** A universal protocol means AI developers don't have to reinvent connectors for each data source. An AI client that speaks MCP can interface with many services interchangeably, much like web browsers work with any website via HTTP. This simplifies development and accelerates connecting AI to real-world applications.
- **Enhanced Interoperability:** By adhering to a common standard, diverse AI and tool providers can work together in one ecosystem. For example, a financial database and a document repository, each with an MCP server, can both be accessed by the same AI agent seamlessly. This avoids siloed AI integrations and promotes a more cohesive environment where tools can be swapped or added without new coding.
- **Security and Control:** MCP was designed with enterprise use in mind – it includes mechanisms for secure, permissioned access to data and tools. The protocol allows fine-grained control over what an AI system can see or do, and since interactions happen through a defined interface, it's easier to monitor and govern those actions. For instance, an MCP server can enforce authentication and logging on all requests, ensuring compliance with security policies.

To adopt MCP, developers typically play one of two roles (or both):

- Server developers: Expose your existing API or data source via an MCP server. This involves implementing the MCP spec for your system – essentially wrapping your database or service with an MCP-compatible interface.

Fortunately, many reference implementations are emerging. Anthropic has open-sourced MCP server templates for popular platforms like Google Drive, Slack, GitHub, and Postgres, which can be adapted to speed up development. By running an MCP server, you make your data/tool accessible to any AI client that speaks the protocol.

- Client developers**:** Integrate MCP client functionality into your AI application or agent. This means your AI system can discover available MCP servers (through a registry or configuration), request context or actions from them, and incorporate the results into its reasoning. For example, if an AI assistant needs customer info from a CRM, it would connect to the CRM's MCP server, fetch the data, and include it in the prompt. Anthropic's Claude and other LLMs are adding support to interact with MCP servers natively, and SDKs are available to help clients manage connections. As a client developer, you ensure your AI knows when and how to query MCP resources during its workflow.

3.2.5. Agentic AI

Agentic AI focuses on giving LLMs and their surrounding systems a level of *autonomy* in how they plan, reason, and act to accomplish open-ended tasks. In an agentic setup, the AI is not just following a fixed script or workflow – instead, it's dynamically deciding which actions to take (which could include calling tools, asking for more information, or creating new subtasks) in order to achieve a goal. Many of the building blocks from GenAI workflows still apply, but agentic systems let the model **orchestrate itself**. This flexibility enables solutions to complex problems that developers can't fully predefine, but it also introduces new challenges in control and predictability.

3.2.5.1. What are AI Agents?

The term *AI agents* can refer to a range of systems with varying levels of autonomy and complexity. For clarity, these can be broadly referred to as **agentic systems**, with a key architectural distinction drawn between **workflows** and **agents**:

- **Workflows** are orchestrated systems that leverage LLMs and tools through predefined code paths. These are typically directive in nature, following structured logic to achieve outcomes.
- **Agents**, in contrast, are systems that utilize LLMs dynamically. They control their own processes, make decisions, and use tools directly to accomplish tasks. Agents operate with a higher degree of autonomy, often adapting their actions based on intermediate results, enabling them to handle complex or open-ended objectives.

In agentic systems, the AI has the ability to **reason, plan, and act** in a goal-directed manner, rather than just respond passively.

You can think of Agentic AI like this:
Agentic AI = Autonomy + Long-term planning + Memory + Tool use.

The LLM isn't just generating an answer – it's interacting with its environment (via tools or APIs), possibly remembering past steps, and adjusting its plan as needed to reach an objective.

3.2.5.2. When (and When Not) to Use Agents

In practice, start with the simplest reliable approach before adding any complexity. Many applications can be served effectively by a single LLM call or by a predefined sequence of prompts and tools. Fully agentic systems make sense when:

- The task requires flexible, model-driven decision-making at scale (e.g., you cannot predefine how many steps or tools will be needed).
- You trust the model's autonomous reasoning enough to let it operate for multiple turns.

However, agentic approaches incur higher latency, expense, and debugging overhead. They may also amplify errors over multiple steps. For tasks that follow predictable routines, a structured workflow (orchestrated prompts and tools) is generally sufficient. Augmentation with retrieval and carefully chosen in-context examples often achieve most goals with less cost and complexity.

3.2.5.3. Agentic AI Architecure Patterns

LLM Reasoning & Tool-Use Loop (Chain-of-Thought + ReAct)

Single-agent reasoning loop that uses an LLM to think step-by-step and call external tools as needed. This pattern underpins many "smart" AI assistants. The agent incrementally breaks down the problem, decides on actions (like invoking an API or database), observes the results, and continues this cycle until it produces a final answer. By interleaving reasoning and acting (the core idea of the ReAct framework), an agent can tackle more complex queries than a single prompt response, with improved accuracy and flexibility. However, each extra "thought" or tool use is an extra LLM invocation – introducing latency and cost, and requiring careful orchestration for reliability.

Architecture

At heart, this pattern is a while-loop around an LLM. On each iteration, the LLM receives the current state (user query plus any intermediate results or "scratchpad" memory) and outputs either a reasoning step or a tool invocation. Modern implementations often use the model's ability to output a function call (tool name + parameters) when appropriate, otherwise outputting text. The system executes the tool if requested (e.g. call a search API) and feeds the result (observation) back into the LLM on the next step. This continues until the LLM indicates it has a final answer. The popular ReAct paradigm formalized this loop of Reasoning ("Thought") → Action → Observation, but many variations exist. For example, an agent may simply chain a fixed sequence of tools or prompts (aka Chain-of-Thought prompting) without dynamic decision-making – useful for static workflows. In practice, ReAct-style agents integrate three core concepts: tool use, memory, and planning within the LLM reasoning loop.

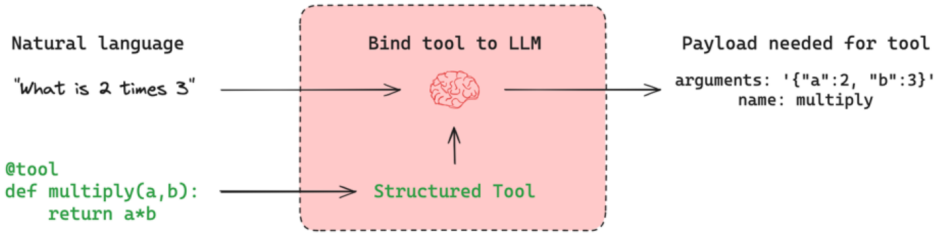


Figure 23. Tool-usage within an LLM agent.

The LLM (agent) is "bound" to external tools via defined interfaces. It can decide to invoke a tool (e.g. a calculator function) rby outputting a structured call, which the runtime executes, then returns the result back into the agent's context. This allows the agent to augment its chain-of-thought with accurate computations, API data, etc., rather than relying on its own limited knowledge.

Trade-offs

The iterative reasoning loop dramatically boosts problem-solving success – e.g. one experiment showed GPT-3.5 with an agent loop solved 95% of coding tasks, versus 67% by GPT-4 in one-shot mode. It's often more effective to "think cheaper model through a task" than to jump to a more powerful model without reasoning. However, this comes at a cost to latency and throughput. Each step is a new LLM inference; a complex query might take 5–10 cycles, 5–10x slower than a single call. As LangChain's team notes, reflection and planning trade a bit of extra compute for better output quality, and may be unsuitable for low-latency applications. There's also a higher chance something goes wrong in a long chain (e.g. a tool fails or the model veers off-topic). Careful design is needed to reap the benefits while managing these downsides.

Frameworks

A number of frameworks make it easier to build and manage these single-agent reasoning loops. LangChain provides out-of-the-box agents (including a ReAct implementation) where you simply define the tools and the LLM, and it handles the loop and prompt formatting. It also offers memory components (for maintaining state) and integrations with tracing/logging. Semantic Kernel (Microsoft) introduces the concept of a Planner – essentially asking the LLM to select a sequence of function calls from available plugins, which aligns with this pattern. Open-source libraries like LangChain's LangGraph allow you to explicitly define the chain-of-thought flow as a graph of nodes (steps) for more control.

Self-Reflective Agents (Reflexion Pattern)

Augmenting an agent with the ability to critique and refine its own outputs. In this pattern, the AI doesn't just forge ahead blindly from question to answer – it pauses to reflect on its solution and possibly correct mistakes. A separate "reflection" phase (which can be implemented as an extra LLM call or an internal prompt) reviews the agent's work and provides feedback, which the agent then uses to improve its answer. This idea, introduced in frameworks like Reflexion, brings a flavor of human-like self-evaluation or a "quality assurance" step into the agent's reasoning process. The benefit is higher reliability and fewer errors (especially on tasks like code generation or logical reasoning), at the cost of extra computation and complexity.

Architecture

A typical reflexive agent has at least two LLM invocations in sequence: one as the Actor (generating a draft solution or answer), and one as the Evaluator (critiquing that solution).

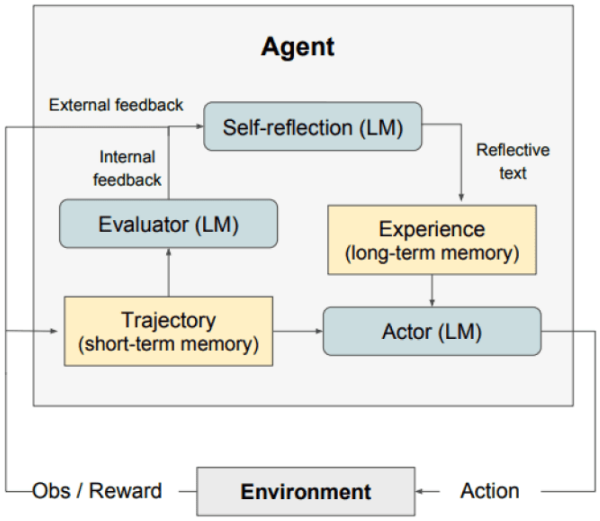


Figure 24. Reflexion pattern.

The evaluator might be a distinct LLM or just a different prompt to the same model, often tasked with finding flaws, mistakes, or unmet requirements in the actor's output. The feedback can be simple (a textual critique) or structured (e.g. a list of errors, or a score). The agent then either revises the answer itself or invokes a fixer LLM to do so, using the feedback. This yields a new answer, which can optionally loop back for another round of critique until it meets some criteria of quality or a max number of iterations. In summary: *Generate → Reflect → (Revise)*, repeated as needed. Concretely, Shinn et al.'s *Reflexion* framework had one LLM produce solutions and another produce "verbal feedback" which was added to the agent's memory to influence the next attempt. Other realizations include "chain-of-thought with self-consistency" (generate multiple answers and pick the best) and "tree-of-thoughts" (systematically explore solution branches), but the Reflexion pattern specifically emphasizes learning from feedback over multiple tries.

Frameworks

Several tools help implement reflection loops. LangChain (LangGraph) provides templates for reflection – essentially combining a generator node and a reflector/critic node in a directed loop. The LangChain blog outlines "Basic reflection" and a full Reflexion implementation where the reflector acts like a teacher giving feedback to the student agent. These frameworks handle the bookkeeping of alternating between agents and deciding when to stop. Alternatively, one can manually orchestrate: after your agent produces an output, send that (and the original query) to an LLM with a prompt like "Critique the above solution. If there are errors or improvements, explain them." Then append the critique to the context and have the original agent try again.

Autonomous Task Planning Loop (AutoGPT-Style)

An agent (or group of agents) that can autonomously decompose a high-level goal into sub-tasks, execute them (using tools), and iterate until the goal is achieved. This pattern emerged with systems like AutoGPT and BabyAGI, which captivated the community by showing AI agents that can "decide what to do next" without human prompts at each step. Essentially, the agent manages a **to-do list** or plan: it generates tasks needed to reach an objective, carries them out one by one (using the earlier patterns of tool use and reasoning), and dynamically updates the plan. This continues in a loop until the objective is met or no further tasks remain. In effect, it's a **self-driving workflow**. The promise is enormous – truly autonomous multi-step agents – but it also amplifies challenges in control, efficiency, and safety, so robust implementation is key. Autonomous task agents are essentially automated orchestrators. They unlock the full potential of agentic AI by reducing the need for human prompts at each step. But with great power comes great responsibility – these agents must be sandboxed, monitored, and iteratively improved.

Architecture

A straightforward implementation uses two main LLM roles: a Planner and an Executor (sometimes called a "Solver"). The Planner takes the high-level goal (and possibly the latest context) and outputs a list of tasks or an updated plan. The Executor then picks the next task, performs it (which could involve calls to tools or other actions), and produces an outcome. The outcome is fed back into the Planner to inform the next planning iteration. This forms a feedback loop: *Plan → Execute → Update → (Plan)* until done. Some systems merge these roles and have a single agent do both planning and execution interleaved (e.g. the agent's chain-of-thought concludes with either an action or a declaration that a sub-goal is achieved). In practice, separating them can be cleaner and allow using different models or prompts for each – the Planner focused on strategy, the Executor on completing a given task. Advanced variants, such as Microsoft's recent "Plan-Execute-Reflect" agents, even insert a reflection phase to evaluate progress and adjust strategy if needed (similar to an outer loop of Reflexion applied to the overall task plan). In fact, Microsoft's Magentic-One generalist agent employs *two nested loops*: an inner loop to monitor progress on the current plan, and an outer loop to rethink the plan if the agent is stalling. This ensures the agent can recover when a naïve plan isn't working – an important robustness feature.

Frameworks

Several orchestration frameworks have built-in support for this pattern. LangChain introduced a "Plan-and-Execute" agent where an LLM first produces a plan (as a series of tool calls or steps), then either executes them itself or delegates each step to another agent call. AutoGen by Microsoft allows you to create an agent that can spawn new tasks and even spawn new agents to handle them. CrewAI (open source) focuses on multi-agent collaboration, which can be adapted for a planner/worker style interaction – essentially one agent assigns tasks to others. The original open-source AutoGPT is basically a specific implementation of this loop using GPT-4: it keeps a list of objectives and completed tasks and runs continuously until done (or until a user-defined threshold). It also had rudimentary methods to self-evaluate if it's off track, which aligns with the reflect-and-replan concept. Tools like Semantic Kernel provide a Planner plugin that can interpret an objective and break it into function calls, which is another way to approach high-level task planning in a controlled manner. When implementing from scratch, a major design question is: *How free-form vs constrained should the agent's planning be?* Free-form might let the LLM make any plan it wants (e.g. "Goal: cook dinner; Plan: 1) find recipes, 2) make a shopping list, 3) buy ingredients, …"). A constrained approach might define certain plan schema or use hard-coded subtasks that the LLM just fills details for. Constrained planning is easier to trust in production (fewer surprises) but less "autonomous" in the general sense.

Multi-Agent Collaboration

Using multiple AI agents working together, typically with specialized roles, to solve a problem. Instead of a single monolithic agent doing everything, multi-agent systems divide the cognitive labor: one agent might be a Planner or Orchestrator, others are specialized executors (e.g. a coder agent, a web browsing agent, a calculator agent), or alternatively, agents might play different expert roles (e.g. a Doctor agent and a Lawyer agent collaborating on a case that has medical and legal aspects). They communicate and coordinate to reach a solution. This pattern draws inspiration from human teams and promises more modular, extensible AI systems. By encapsulating distinct skills in separate agents, you can reuse and maintain them more easily (much like microservices vs a monolithic app). However, it introduces communication overhead and requires a coordination mechanism to prevent chaos.

Architectures

There are a few common architectures for multi-agent systems:

- **Orchestrator + Specialists:** One agent is in charge of delegating tasks to one or more other agents. This orchestrator (or "manager") agent decides which specialized agent is best for each subtask. For example, the orchestrator parses a user request, and for a given step, if it requires coding, it invokes the Coder agent; if it requires a web search, it invokes the Researcher agent, etc. The orchestrator then integrates their results. Microsoft's Magentic-One system follows this: a lead orchestrator coordinates four specialist agents (Coder, WebSurfer, etc.) and tracks the overall progress. This architecture centralizes decision-making, which simplifies global oversight and avoids uncontrolled agent chatter. It's akin to the autonomous planning loop but with different agents executing different steps.
- **Peer Collaboration (no fixed leader):** Multiple agents converse or negotiate to reach a solution. For example, two agents could engage in a debate: one generates a solution, the other critiques (like an implicit Reflexion but between distinct agents rather than one agent self-reflecting). Or a group of agents brainstorm collectively, each

contributing knowledge. In this setup, you might still need a mechanism to eventually consolidate the outcome (which could be a special "moderator" agent or a simple rule like take the most confident answer). This approach is more emergent – the system's behavior is less scripted, which can sometimes yield creative results, but also risks instability (agents can confuse each other or go in circles if not carefully prompted). OpenAI's experiments with having GPT-4 talk to itself in roles, or the Socratic Models approach, fall in this category.

- **Environment-mediated Agents:** Here, agents don't communicate directly but via an environment (a shared memory, a board, a document, etc.). Each agent observes the environment and acts on it. A recent example is the "Generative Agents" paper where agents act in a simulated world (writing to a common memory about what they're doing) – not exactly a problem-solving case, but a paradigm for multi-agent interaction. In a more task-oriented sense, environment mediation could be as simple as a shared spreadsheet that multiple agent scripts edit until a solution emerges. This reduces direct dialogue management but requires designing the environment and rules of interaction.

Pattern	Description	When to Use	Latency/Cost
LLM Reasoning + Tool-Use (ReAct)	Single LLM with stepwise reasoning and tool calls	Great for moderate complexity tasks requiring external info or computations. Reuses powerful LLM reasoning for control	**Medium** – multiple LLM calls per query (usually 2–10). Cost grows with steps.
Self-Reflective (Reflexion)	LLM generates answer, then self-critiques and improves it in extra iterations	For high accuracy needs (code, finance, legal) where the extra time is justified to catch mistakes	**High** – 2x to 5x the calls of single-pass. Higher compute cost, higher latency
Autonomous Task Loop (AutoGPT)	One (or few) agents autonomously plan and execute a sequence of tasks until goal done	When you have a complex, open-ended objective that can be broken down (e.g. multi-step workflows, research tasks) and you want minimal human intervention	**Very High** – could be dozens of LLM calls, minutes or more runtime. Cost can scale linearly with task complexity
Multi-Agent Collaboration	Multiple specialized agents communicating or coordinated by a leader	When distinct skills or parallelism are needed – e.g. multimodal problems, or to modularize system by expertise. Also for experimental setups where agents verify each other	**High** – overhead of inter-agent communication (many messages). If run in parallel, wall-clock time can be moderate, but total compute is high

Table 1. Comparison of Patterns and Decision Guidelines.

In practice, these patterns are not mutually exclusive. Many real systems **combine patterns** to balance their strengths. For example, a multi-agent system might also use the Reflexion pattern internally (each agent self-refines before sending output to others), or an AutoGPT-style agent might spin up a temporary helper agent (multi-agent hybrid) for a specific subtask. It's useful to think of these as **building blocks** that can be composed.

Guidelines to Choose a Pattern:

- **Simple query or single-step task?** Use a straightforward tool call or Chain-of-Thought prompt. No need for full agent loop. *E.g.* if the user only asks for today's weather, an LLM calling a weather API once suffices.
- **Complex but bounded task that can be done in one session?** A single-agent ReAct is often easiest to deploy. It can handle multi-hop reasoning and tool use within one conversational session. This covers many "assistant" use cases (research Q&A, customer support answers, etc.).

- **Need high confidence in correctness?** Introduce a reflection/refinement step. Especially for things like code or math – the Reflexion pattern or a "critic agent" can drastically reduce errors. If latency allows, this is a good upgrade to a basic agent.
- **Long-running process with multiple phases?** If the task feels like a project with sub-goals (not just a single Q&A), consider the autonomous task loop. This is for when an agent needs to *remember intermediate results and adapt its plan* – something a simple ReAct agent with a fixed prompt may struggle with. Ensure you can allocate sufficient time/compute and that you put strict guardrails around it initially.
- **Requires diverse expertise or parallel work?** Multi-agent might perform best here. For example, an application that involves vision, language, and database queries might naturally map to different agents that specialize in each, overseen by a coordinator. Or if you want redundancy (two agents double-checking each other's answers), that's a multi-agent debate pattern. Use orchestrator+specialists design to maintain control.
- **Unstructured exploratory scenarios?** If you're in R&D or building an experimental agent that you want to be very flexible (maybe an agent that can spawn new agents, or try creative approaches), a multi-agent sandbox can be interesting. But for productization, try to impose structure once you identify common patterns.

3.2.5.4. Challenges of Agentic Architectures

1. Increased Complexity and Cost
   Running multiple prompts, extended sessions, or concurrent models raises inference costs and latency.
2. Error Amplification
   Agents that iterate and make decisions autonomously can compound small inaccuracies, leading to more significant issues if not carefully monitored or sandboxed.
3. Security and Governance
   Opening an agent's ability to call external tools must be carefully controlled. Sensitive tasks often require on-premises or virtual sandbox isolation to prevent misuse.
4. Reliability and Transparency
   Debugging tool usage or memory states can be more involved when the system decides how and when to invoke resources.
5. Alignment and Ethical Concerns
   Autonomous decision-making raises concerns about bias, compliance, and oversight. Incorporating robust guardrails, human feedback loops, or specialized alignment techniques is essential.

In many settings, a simple orchestrated workflow is preferable. If you do implement agentic designs, prototype thoroughly in safe environments, add guardrails (like user-in-the-loop checkpoints or maximum iteration limits), and measure whether the flexibility truly offsets the extra complexity.

3.3. AI Validation

Model validation is an essential step in machine learning or artificial intelligence system development. It helps ensure the model performs well with new, unseen data and operates as intended.

Insufficient model validation reduces confidence in its ability to generalize to unprocessed data. Furthermore, validation helps identify the best model, parameters, and accuracy criteria for a given task.

Model validation also helps uncover potential problems before they escalate. It allows for comparison of several models, enabling the selection of the most appropriate one for the specific need. Moreover, it helps evaluate the model's accuracy when exposed to new data.

Model evaluation provides a crucial level of certainty for enterprises operating in dynamic technological environments where actions can significantly impact market dynamics. In machine learning, model evaluation is a systematic, metrics-based method for assessing a model's effectiveness.

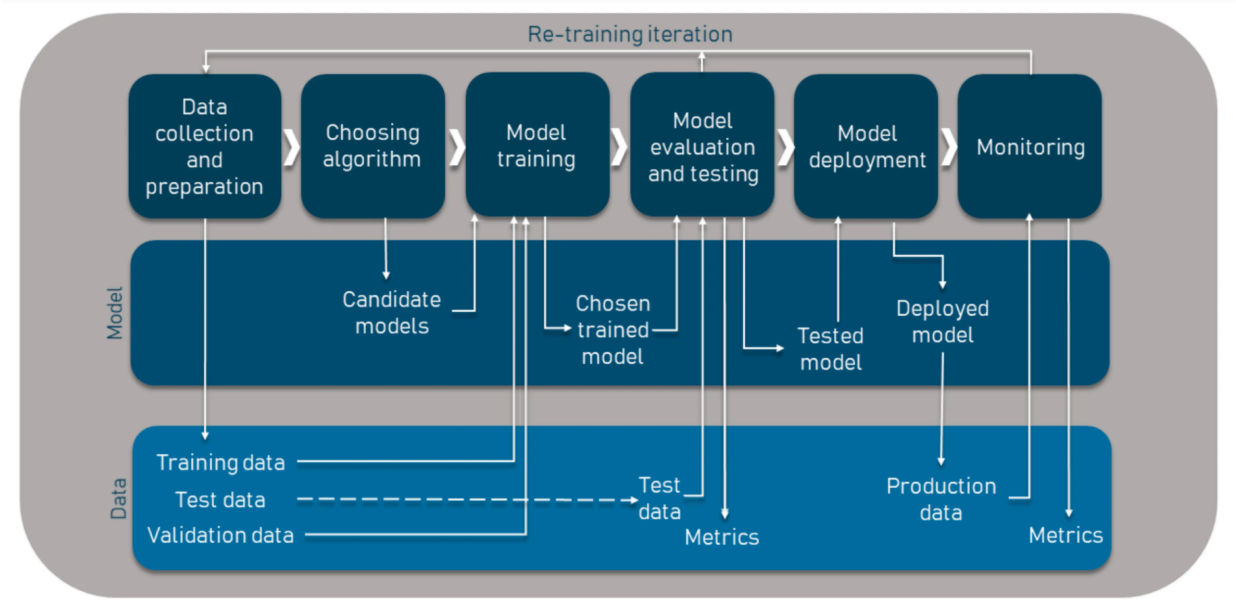


Figure 25 – Model Development and Validation Activities

3.3.1. Classical ML Model Validation Techniques

3.3.1.1. Validation Metrics

Evaluating machine learning models requires careful selection of appropriate metrics. These metrics provide insights into a model's performance and help guide optimization efforts. The choice of metric often depends on the type of problem being addressed – whether it's a regression task predicting continuous values or a classification task assigning data points to categories. Below are some common evaluation metrics, categorized by their applicability.

Error-based Metrics (for Regression)

- **Mean Squared Error (MSE):** Measures the average squared difference between predicted and actual values.
- **Mean Absolute Error (MAE):** Measures the average absolute difference between predicted and actual values.
- **Root Mean Squared Error (RMSE):** The square root of MSE helps interpret errors in the same units as the target variable.

Classification-specific Metrics

- **Accuracy:** Proportion of correct predictions.
- **Precision:** Proportion of positive predictions that are actually positive.
- **Recall:** Proportion of actual positives that are correctly identified.
- **F1-Score:** Harmonic mean of precision and recall, balancing both metrics.

3.3.1.2. Train-Test Split

A train-test split is a method of machine learning model validation that uses new, unseen data to replicate the model's behavior under testing conditions.

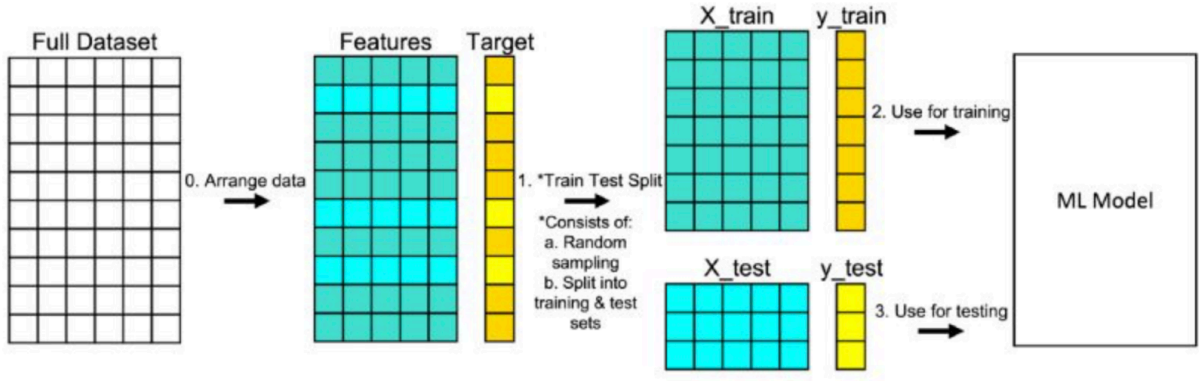


Figure 26 – Train-Test Split

This process involves creating two datasets: a training set (used to develop the model) and a testing set (used for model evaluation). While straightforward, if the split doesn't accurately reflect the general data distribution, the resulting estimations may be unreliable.

3.3.1.3. K-Fold Cross-Validation

When performing a train-test split, K-fold cross-validation divides the data into 'k' equal sections, as depicted on the diagram below.

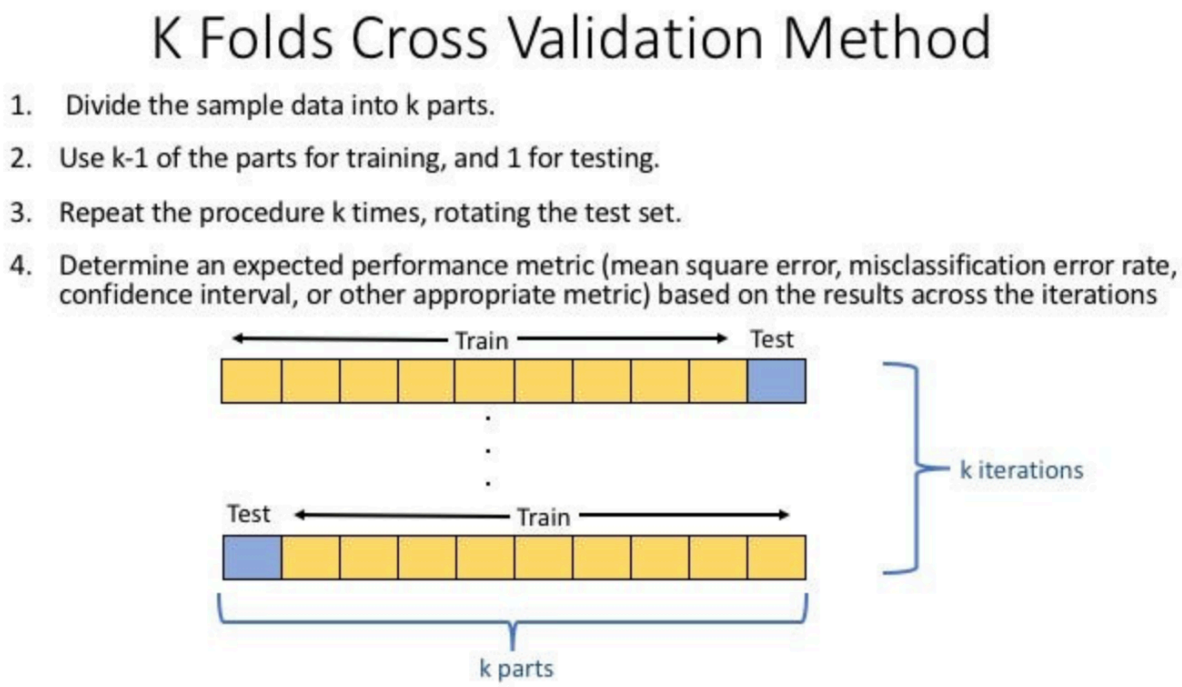


Figure 27 – K-Fold Cross-Validation

3.3.1.4. Stratified K-Fold Cross-Validation

Stratified K-fold cross-validation is a technique where data is shuffled and then separated into 'n' portions. This ensures that each portion contains a representative share of the dataset, helping to mitigate any imbalances that may have occurred during the training process.

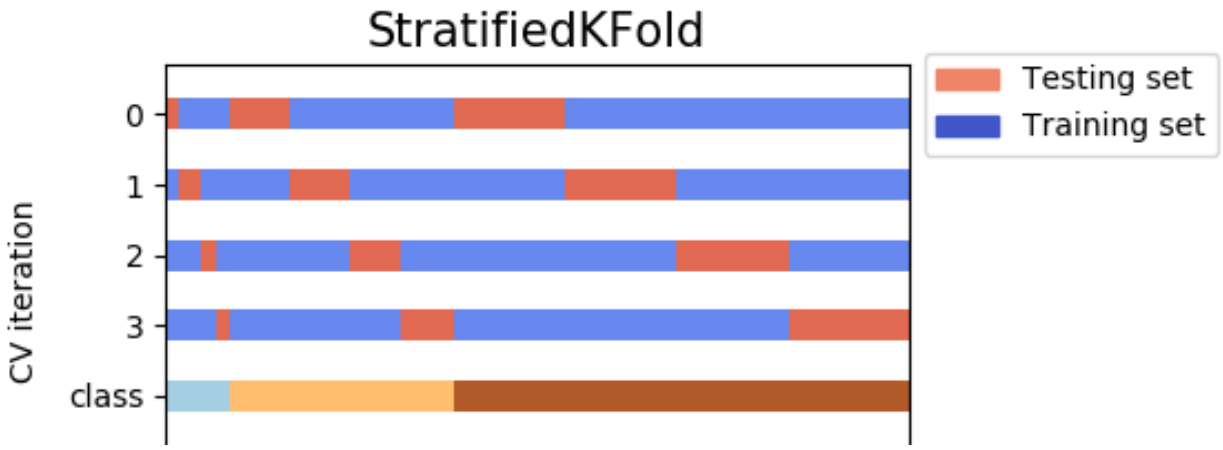


Figure 28 – Stratified K-Fold Cross-Validation

Stratified K-fold cross-validation helps prevent the model from being biased towards the majority class and provides a more accurate assessment of its performance across all classes.

3.3.1.5. Leave-One-Out Cross-Validation

Leave-One-Out Cross-Validation (LOOCV), a variant of K-fold Cross-Validation, is a popular method where the entire dataset is split into folds. In this approach, each data point becomes its own test set, and the model is trained on the remaining data.

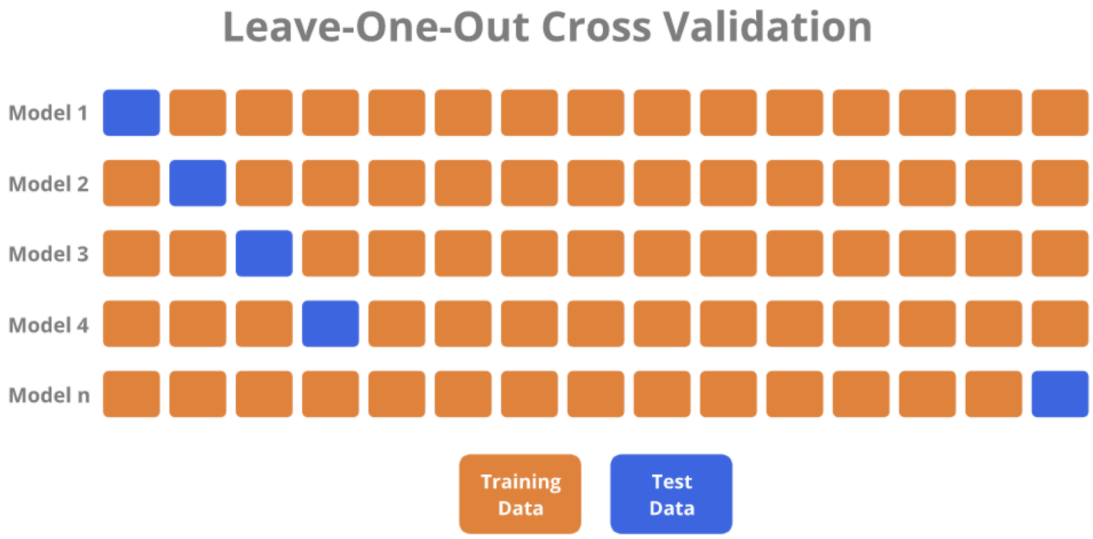


Figure 29 – Leave-One-Out Cross-Validation

While LOOCV offers the most accurate performance estimate, it can be computationally expensive for large datasets.

3.3.1.6. Holdout Validation

Holdout validation is a process of reserving some data for review, similar to a train-test split. However, this data is kept separate throughout the entire training process and evaluated only after a final model is developed.

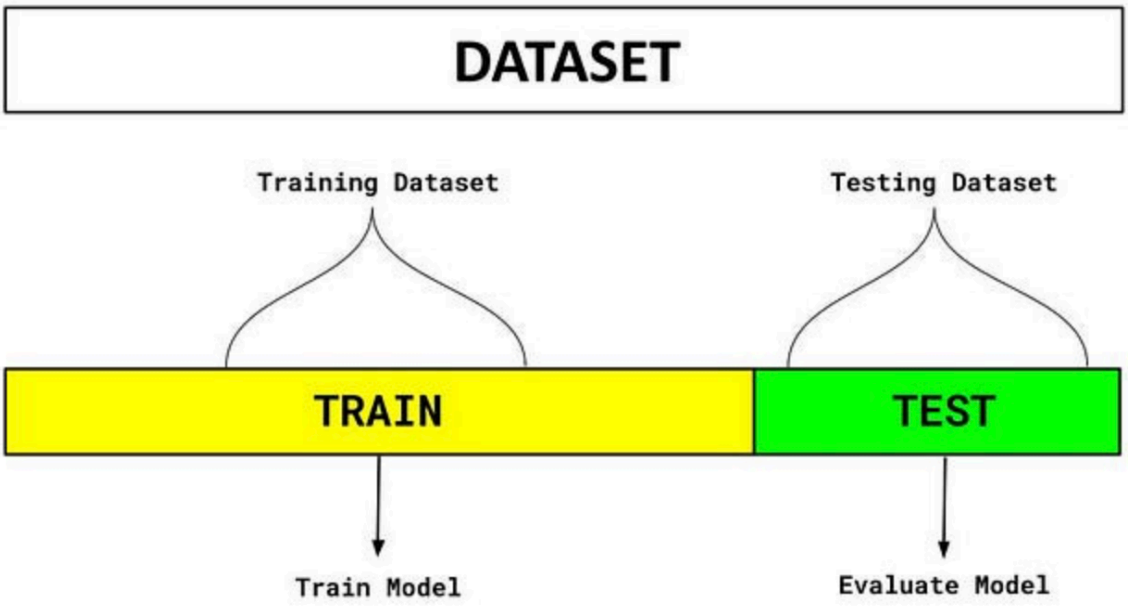


Figure 30 – Holdout Validation

This approach can be particularly beneficial for constantly updating datasets, as the holdout set allows for an assessment of the model's performance on the most recent data.

3.3.1.7. Time Series Cross-Validation

In time series analysis, cross-validation involves using overlapping windows. The model is trained on one window and evaluated on the next, moving sequentially through the data. This approach accounts for the natural temporal dependencies found in time series data and provides a more realistic evaluation of the model's ability to predict future values.

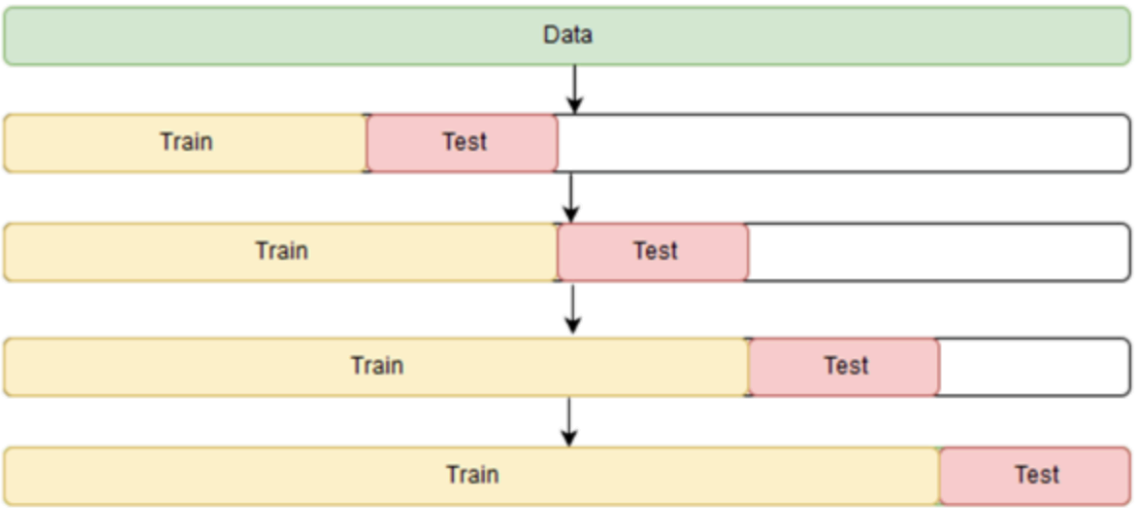


Figure 31 – Time-Series Cross-Validation

3.3.1.8. Handling Imbalanced Datasets During Validation

For imbalanced datasets, conventional metrics like accuracy, precision, and recall can be misleading and distort model validation. Dataset bias can lead to an inaccurate representation of the model's performance.

In such situations, it is crucial to use metrics specifically designed for imbalanced datasets, such as:

- **F1-score:** Balances precision and recall by calculating their harmonic mean, providing a more comprehensive evaluation of both.
- **G-Mean:** Calculates the geometric mean of sensitivity for each class, offering a better overall performance picture across all classes.
- **AUC-ROC:** Assesses the model's ability to differentiate between classes, providing a reliable evaluation regardless of class distribution.
- **Precision-Recall Curves:** Visualizing the trade-off between precision and recall across multiple thresholds provides a better understanding of the model's performance under diverse conditions.

3.3.2. Classical ML Model Validation Best Practices

3.3.2.1. Select The Right Validation Technique

The validation process, tailored to the characteristics of the model and dataset, is essential for building reliable and robust machine learning models. Choosing the appropriate validation strategy depends on factors such as dataset size, data distribution, and the presence of imbalanced classes. These considerations are key to accurately assessing a model's generalization ability to unseen data.

For **large datasets**, straightforward techniques such as the **2-way holdout method (train/test split)** are generally sufficient, with **confidence intervals** calculated using **normal approximation**.

For **small datasets**, more robust validation techniques are required. These include **(repeated) k-fold cross-validation** or **leave-one-out cross-validation**, typically performed **without a separate independent test set**. To quantify model uncertainty, confidence intervals can be computed using the **0.632 or 0.632+ bootstrap method**.

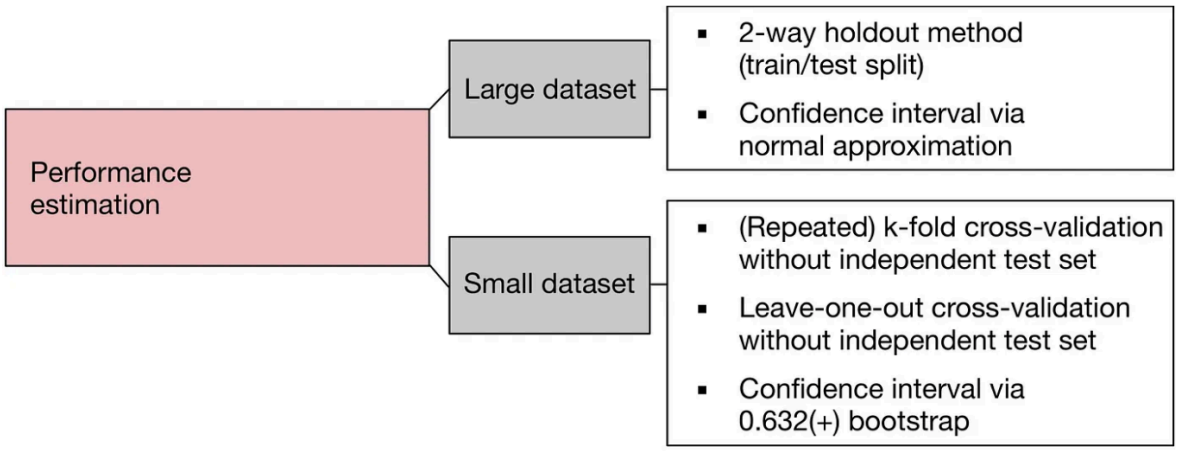


Figure 32 – Metrics by Data Set Size

3.3.2.2. Tune & Benchmark Hyperparameters

After selecting an appropriate validation approach, hyperparameter tuning should be performed using methods such as **grid search**, **random search**, or **Bayesian optimization**. Comparing model performance against benchmark datasets and standardized models helps contextualize results and ensure meaningful evaluation.

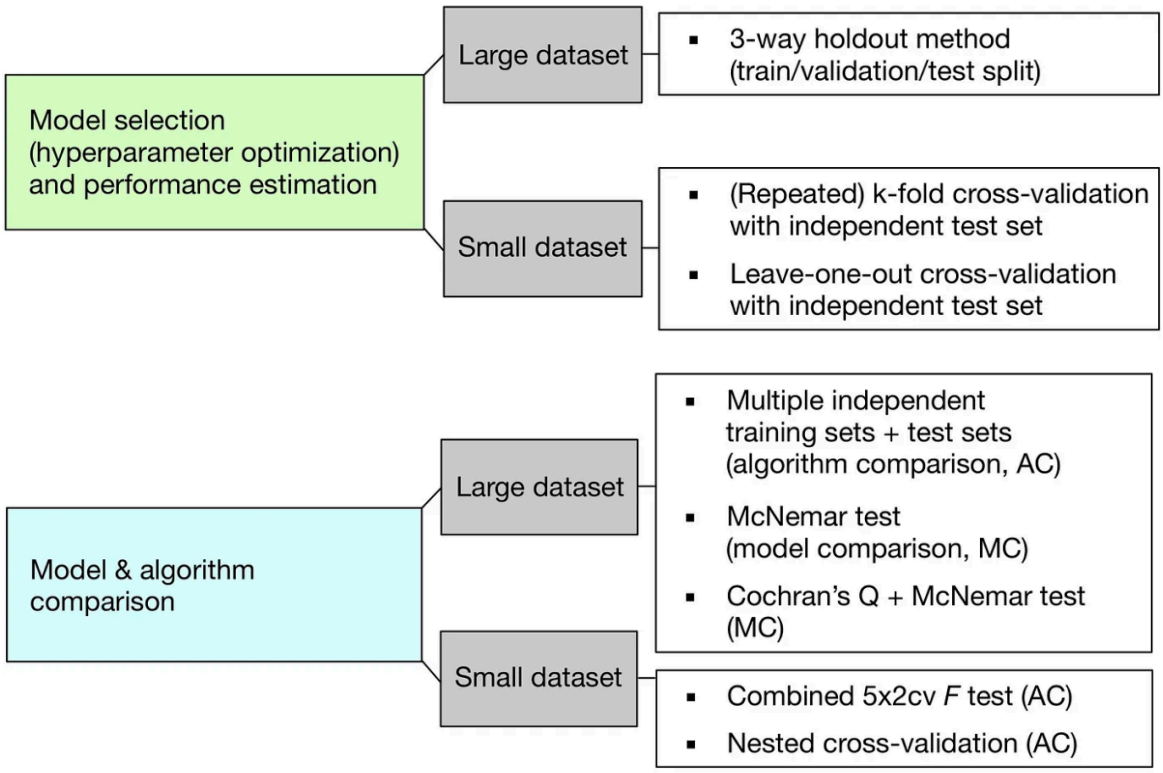


Figure 33 – Evaluation Strategies for Model Selection and Algorithm Comparison Based on Dataset Size

3.3.2.3. Select a diverse set of metrics to evaluate performance

Evaluation metrics should be tailored to the type of task, learning paradigm, and model architecture. Below is a categorized list of best-practice metrics and validation considerations:

Supervised Learning	Regression Models	• RMSE (Root Mean Squared Error): Measures the square root of the average squared differences between predicted and actual values. • MAE (Mean Absolute Error): Averages the absolute differences between predictions and ground truth. • R^2 (Coefficient of Determination): Indicates the proportion of variance in the dependent variable explained by the model. • MAPE (Mean Absolute Percentage Error): Averages the absolute percentage difference between predicted and actual values.
	Classification Models	• Accuracy: Proportion of correct predictions over the total number of samples. • Precision, Recall, F1 Score: • *Precision:* Of all predicted positives, how many are truly positive? • *Recall:* Of all actual positives, how many did we correctly predict? • *F1 Score:* Harmonic mean of precision and recall. • ROC-AUC (Receiver Operating Characteristic – Area Under Curve): Measures model's ability to distinguish between classes across different thresholds. • PR-AUC (Precision-Recall AUC): Focuses on positive class performance in imbalanced datasets.

Unsupervised Learning	Clustering	• Silhouette Coefficient: Evaluates how similar points are to their own cluster versus other clusters. • Davies-Bouldin Index: Assesses average similarity between each cluster and the cluster most similar to it (lower is better). • Calinski-Harabasz Index: Measures within-cluster dispersion vs. between-cluster separation (higher is better). • Adjusted Rand Index (ARI): If ground truth labels are available, compares clustering similarity to true labels.
	Anomaly Detection	• Precision-Recall Curves, ROC Curves: Evaluate how well the model distinguishes anomalies from normal data. • F1 Score: Specifically for the anomaly class, measuring both precision and recall. • AUC (Area Under Curve): Reflects the model's ability to rank anomalies higher than normal instances.
	Neural Networks for Unsupervised Tasks (e.g., autoencoders)	• Reconstruction Error: Measures how well the autoencoder reconstructs input data (lower error is better). • Topographic/Quantization Error (Self-Organizing Maps): Quantifies how well the SOM preserves data topology.
Hybrid Models	Hybrid Models	• Accuracy, Precision, Recall, F1 Score (Classification) or RMSE, MAE (Regression) • Evaluate overall performance of the combined system. • Out-of-Bag Error (for Random Forest-like hybrids): If ensembles are part of the hybrid, OOB error gives an unbiased performance estimate. • Data Leakage Checks: Ensures that the combined approach isn't inadvertently accessing information from the validation/test set.
Deep Learning Models	Convolutional Neural Networks	• Accuracy, Top-k Accuracy: Common for image classification tasks. • Loss Curves (Training vs. Validation): Track overfitting/underfitting by monitoring cross-entropy or MSE loss.
	RNNs/Transformers	• Perplexity: Measures predictive performance in language modeling. • BLEU, ROUGE: For text generation or summarization tasks.
	Reinforcement Learning	• Cumulative Reward: Sum of rewards over an episode to gauge policy performance. • Policy Evaluation Metrics: Such as average reward per step, success rate in simulated tasks.
Random Forest Models	Random Forest Models	• Accuracy, Precision, Recall, F1 Score (Classification): Evaluate the ensemble's performance on labeled data. • RMSE, MAE (Regression): Assess how close predictions are to actual values. • Out-of-Bag Error (OOB): Internal performance measure that uses samples not included in each decision tree's bootstrap sample.
Support Vector Machines	Support Vector Machines	• Accuracy, Precision, Recall, F1 Score (Classification): Evaluate boundary separation quality. • ROC-AUC: Measures classification ability across varying thresholds. • RMSE, MAE (Regression): For SVR (Support Vector Regression) tasks.
Neural Network Models	Neural Network Models	• Accuracy, Precision, Recall, F1 Score (Classification): Evaluate classification quality.

		• RMSE, MAE, R^2 (Regression): Gauge how close predictions are to the ground truth. • Loss Curves & Early Stopping: Track training vs. validation loss to detect overfitting.
K-Nearest Neighbors Models	K-Nearest Neighbors Models	• Accuracy, Precision, Recall, F1 Score (Classification): For classification tasks. • RMSE, MAE (Regression): For regression tasks. • Distance Metric Evaluation: Compare performance with Euclidean vs. Manhattan or other distance metrics. • Confusion Matrix: Helps visualize classification performance.
Bayesian Models	Bayesian Models	• Log-Likelihood or Posterior Predictive Checks: Evaluate how well the model fits observed data. • Bayesian Information Criterion (BIC), Deviance Information Criterion (DIC): Provide model fit comparisons under Bayesian frameworks. • Convergence Diagnostics: Trace plots, Gelman-Rubin statistic to ensure Markov Chain Monte Carlo (MCMC) convergence.
Clustering Models	Clustering Models	• Internal Validation: Silhouette Coefficient, Davies-Bouldin Index, Calinski-Harabasz Index. • External Validation: Adjusted Rand Index (ARI), if ground truth labels exist. • Stability & Scalability: Evaluate cluster stability across multiple runs and data sizes.

Best Practices for Model Validation

- **Split the data** using appropriate validation strategies (e.g., k-fold cross-validation, bootstrap, holdout).
- **Iterate on validation** as a continuous process throughout model development.
- **Document the validation process, metrics, and results** clearly to ensure transparency and reproducibility.
- **Incorporate interpretability and explainability** into the validation and evaluation pipeline.
- **Control for bias and fairness** using detection techniques and metrics such as demographic parity and equalized odds.
- **Benchmark model performance** against standard datasets and strong baseline models.
- **Include validation results** in a formal Model Evaluation Report with performance dashboards and validation curves.
- **Verify data quality** by assessing completeness, accuracy, consistency, and representativeness; document findings in a Data Quality Report or annotated data sheet.
- **Establish automated monitoring and retraining pipelines** to ensure model performance over time.
- Include to Data Quality and integrity report verification results about data completeness, accuracy, and consistency, also results about **Representativeness & Bias Analysis.**
- Document data sources, preprocessing, and limitations into *Data Quality Report* and annotated data sheets.

3.3.3. GenAI Model Validation Techniques

3.3.3.1. Validation Metrics

Response correctness, semantic similarity, and hallucination, among other LLM evaluation metrics, assess an LLM system's output against relevant standards. These metrics are crucial for LLM evaluation as they help measure the performance of various LLM systems, including the LLM itself.

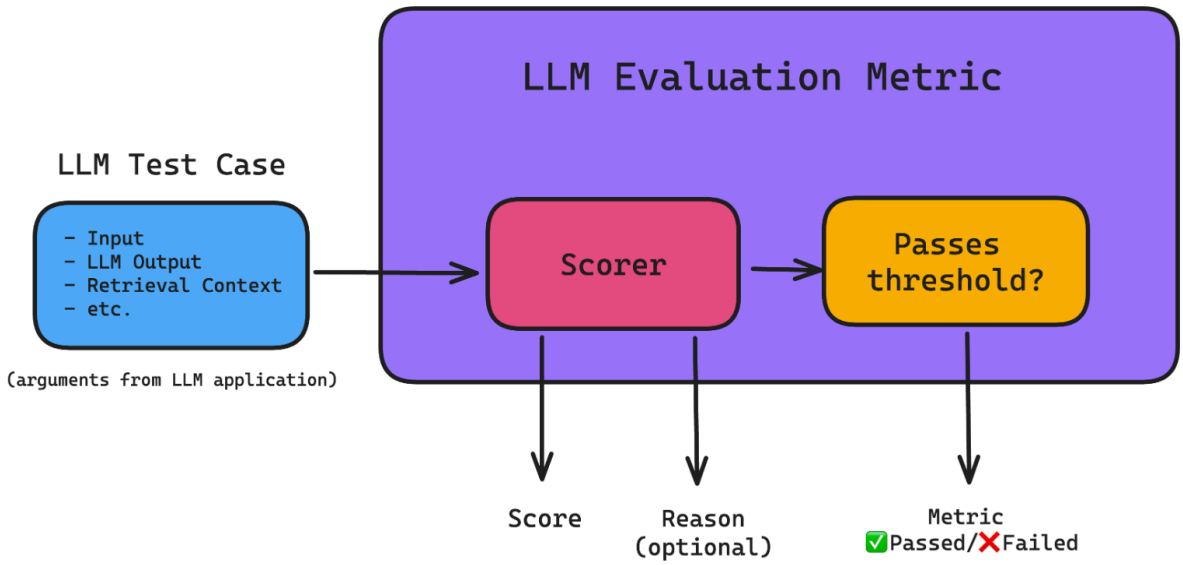


Figure 34 – An LLM Evaluation Metric Architecture

The most important and common metrics to consider before launching an LLM system into production typically include the following:

- **Answer Relevance:** Determines whether an LLM output effectively addresses the given input in an informative and concise manner.
- **Prompt Alignment:** Determines whether an LLM output accurately follows instructions from the prompt template.
- **Correctness:** Determines whether an LLM output is factually accurate based on a known ground truth.
- **Hallucination:** Determines whether an LLM output contains fabricated or made-up information.
- **Contextual Relevance:** Determines whether the retriever in a RAG-based LLM system is able to extract the most relevant information to provide as context for the LLM.
- **Responsible Metrics:** Includes metrics such as bias and toxicity, which assess whether an LLM output contains generally harmful or offensive content.
- **Task-Specific Metrics:** Includes metrics such as summarization, which usually involves custom criteria depending on the specific use case.

Metric Scorer Types

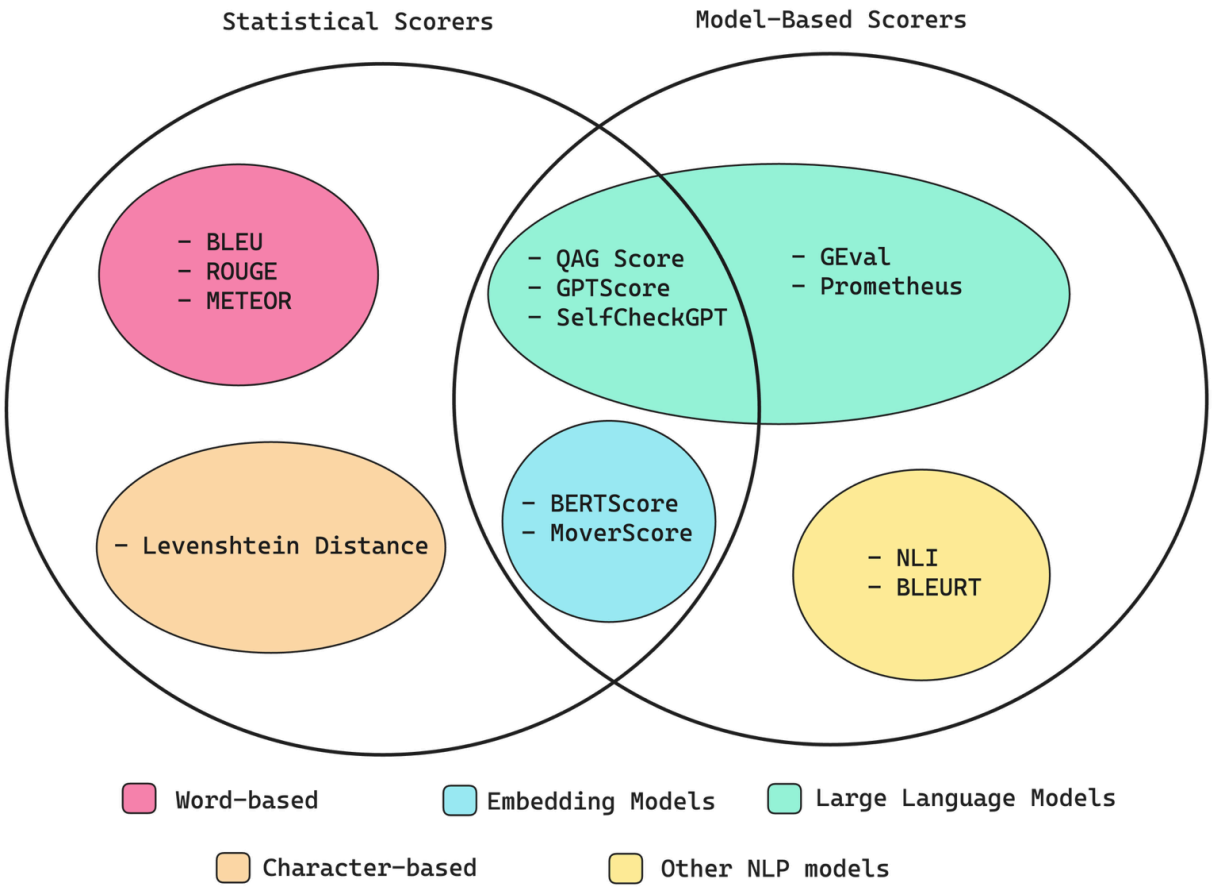


Figure 35 – An LLM Evaluation Metric Architecture

Statistical Scorers:

- The BLEU (BiLingual Evaluation Understudy) scorer evaluates an LLM application's output against annotated ground truths (or expected outputs). It calculates the precision for each matching n-gram (n consecutive words) between an LLM output and the expected output, calculates their geometric mean, and applies a brevity penalty if needed.
- The ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scorer is primarily used for evaluating text summaries from NLP models and calculates recall by comparing the overlap of n-grams between LLM outputs and expected outputs. It determines the proportion (0–1) of n-grams in the reference that are present in the LLM output.
- The METEOR (Metric for Evaluation of Translation with Explicit Ordering) scorer is more comprehensive as it calculates scores by assessing both precision (n-gram matches) and recall (n-gram overlaps), adjusted for word order differences between LLM outputs and expected outputs. It also leverages external linguistic databases like WordNet to account for synonyms. The final score is the harmonic mean of precision and recall, with a penalty for ordering discrepancies.
- The Levenshtein distance (or edit distance, which you may recognize as a LeetCode hard DP problem) scorer calculates the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one word or text string into another. This can be useful for evaluating spelling corrections or other tasks where precise character alignment is critical.

Statistical scorers are reliable yet imprecise, as they fail to consider semantics.

Model-Based Scorers (Non LLM based):

- **The NLI scorer**, which uses Natural Language Inference models (which is a type of NLP classification model) to classify whether an LLM output is logically consistent (entailment), contradictory, or unrelated (neutral) with respect to a given reference text. The score typically ranges between entailment (with a value of 1) and contradiction (with a value of 0), providing a measure of logical coherence.
- **The BLEURT** (Bilingual Evaluation Understudy with Representations from Transformers) scorer, which uses pre-trained models like BERT to score LLM outputs on some expected outputs.

These scorers can produce inconsistent results. For example, NLI scorers may struggle with long texts, while BLEURT's performance is limited by the quality and representativeness of its training data.

Modern LLM Judge Approaches:

- G-Eval: This framework, inspired by research demonstrating improved human alignment through LLM evaluation, utilizes LLMs to assess LLM outputs. G-Eval excels at creating task-specific metrics and consistently outperforms traditional and non-LLM evaluation methods. Higher Spearman and Kendall-Tau correlations indicate a stronger alignment with human judgment.
- DAG (Deep Acyclic Graph): While G-Eval is well-suited for evaluations requiring subjective assessment, scenarios demanding strict adherence to explicit success criteria benefit from a decision-based scoring approach. Consider, for example, summarizing a patient's medical history for a hospital setting, where specific headings must appear in a defined order. The DAG scorer, an LLM-driven decision tree, is ideal for such cases. Each node represents an LLM judgment, and edges represent conclusions, ultimately leading to a hard-coded score based on the chosen evaluation pathway. G-Eval can also be incorporated as a terminal node to generate scores. By breaking down the evaluation process into granular steps, DAG scoring enables deterministic outcomes. Furthermore, DAG can handle edge cases where an LLM output fails to meet minimum evaluation criteria, such as improper formatting; in these instances, G-Eval can be used as a leaf node.
- Prometheus: This open-source solution offers capabilities comparable to GPT-4's evaluation performance, provided the evaluation suite includes appropriate reference materials (reference answers and scoring rubrics). Like G-Eval, Prometheus utilizes a fine-tuned Llama-2-Chat model, trained on 100,000 feedback examples generated by GPT-4. The dataset is publicly available on Hugging Face within the Feedback Collection. While sharing core principles with G-Eval, Prometheus differs in several key aspects:
  - G-Eval is a framework using GPT; Prometheus is an LLM fine-tuned for evaluation.
  - G-Eval generates evaluation steps using Chain-of-Thought (CoT) prompting; Prometheus incorporates the scoring rubric directly within the prompt.
  - Prometheus requires reference or example evaluation results for effective operation.
- GPTScore: In contrast to G-Eval's form-filling paradigm, GPTScore evaluates LLM outputs by assessing the conditional probability of generating the target text.
- SelfCheckGPT: This simple, sampling-based approach is used for fact-checking LLM outputs. It operates on the principle that hallucinated content is unlikely to be reproducible, whereas responses grounded in factual knowledge will exhibit consistency across multiple samples. This reference-less approach is particularly valuable in production environments.
- Question Answer Generation (QAG) Score: This scoring method leverages LLMs' reasoning capabilities to reliably evaluate LLM outputs. It uses answers (typically "yes" or "no") to close-ended questions (either pre-defined or generated) to calculate a final metric score. The reliability stems from the fact that LLMs are not directly generating the evaluation scores. For example, when assessing faithfulness (the proportion of claims in an LLM output that are accurate and consistent with the ground truth), the number of accurate claims can be divided by the total number of claims made by the LLM. This approach combines LLM reasoning with a deterministic calculation, resulting in accurate and reliable scores.

4. AI Delivery Best Practices

4.1. Deliverables Checklist

This deliverables checklist provides a reference for AI practitioners, delivery managers, and stakeholders, ensuring comprehensive coverage of AI initiatives. It outlines key phases—scoping, discovery, MVP validation, and scaling—and aligns them with common service types: AI strategy, product development, platform implementation, governance, and operating model. Using this checklist helps teams deliver consistent, high-quality AI implementations that are scalable, secure, and impactful across the enterprise. Use it to define scope and verify delivery completeness.

AI Factory - Deliverables Checkli

4.2. AI Validation Strategy

4.2.1. Pre-Deployment Validation

Before deploying any LLM or RAG model, rigorous validation ensures safety, quality, and alignment with requirements:

- **Model Selection & Baseline Testing:** Evaluate candidate models on tasks using diverse datasets and metrics like accuracy, F1, BLEU, ROUGE, or perplexity. Tools like GLUE, HELM, and OpenAI Evals are recommended. Record baseline performance for future comparison.
- **Fine-Tuning Validation:** Ensure fine-tuning improves domain-specific performance without overfitting. Validate robustness across test datasets and retain core base-model strengths. If using RLHF, confirm improvements in helpfulness and safety.
- **Adversarial Robustness Testing**: Stress-test models with adversarial prompts (e.g., toxic provocations, logic puzzles, or prompt injections). Count and address critical failures like harmful content or data leakage.
- **Deployment-Readiness Checks**: Use formal checklists (e.g., https://arxiv.org/pdf/2403.18958) to confirm readiness, including safety, scalability, latency, and performance on real-world queries. Verify alignment with business and non-functional requirements.

4.2.2. Ongoing Monitoring

Ensure deployed models remain reliable through:

- **Drift Detection:** Monitor for input/output data drift. Automated alerts trigger retraining if drift exceeds thresholds. (See: https://www.fiddler.ai/blog/how-to-monitor-llmops-performance-with-drift).
- **Performance Tracking:** Continuously log accuracy, user feedback, and factual correctness. Investigate quality drops (e.g., hallucinations, incoherent responses) and update or roll back as needed.
- **Incident Handling & Feedback Loops:** Establish protocols for addressing harmful outputs or errors. Log incidents and feed failure cases back into fine-tuning or prompt adjustments. Maintain versioning and documentation for traceability.

4.2.3. Metrics & Evaluation Criteria

Effective evaluation spans accuracy, robustness, fairness, safety, and efficiency:

- **Accuracy & Task Performance:** Use task-specific metrics (e.g., F1 for QA, BLEU for translation, hallucination rate for generative models). Complement quantitative measures with human/AI judgment for complex tasks.
- **Robustness & Reliability:** Test resilience to typos, dialects, adversarial inputs, and edge cases. Track failure modes (e.g., data leakage or misclassifications) to improve model reliability.

- **Bias & Fairness:** Assess outputs for demographic parity and harmful stereotypes using fairness toolkits (e.g., Fairlearn). Apply counterfactual fairness tests and qualitative evaluation to mitigate biases.
- **Safety & Alignment:** Measure toxicity, compliance with ethical guidelines, and groundedness (for RAG models). Use tools like Perspective API or AI-based evaluators (e.g., GPT-4 as a judge) to assess safety.
- **Efficiency & Performance:** Track latency, throughput, resource utilization, and cost efficiency. Balance accuracy with runtime performance using benchmarks like HELM.
- **RAG-Specific Metrics:** Evaluate retrieval accuracy (e.g., Recall@K) and groundedness (factual consistency with retrieved sources). Measure combined retrieval and generation quality.

Evaluation combines statistical metrics, heuristic tests, and human/AI judgment. Tools like OpenAI Evals, LM Eval Harness, and Promptfoo enable scalable evaluations. Human spot checks validate AI-based assessments.

4.2.4. Infrastructure and Metrics Automation

Automated evaluations streamline workflows using MLOps tools and cloud platforms:

- **Cloud Services:** AWS SageMaker (e.g., Clarify for bias, Bedrock for AI-driven evaluations), Azure ML (Responsible AI dashboard), GCP Vertex AI (drift detection), and Databricks (MLflow integration) support scalable, automated evaluations.
- **MLOps Pipelines:** Integrate evaluations into CI/CD pipelines for automated testing at each stage. Tools like MLflow, Vertex Pipelines, and GitHub Actions enforce gated approvals based on evaluation outcomes.
- **Monitoring & Alerts:** CloudWatch, Azure Monitor, and Prometheus/Grafana enable real-time tracking and alerting for drift, latency, and quality issues. Open-source libraries like Evidently AI facilitate drift detection and periodic reporting.
- **Open-Source Tools:** Tools like EleutherAI's LM Eval Harness, Hugging Face Evaluate, and AI Fairness 360 enable customized evaluations. Promptfoo supports prompt testing and red-teaming.
- **Data & Version Management:** Version-controlled datasets and MLflow logs ensure reproducibility. Generate evaluation reports for audits and compliance.

4.2.5. Alignment with AI Governance & Compliance Standards

The framework aligns with governance standards like the EU AI Act, NIST AI Risk Management Framework, and ISO 42001:

- **Risk Management:** Systematically identify, test, and document risks (e.g., bias, misinformation) as required by the EU AI Act. Maintain traceability and monitor for post-deployment risks.
- **ISO 42001 Compliance:** Follow ISO principles for risk assessment, documentation, and continuous improvement. Treat evaluations as internal controls for lifecycle governance.
- **Fairness & Transparency Audits:** Conduct fairness audits (e.g., demographic parity tests) and mitigate biases. Publish Model Cards with evaluation results, limitations, and transparency details.
- **Domain-Specific Compliance:** Address sector-specific regulations, such as FDA GMLP for healthcare or GDPR for privacy. Implement mechanisms like source traceability (for RAG) and human-in-the-loop fallbacks for high-stakes decisions.

4.2.6. Deliverables and Documentation

Deliverables provide clear guidance, records, and insights:

- **Evaluation Playbook:** Step-by-step manual for evaluation procedures (e.g., adversarial testing, bias audits) aligned with industry standards.
- **Validation Checklist:** Pre-deployment checklist ensuring all criteria (e.g., accuracy, bias, safety) are met and documented.
- **Pipeline Workflow:** Visual diagram of the evaluation and deployment pipeline, highlighting gates, feedback loops, and responsibilities.
- **Monitoring Dashboard Specs:** Recommendations for real-time dashboards tracking drift, accuracy, safety, and efficiency with alert thresholds.

- **Comprehensive Evaluation Report:** Detailed findings for each evaluation cycle, including metrics, risks, and mitigations. Tailored for technical teams and compliance officers, with citations to standards.
- **References & Appendices:** List of standards, research, and methodologies (e.g., ISO 42001, NIST RMF) used in evaluations.

These deliverables ensure that models are thoroughly tested, compliant, and ready for production while providing stakeholders with transparent evidence of quality and risk management.

APPENDIX A: Glossary of Terms

TERM	DEFINITION
Agile	An umbrella term for iterative and flexible approaches to software or project development that emphasize rapid delivery, team collaboration, and adaptability to changing requirements. Agile methods prioritize continuous improvement and customer feedback over rigid planning.
AI Factory	An organizational and technical framework designed to consistently deliver AI-driven products that generate business value, while effectively managing risks and optimizing costs. It integrates data, models, infrastructure, human expertise, and operational processes into a cohesive system, enabling the scalable development and operation of AI solutions.
AI Governance	The framework of policies, processes, and standards put in place to oversee the development and use of AI systems, ensuring they are safe, ethical, and compliant with regulations. Effective AI governance addresses issues like fairness, transparency, accountability, and risk management in AI initiatives.
CI/CD (Continuous Integration/Continuous Delivery)	A DevOps practice that automates the building, testing, and deployment of software, enabling frequent and reliable releases. Continuous Integration (CI) merges code changes regularly with automated tests, and Continuous Delivery (CD) ensures those changes can be deployed to production quickly through streamlined, repeatable processes.
DataOps	An agile, DevOps-inspired approach to data analytics and management that improves the *communication, integration, and automation of data flows* between data managers and data consumers across an organization. DataOps aims to streamline data pipeline development and maintenance, ensuring high data quality and faster delivery of insights aligned with business needs.
DevOps	A set of practices that combine software development (Dev) and IT operations (Ops) to shorten the systems development life cycle while delivering features, fixes, and updates frequently in close alignment with business objectives. DevOps emphasizes automation, shared ownership, and rapid feedback, reducing the time between code changes and deployment to production **without** sacrificing quality.
Feature Store	A centralized data platform that makes it easy to **build, manage, and serve** machine learning features for model training and inference. A feature store ensures that engineered features (predictor variables) are consistent and reusable across different ML projects, providing an offline store for batch data and an online store for real-time feature lookup.
FinOps	A cultural practice (from *"Finance + DevOps"*) for cloud financial management that promotes collaboration between engineering, finance, and business teams to optimize cloud costs and value. FinOps establishes shared accountability for cloud usage and spending, enabling data-driven decision making to balance speed, cost, and quality in cloud services.
Generative AI	A class of artificial intelligence techniques (often using advanced **generative models**) that learn patterns from existing data and produce new content such as text, images, or videos. Generative AI systems (e.g. Large Language Models or GANs) can create novel outputs resembling the data they were trained on, enabling applications like content creation, image generation, and conversational agents.
Human-in-the-loop (HITL)	A design approach for AI/ML systems that integrates human expertise into the model's lifecycle (training, validation, or operation) to improve outcomes.

	In human-in-the-loop setups, people provide feedback, annotations, or oversight – for example, reviewing model decisions or labeling data – thereby enhancing the accuracy, fairness, and safety of AI by leveraging human judgment where automation alone may fall short.
Inference Engine	The component of an intelligent system that applies logical rules or learned models to a knowledge base (or input data) in order to deduce new information, reach conclusions, or generate predictions. In modern AI platforms, an inference engine often refers to the runtime system that hosts a trained model and computes its outputs (inferences) from new inputs, optimizing for performance and accuracy.
Large Language Model (LLM)	A type of machine learning model with a very large number of parameters, designed for natural language processing tasks such as language understanding and generation. LLMs are trained on massive text corpora and can produce human-like text, answer questions, or carry on conversations (e.g. GPT-4 or other transformer-based models), by predicting the next word in a sequence based on learned language patterns.
MLOps (Machine Learning Operations)	A paradigm and set of best practices at the intersection of machine learning, software DevOps, and data engineering, aimed at reliably and efficiently deploying and maintaining ML models in production. MLOps bridges the gap between model development and operational deployment by automating the ML lifecycle (from data preparation and model training to continuous integration/delivery, monitoring, and governance), ensuring models are robust, scalable, and aligned with business goals.
ModelOps	A discipline focused on the governance and end-to-end lifecycle management of operationalized AI and analytics models across an enterprise. ModelOps orchestrates the deployment of models into production and oversees their performance, updates, and compliance with both technical and business KPIs. In practice, it ensures that all models (ML or rule-based) are properly versioned, monitored for drift or degradation, retrained as needed, and subjected to consistent governance policies throughout their life in production.
Prompt Engineering	The process of crafting and structuring the input *prompt* (i.e. the text or query given to a generative AI model) in order to achieve the best possible result from that model. Prompt engineering may involve providing context, instructions, or examples in a prompt so that large language models or other generative AI systems produce outputs that are accurate, relevant, and aligned with the intended task or style.
Responsible AI	An approach to AI development and deployment that prioritizes safety, ethics, and trustworthiness. Responsible AI practices ensure that AI systems are fair (mitigating bias), transparent (explainable in their decisions), accountable (with human oversight), and compliant with laws and societal values – so that AI-driven outcomes align with ethical principles and do not cause unintended harm.
Stakeholder Management	The process of identifying, prioritizing, and engaging stakeholders – individuals or groups who can affect or are affected by a project – throughout the course of an initiative. Effective stakeholder management involves understanding stakeholders' needs and expectations, communicating with them regularly, and addressing their concerns to ensure support and alignment. This helps in securing buy-in and reducing resistance, thereby increasing the likelihood of project success.
Value Stream	In a lean or product development context, a value stream is the end-to-end sequence of activities required to transform an idea or request into a delivered product or service that provides value for the customer. Mapping out value streams (e.g. from initial concept, through development stages, to

	final delivery and support) helps organizations analyze and optimize the flow of value, by eliminating waste and bottlenecks and ensuring each step contributes to the final outcome.