

How to become a Staff AI Engineer

Chapter 13: Other AI Systems Beyond LLMs

Celso Sousa

LinkedIn: <https://www.linkedin.com/in/celso-sousa/?locale=en-US>

Portfolio: <https://celsosousa.com.br/project-portfolio/>

Ebook: <https://celsosousa.com.br/ebook-staff-ai-engineer/>

Chapter Objective

AI Engineering is not limited to LLMs, agents, or generative systems. Many of the applications with the greatest economic impact continue to be built with recommender systems, ranking, forecasting, Computer Vision, Graph Machine Learning, and Reinforcement Learning.

These system families solve structurally different problems. Recommendation must select a small number of relevant items from potentially massive universes. Forecasting must represent temporal dependencies and uncertainty. Computer Vision transforms visual information into representations and decisions. Graph Machine Learning exploits explicit relationships among entities. Reinforcement Learning seeks to optimize sequences of decisions in which current actions can modify future states and rewards.

For a Staff AI Engineer, understanding these differences is essential because one of the most important architectural decisions occurs before model selection: identifying which paradigm adequately represents the structure of the problem.

When studying this chapter, the objective is not to memorize specific algorithms, but to develop a mental model capable of selecting, combining, operating, and governing different classes of AI systems:

- **Understand recommendation as a retrieval, matching, and ranking problem:** Systems at scale typically use candidate generation, ranking, re-ranking, and feedback mechanisms;
- **Recognize that ranking optimizes order, not merely classification:** In many products, the position of results determines a large portion of the value perceived by the user;
- **Treat time as a structural part of the problem in forecasting:** Trend, seasonality, autocorrelation, forecast horizon, and regime changes affect training, validation, and operation;
- **Select the appropriate granularity in Computer Vision:** Classification, object detection, and segmentation provide different capabilities and also incur different labeling, compute, and infrastructure costs;
- **Use Graph ML only when relationships provide material signal:** The existence of connected entities does not automatically justify a graph-based architecture;
- **Reserve Reinforcement Learning for genuinely sequential problems:** RL introduces exploration, reward design, credit assignment, and risks that do not exist to the same extent in Supervised Learning;
- **Design observability, security, and governance according to the paradigm:** Feedback loops, data poisoning, temporal leakage, graph drift, and reward hacking require different controls;
- **Connect technical metrics to the system's economic mechanism:** NDCG, forecast error, detection quality, or cumulative reward only have value when they improve real decisions and KPIs.

The central principle is simple: **the class of an AI system must correspond to the actual structure of the problem; the most sophisticated, recent, or popular technology is not necessarily the most appropriate architecture.**

13.1. Recommender Systems and Ranking

Recommender systems seek to select potentially relevant items for a given user or context. They appear in e-commerce, streaming, social media, advertising, marketplaces, news, education, and enterprise applications.

At a small scale, recommendation may appear to be merely a problem of calculating similarity or predicting the probability of interaction. In production, however, the challenge is systemic. Catalogs may contain millions of products, videos, or documents, user behavior changes continuously, and the system itself influences the data that will be observed in the future.

For this reason, recommendation at scale should be treated as a compound AI system.

13.1.1. Recommendation as a Multi-Stage Architecture

The fundamental problem is to find a small number of relevant items within a potentially enormous universe. Running a sophisticated model over every candidate for every user would typically be infeasible in terms of cost and latency.

The architecture therefore tends to decompose the problem into stages:

- **Candidate generation maximizes initial coverage:** It reduces millions of items to hundreds or thousands of candidates with high Recall;
- **Ranking increases selection precision:** It uses more sophisticated features and models to order only the retrieved subset;
- **Re-ranking introduces additional policies:** It can incorporate diversity, availability, inventory, safety, fairness, or business objectives;
- **Fallback keeps the system operational:** Popularity, rules, or contextual recommendations can be used when personalization does not have sufficient signal;
- **Feedback closes the learning loop:** Clicks, purchases, views, ratings, and other events feed future system updates.

This decomposition separates two distinct needs: recall-oriented retrieval and precision-oriented ranking.

The most important architectural point is that errors in early stages constrain everything that follows. If an excellent item does not enter the candidate set, even the most sophisticated ranker available cannot recover it.

13.1.2. Collaborative Filtering, Content-Based, and Hybrid Systems

One of the most important sources of information for recommendation is the interaction history between users and items. Collaborative Filtering relies precisely on this collective behavior. The intuition is that users with similar interaction patterns may have similar preferences, or that items consumed by similar audiences may have an implicit relationship.

The main approaches have different characteristics:

- **User-based Collaborative Filtering exploits similarity among users:** Preferences from similar users can provide signals for new recommendations, but directly comparing large populations can become expensive;
- **Item-based Collaborative Filtering exploits co-occurrence among items:** Relationships such as "users who consumed X also consumed Y" can be precomputed and tend to provide good interpretability;
- **Matrix Factorization learns latent factors:** Users and items are represented in compact vector spaces in which compatibility can be computed efficiently;
- **Content-based recommendation uses explicit characteristics:** Category, description, brand, price, topics, and embeddings can represent the content of each item;
- **Hybrid recommendation combines multiple signal sources:** Interaction embeddings, textual embeddings, metadata, popularity, and context can be used simultaneously.

Each approach solves part of the problem and introduces specific limitations.

Collaborative Filtering can discover implicit patterns that are not described in the metadata, but it depends on historical interactions. Content-based methods can operate more effectively with new items, provided their attributes are available, but they can produce recommendations that are excessively similar to already observed behavior.

Hybrid systems reduce this dependence on a single source of evidence. In production, this property also improves robustness and creates additional fallback options when a particular signal disappears or degrades.

The Staff-level perspective is to treat recommendation as a composition of signals rather than as the isolated selection of a single algorithm.

13.1.3. Candidate Generation, Ranking, and Re-Ranking

Candidate generation exists to rapidly reduce the search space. Its primary objective is not to produce the final ordering, but to ensure that potentially relevant items remain available for subsequent stages.

Different mechanisms can participate simultaneously:

- **Popularity provides robust and inexpensive candidates:** It is particularly useful as a fallback or when little historical data exists;
- **Collaborative signals exploit collective behavior:** They can retrieve items related to the user's consumption pattern;
- **Embeddings map users, queries, and items into vector spaces:** This enables Approximate Nearest Neighbor techniques for efficient retrieval;
- **Business rules add deterministic constraints:** Availability, region, eligibility, or other rules can eliminate invalid candidates;
- **Multiple retrievers can increase coverage:** Different candidate-generation sources can be combined before ranking.

The size of the candidate set creates a direct trade-off:

- **More candidates tend to increase Recall:** Good items are more likely to reach the ranker;
- **More candidates increase downstream cost:** Ranking and re-ranking must process more examples;
- **Fewer candidates reduce latency and compute:** However, they increase the risk of eliminating relevant items too early.

After this stage, the ranker can use more expensive signals, such as detailed user history, item characteristics, context, price, time, and device.

The objective must also be selected carefully. A model can optimize clicks, conversion, watch time, or revenue, but the selected metric influences the product's emergent behavior.

Maximizing CTR in isolation can favor clickbait. Maximizing watch time without additional controls can prioritize patterns that increase immediate consumption while harming long-term objectives.

For this reason, mature systems distinguish three classes of objectives:

- **Primary objective guides the main model:** It represents the behavior the system is intended to optimize;
- **Guardrails constrain undesirable effects:** Safety, diversity, fairness, or business limits can prevent degenerate solutions;
- **Long-term metrics evaluate ecosystem health:** Retention, lifetime value, coverage, or satisfaction can reveal effects that are invisible in immediate metrics.

The ordering produced by the ranker also does not have to be the ordering displayed to the user. A re-ranking policy layer can apply rules and constraints after scoring.

This separation improves governance and experimentability. The model estimates relevance. The policy layer controls the product's final behavior.

13.1.4. Cold Start, Exploration, and Feedback Loops

Personalization depends on evidence. When a new user, item, or platform has little historical data, the cold-start problem emerges.

The architecture must explicitly handle this absence of information:

- **New users can receive popular items:** This strategy provides a robust baseline while interactions are not yet available;
- **Contextual signals can provide initial personalization:** Device information, session context, or other permitted information can reduce uncertainty;
- **Onboarding can explicitly collect preferences:** In some products, a small number of initial choices already provides sufficient signal;
- **New items can use content-based features:** Metadata and embeddings enable recommendation before interactions accumulate;
- **Fallbacks must remain available:** Personalization should not become a single point of failure.

In addition to data scarcity, recommendation faces the exploration-versus-exploitation problem.

Exploitation presents items that the system already believes to be good. This tends to maximize immediate reward. Exploration presents less-known alternatives to obtain information about users, items, or opportunities.

Both extremes are problematic:

- **Excessive exploitation reduces discovery:** The system can reinforce existing preferences and create filter bubbles;
- **Excessive exploration degrades the experience:** Users begin receiving low-relevance alternatives;

- **Controlled exploration increases knowledge:** Small levels of experimentation can uncover new items and changes in preference;
- **Action risk must constrain experimentation:** Not every domain allows users to be exposed to unknown alternatives.

Recommendation also has an important characteristic: **the system changes the data that it will later use for learning.**

Displayed items receive more opportunities for interaction. Items that are not displayed receive less data. This creates feedback loops.

Popularity bias is a classic example. Popular items receive greater exposure, generate more interactions, and consequently begin to appear even more relevant.

Mitigations can involve exploration, normalization, diversity constraints, and exposure-aware evaluation.

The architectural consequence is profound: **recommender systems do not merely model an environment, they participate in creating the very environment they subsequently observe.**

13.1.5. Learning to Rank and Evaluation Metrics

Learning to Rank learns functions designed to order items according to relevance. This problem appears in search, recommendation, advertising, marketplaces, and also in retrieval pipelines for RAG.

The difference from traditional classification is that the final objective is not merely to predict each item correctly in isolation. Value depends on its relative position within the list.

There are three main conceptual families:

- **Pointwise approaches treat items individually:** The model can predict a relevance score or probability of click and subsequently order the results;
- **Pairwise approaches learn relative preferences:** Training attempts to determine whether one item should appear above another;
- **Listwise approaches treat the list as the optimization unit:** Training seeks to directly approximate the desired behavior of the complete ranking.

These strategies introduce different trade-offs.

Pointwise methods are simple, scalable, and reuse traditional regression or classification infrastructure, but they do not directly optimize relationships among positions. Pairwise methods bring training closer to relative preference, but can generate a very large number of pairs. Listwise methods better represent the product observed by the user, but add training complexity.

In many systems, the correct decision is not to use the theoretically most sophisticated method, but to determine whether the ranking improvement justifies the additional cost.

Evaluation must also represent how results are consumed:

- **NDCG combines relevance and position:** Highly relevant results at the top receive greater weight than the same results in lower positions;
- **MRR emphasizes the first relevant result:** It is useful when the user typically needs to quickly find a single suitable answer;

- **Candidate Recall measures the retrieval stage:** It indicates whether relevant items are reaching the ranker;
- **Conversion or task outcomes measure downstream effects:** A good list is useful only when it contributes to the actual objective;
- **Coverage and diversity measure systemic effects:** They can reveal excessive concentration even when click metrics remain strong.

An important conclusion is that evaluation must be decomposed by stage. Measuring only the final outcome makes it difficult to diagnose whether degradation occurred in candidate generation, ranking, policy, or user behavior itself.

13.2. Time Series and Forecasting

Time Series models data in which temporal ordering is part of the information. Sales, demand, prices, traffic, energy consumption, capacity, and incidents are examples in which observations cannot be treated as independent samples.

The past influences the future and, consequently, the temporal dimension changes feature engineering, validation, deployment, monitoring, and interpretation.

Forecasting should also not be treated merely as a problem of minimizing statistical error. The final objective is to anticipate a decision.

13.2.1. Temporal Structure: Trend, Seasonality, and Autocorrelation

A time series often combines different mechanisms. Understanding this decomposition helps both modeling and failure interpretation.

The most important components include:

- **Trend represents long-term structural movements:** Persistent growth or decline can make models that assume stability systematically biased;
- **Seasonality represents recurring patterns:** Time of day, day of week, month, or other cycles can produce predictable behavior;
- **Autocorrelation represents dependence between current and previous values:** Lagged values and moving averages can capture part of this signal;
- **Noise represents unexplained variation:** Not every observed change should be treated as a new pattern;
- **External events can temporarily alter the series:** Promotions, incidents, or operational changes may require additional context.

Seasonality has consequences beyond business forecasting. Predictable peaks can guide autoscaling, capacity reservation, and staffing.

This demonstrates an important characteristic of AI Engineering: **the same model can simultaneously contribute to business and operational decisions.**

Autocorrelation, on the other hand, introduces one of the greatest risks in forecasting: leakage. Temporal features must be calculated using only information that would have been available at the actual time of the decision. An apparently simple moving average can invalidate the evaluation if it accidentally incorporates future observations.

13.2.2. Forecast Horizon and Probabilistic Forecasting

Before selecting a model, it is necessary to define how far into the future the decision needs to look. Forecasting five minutes, one week, or one year ahead represents different problems. As the forecast horizon increases, the available information about the future decreases and uncertainty tends to increase.

The Staff-level question should not begin with "which model should we use?" but with: **which decision depends on this forecast, and how far in advance must that decision be made?**

This distinction affects architecture and evaluation:

- **Short horizons can strongly leverage recent signals:** Autocorrelation and current events can be highly informative;
- **Long horizons increase dependence on structural factors:** Trend and external changes begin to dominate part of the uncertainty;
- **Different horizons may require different models or features:** A single pipeline does not necessarily optimize every interval;
- **The value of the forecast depends on the time available to act:** An excellent forecast delivered too late may have little operational value.

In addition to the horizon, it is necessary to determine how uncertainty should be represented. Point forecasting provides a single expected value. This output is simple, but it may be insufficient when errors have asymmetric consequences.

Probabilistic forecasting provides information about the distribution of possible outcomes:

- **A central scenario communicates the primary expectation:** It may be sufficient for low-criticality planning;
- **Intervals represent uncertainty:** They make it possible to distinguish stable situations from poorly predictable scenarios;
- **Percentiles can guide conservative decisions:** Capacity planning can use a high percentile when insufficient capacity is much more expensive than idle capacity;
- **Distributions make it possible to incorporate risk asymmetry:** Inventory, infrastructure, and financial risk may require different decisions for the same expected mean.

In many cases, therefore, the forecast distribution has more business value than a slightly more accurate point estimate.

13.2.3. Evaluation, Leakage, and Regime Change

Temporal data generally should not be randomly split between training and validation. The objective of evaluation is to reproduce the asymmetry that exists in production: **the model knows the past and must predict the future.**

This principle fundamentally changes experimental design:

- **Training must use only information that was previously available:** The pipeline must preserve point-in-time correctness;
- **Validation must simulate future decisions:** A random split can allow later patterns to contaminate training;
- **Rolling evaluation tests multiple windows:** This makes it possible to measure stability across different periods and regimes;

- **The metric must reflect the decision:** Percentage errors may work well in some domains and be inappropriate when values near zero are frequent;
- **Directional bias must be monitored:** Consistently underestimating demand can be operationally worse than symmetric errors of the same magnitude.

Another problem appears when the structure of the series itself changes. Economic crises, product launches, pricing changes, regulatory changes, or extraordinary events can transform the relationship between past and future.

This phenomenon characterizes regime change. When this occurs, a complex model trained on years of historical data may perform worse than a simple approach capable of adapting quickly. System maturity therefore depends not only on producing forecasts, but also on detecting when historical assumptions are no longer valid.

13.3. Computer Vision

Computer Vision transforms images or videos into representations useful for decision-making. Applications appear in industry, agriculture, medicine, retail, security, vehicles, and document processing.

A fundamental decision occurs before architecture selection: **what level of visual information does the business actually need?**

Classification, object detection, and segmentation answer different questions and require different amounts of data, labeling, compute, and infrastructure.

13.3.1. Classification, Detection, and Segmentation

Image classification assigns one or more labels to the entire image. It is appropriate when the system needs to determine what is present, but not where it is located.

Object detection adds localization. The model must simultaneously identify classes and determine the image regions associated with the objects.

Segmentation increases granularity even further and classifies regions or pixels.

These capabilities form a progression of complexity:

- **Classification determines whether a given property is present:** The pipeline tends to be simpler and annotation is typically less expensive;
- **Detection identifies objects and their approximate locations:** Bounding boxes add spatial information and also increase labeling cost;
- **Segmentation provides detailed localization:** Pixel-level outputs may be necessary when the shape or extent of a region matters;
- **Greater granularity increases output volume:** The downstream system must store, process, and interpret more complex results;
- **Greater granularity generally requires more compute and data:** The technical improvement must produce sufficient operational impact to justify the cost.

A correct detection with inadequate localization may be useless in industrial inspection. Likewise, extremely accurate segmentation can represent overengineering when a bounding box is already sufficient to support the required decision.

The Staff-level question should be: **what is the minimum visual granularity that solves the business problem?**

13.3.2. Visual Architectures and Representation Capacity

Vision Transformers apply mechanisms derived from Transformers to images, representing visual regions as tokens processed through attention. They provide important properties:

- **High representation capacity enables scaling with data:** Larger models can learn complex visual patterns;
- **Transfer learning reduces the need for end-to-end training:** Visual foundation models can be adapted to different tasks;
- **Representations compatible with multimodal architectures facilitate composition:** Text and images can be processed within similar ecosystems.

However, this does not eliminate convolutional architectures. CNNs remain competitive across many workloads, particularly when datasets are smaller, compute is constrained, or edge deployment is highly important.

Therefore, the decision should not assume that a newer architecture is automatically superior:

- **Larger models can provide greater capacity:** However, they increase memory requirements, cost, and latency;
- **Smaller architectures can reduce operational cost:** This may be decisive when inference occurs at high volume;
- **Transfer learning can reduce development effort:** But the foundation model being used must still be validated within the specific domain;
- **Edge constraints can reverse architectural preferences:** A model that performs slightly worse on a benchmark may produce a much better solution when it can run locally within hardware constraints.

AI Engineering must optimize the complete system, not merely the model's offline metric.

13.3.3. Data Engineering and Annotation Quality

In Computer Vision, much of the system's quality is determined before training.

Lighting, camera, resolution, angle, background, and the capture process directly alter the pixel distribution. An apparently minor hardware change can create enough distribution shift to significantly degrade the model. The capture pipeline must therefore be considered part of the ML system.

Several aspects require explicit treatment:

- **Sensors must have known characteristics:** Camera changes can alter the input distribution;
- **Lighting should be controlled whenever possible:** Unnecessary variability increases modeling difficulty;
- **Resolution must be sufficient for the task:** Information absent from the sensor cannot be recovered by the model;
- **Environmental conditions must appear in the dataset:** Models trained only under ideal conditions may fail in production;
- **Data distribution must be observable:** The team must be able to detect when the capture environment has changed.

Label quality also establishes an upper bound on observable performance. Detection and segmentation can exhibit inconsistent boxes, omitted objects, ambiguous classes, and disagreement among annotators. If experts

regularly disagree about the correct answer, requiring near-perfect model accuracy may be a target that is inconsistent with the ground truth itself.

Labeling guidelines must therefore be documented, versioned, and audited. Improving the capture process or annotation consistency can produce greater gains than replacing the architecture with a larger model.

13.3.4. Edge Computer Vision

Some applications need to execute inference close to the sensor. Factories, drones, mobile devices, and agricultural systems are examples in which sending every image to centralized infrastructure may be inappropriate.

Edge deployment provides important benefits:

- **Lower latency enables fast local decisions:** The system does not depend on a round trip to a remote service;
- **Lower network dependence increases availability:** Connectivity failures do not necessarily interrupt the primary function;
- **Greater privacy can reduce data exposure:** Sensitive images can remain on the device;
- **Lower bandwidth reduces communication cost:** Only relevant results or events need to be transmitted.

These advantages create new constraints:

- **Memory is typically limited:** Large models may not fit on the device;
- **Compute is constrained:** Throughput and latency become highly dependent on the architecture;
- **Energy can be a critical resource:** Inference must compete with other device functions;
- **Updates are more complex:** Distributing new versions to thousands of devices requires additional governance.

Techniques such as quantization, pruning, and smaller architectures become particularly important.

Edge AI is a clear example of AI Engineering as optimization under constraints: **the best architecture is the one that provides sufficient quality within the actual budget for hardware, energy, connectivity, and operations.**

13.4. Graph Machine Learning

Graph Machine Learning is appropriate for problems in which relationships among entities provide material information.

A graph represents nodes, edges, and attributes. People and relationships, accounts and transfers, products and purchases, papers and citations, or systems and dependencies are natural examples.

Tabular Machine Learning tends to represent each row as a relatively independent observation. Graph ML adds relational context. This capability can be extremely valuable, but it carries significant data, infrastructure, and operational costs.

13.4.1. Relational Representations

An entity can be characterized not only by its own features, but also by its position within the network and by the entities to which it is connected.

Fraud provides a useful intuition. An account may appear normal when analyzed in isolation. However, its connections may reveal proximity to several accounts associated with fraudulent behavior.

Graph representations can capture signals such as:

- **Topology describes the global or local structure:** Certain positions in the graph may have operational meaning;
- **Neighborhood represents relational context:** Nearby entities can provide additional evidence;
- **Relationships carry their own semantics:** The type and direction of edges can be as important as the nodes;
- **Structural patterns can reveal information absent from tabular features:** Communities or connection sequences can generate new signals.

However, the mere fact that data can be represented as a graph does not justify Graph ML.

The correct question is whether the relational structure materially improves the decision. Otherwise, a tabular representation or a simpler model may be less expensive and operationally superior.

13.4.2. Message Passing and Graph Neural Networks

Graph Neural Networks frequently use message passing. Each node updates its representation by combining its own information with information originating from neighboring nodes. After multiple steps, the embedding can incorporate progressively more distant context.

This operation has a natural trade-off:

- **More hops increase context:** Nodes begin incorporating information from larger regions of the graph;
- **More hops increase compute:** Neighborhoods can grow rapidly;
- **More context can introduce noise:** Not every distant relationship is useful;
- **Large graphs require sampling:** Processing the full neighborhood may be infeasible at scale.

GNNs can be used for different tasks:

- **Node classification assigns properties to entities:** An account can be classified as fraudulent or non-fraudulent;
- **Link prediction estimates potential relationships:** It can be used for recommendation or connection discovery;
- **Graph classification produces a decision about the entire structure:** A molecule can be classified according to a particular property.

The architecture must be selected according to the task and the observed relational gain, not merely because a GNN library is available.

13.4.3. Graph Embeddings and Composition with Other Models

Graph embeddings convert nodes or relational structures into vector representations. These vectors can subsequently be used for similarity, retrieval, recommendation, or downstream models.

This composition is important because Graph ML does not need to dominate the entire architecture. A user embedding derived from a social structure, for example, can be used as an additional feature in a Gradient Boosting model.

This strategy creates several advantages:

- **The relational component remains specialized:** The graph pipeline solves only the part of the problem in which relationships are relevant;
- **Downstream models can remain simple:** Traditional infrastructure can continue to be used;
- **Graph features can be precomputed:** This reduces cost on low-latency paths;
- **The architecture becomes more modular:** The system can easily compare versions with and without relational signal.

In many products, this composition is more practical than an end-to-end GNN architecture.

This pattern reinforces a general AI Engineering principle: **different paradigms can operate as components of a larger system without requiring the entire platform to inherit the complexity of its most sophisticated component.**

13.4.4. Temporal Leakage and Scalability

Graphs introduce their own forms of leakage. If a historical decision is evaluated using edges that were only created later, the model receives information that did not exist at the actual time of prediction.

Imagine an account evaluated in January. If the graph used during training includes relationships created in March, the evaluation contains temporal leakage. Point-in-time correctness is therefore as important in Graph ML as it is in forecasting.

Scalability represents another challenge. Enterprise graphs may contain billions of nodes and edges.

Several strategies help make the problem operationally feasible:

- **Neighborhood sampling limits the number of neighbors processed:** It reduces cost but loses part of the structure;
- **Partitioning distributes the graph:** It facilitates processing over very large volumes;
- **Precomputed embeddings shift compute offline:** Online serving can use representations that have already been calculated;
- **Distributed training enables learning at scale:** However, it adds infrastructure and synchronization complexity.

The recurring trade-off is quality versus scale. More structural context can improve representation, but the architecture must pay for that context in storage, bandwidth, memory, and compute.

13.5. Reinforcement Learning

Reinforcement Learning learns behavior through interaction with an environment. Unlike Supervised Learning, in which training typically seeks to learn a relationship between inputs and targets, RL must select actions and observe rewards. Current actions can also alter future states and modify subsequent opportunities.

This property makes RL suitable for sequential decision-making problems, but it introduces a new class of operational complexity. State representation, action space, reward design, exploration, policy evaluation, and safety become fundamental parts of the architecture.

13.5.1. State, Action, Reward, Policy, and Value

The fundamental components of RL define the problem the system will actually learn:

- **State represents the information available about the current situation:** It must contain sufficient evidence to support useful decisions;
- **Action represents the available decisions:** It can exist in discrete or continuous spaces;
- **Reward provides the optimization signal:** The agent learns to favor trajectories that increase cumulative reward;
- **Policy defines operational behavior:** It determines which action to select for a given state;
- **Value estimates future consequences:** It enables reasoning about benefits that do not appear immediately.

Each component has engineering consequences. The actual state may not be fully observable. In this case, the agent operates from incomplete observations and must infer missing information.

The action space also functions as a security boundary. An agent should not have more capabilities than its function requires. Restricting possible actions is often safer than allowing dangerous actions and relying exclusively on learned behavior to avoid them.

Reward design may be the most delicate component. The system optimizes what is measured, not necessarily what the organization intended to measure.

If reward represents usage time, the system may discover strategies that increase engagement while degrading the long-term experience. For this reason, reward engineering should be treated as incentive design.

The resulting policy represents operational behavior and must be versioned, evaluated, progressively deployed, and support rollback in the same way as other production components.

13.5.2. Exploration, Exploitation, and Multi-Armed Bandits

To discover better actions, the system must experiment with alternatives. This process is exploration. Exploitation uses what the current policy already considers the best option.

The trade-off appears in recommendation, advertising, pricing, and resource allocation:

- **More exploitation improves expected immediate outcomes:** The system favors alternatives that already have positive evidence;
- **More exploration produces new information:** The platform learns about less-known options;
- **Exploration can reduce immediate performance:** Some users receive less-validated actions;
- **Action risk limits how much experimentation is acceptable:** Critical environments cannot tolerate unrestricted exploration.

In production, exploration can mean executing poorly understood actions on real users or processes. For this reason, mechanisms such as bounded exploration, sandboxing, simulation, offline learning, and human approval may be necessary.

Not every exploration problem requires full RL.

Multi-Armed Bandits provide a simpler formulation in which the system chooses among alternatives and observes reward without necessarily modeling complex consequences across multiple future states. They may be appropriate for content selection, recommendation, or experiment allocation.

The Staff-level principle is to use the smallest class of method that adequately represents the problem. If the decision consists only of choosing among alternatives and observing a relatively immediate return, full RL may introduce complexity without proportional benefit.

13.5.3. Offline Reinforcement Learning and Simulation

Learning directly in production can be expensive or dangerous. Offline Reinforcement Learning attempts to use historical data to learn policies without requiring unrestricted exploration in the real environment.

This approach reduces part of the risk, but introduces another limitation: **the dataset represents only regions of the action space that were visited by historical policies.**

- **Historical data avoids part of online experimentation:** This can be essential in high-cost environments;
- **Dataset coverage limits generalization:** The policy may produce unreliable estimates for rarely observed situations;
- **The data-collection policy influences the available data:** Historical data does not necessarily represent every potentially good action;
- **Evaluation remains difficult:** Better offline performance does not guarantee superior behavior after deployment.

Simulation provides another alternative. A simulator allows policies to be trained and tested without immediately exposing the real world to the consequences of their actions.

Robotics, operations, games, and planning can benefit from this strategy. However, a sim-to-real gap exists. A simulator never represents reality perfectly.

An excellent policy in simulation may exploit artificial properties of the environment and fail when those properties do not exist in production. For this reason, fidelity, domain randomization, and progressive deployment remain important.

Simulation reduces development risk. It does not eliminate the need for validation in the real environment.

13.5.4. Reward Hacking, Credit Assignment, and Safety

Reward hacking occurs when the agent discovers a strategy capable of increasing the metric without fulfilling the original intent.

Imagine a system rewarded according to the number of completed tasks. If tasks can be marked as completed without being properly resolved, reward increases while real value decreases. This divergence demonstrates why a single metric is rarely sufficient for complex behaviors.

High-stakes rewards may require:

- **Multiple signals to reduce fragile proxies:** Different indicators make degenerate optimizations more difficult;
- **Deterministic guardrails to prevent prohibited actions:** Not every rule needs to be learned;
- **Evaluation independent of the training reward:** The same signal should not be the only evidence of success;
- **Trajectory auditing:** It is necessary to understand how the policy reached a particular outcome;
- **External business and safety metrics:** Reward can increase while the actual system becomes worse.

Another challenge is long-term credit assignment. In long tasks, the final reward may occur long after the actions that contributed to the outcome. When a sequence contains dozens of decisions, identifying the step responsible for success or failure becomes difficult. This increases the importance of trajectory-level observability.

RL also has specific security and safety risks:

- **Unsafe exploration can execute harmful actions:** Trial-and-error learning is not acceptable for every capability;
- **Policies can exploit environment bugs:** The system may discover unanticipated behaviors;
- **Distribution shift can invalidate an apparently safe policy:** Unseen states can produce inappropriate actions;
- **Reward hacking can conceal poor behavior:** Internal metrics may remain positive;
- **Excessive action spaces increase blast radius:** Permissions must follow the principle of least privilege.

External controls such as action constraints, safety policies, runtime validation, human approval, and kill switches must be considered in critical systems. The fundamental rule is not to depend exclusively on the reward function to enforce safety.

13.5.5. Reinforcement Learning vs. Supervised Learning

RL should be used when there is meaningful interaction between current actions and future consequences. When a reliable dataset of input-output pairs already adequately describes the desired behavior, Supervised Learning tends to be simpler.

RL introduces additional responsibilities:

- **Exploration must be managed:** The system may need to execute suboptimal actions in order to learn;
- **Reward design must represent intent:** Poor proxies can generate unexpected behavior;
- **Simulation or experimentation environments may be necessary:** This increases development cost;
- **Training can be less stable:** Policy changes alter the observed distribution;
- **Evaluation becomes more difficult:** Outcomes depend on trajectories and long-term effects;
- **Safety requires additional controls:** The learning process itself can modify operational behavior.

For this reason, a sequence of decisions alone is not sufficient to justify RL. If a supervised policy or deterministic mechanism adequately solves the problem, it may provide a more economical, predictable, and safe solution.

13.6. Production Engineering Across Different Paradigms

The families presented so far have different learning mechanisms, but they converge when they need to operate as real systems. Retrieval, ranking, observability, security, governance, deployment, and evaluation become cross-cutting concerns. In addition, mature products frequently combine multiple paradigms within the same architecture.

13.6.1. Recommendation, Search, RAG, and Multimodal Composability

Recommendation, search, and RAG share an important architectural structure. All of them can begin with a very large universe, execute candidate generation, apply ranking, perform re-ranking, and submit the results to a policy layer.

What changes is the objective:

- **Recommendation seeks to maximize preference or utility:** The primary context typically involves the user and behavior;
- **Search seeks to maximize relevance to a query:** The explicit intent of the query occupies a central role;
- **RAG seeks to retrieve useful evidence for generation:** The value of a document also depends on how it will be used by the generative model.

This similarity allows knowledge from Information Retrieval, Ranking, and Evaluation to be reused across different classes of applications. These families can also be combined.

A recommender can use collaborative signals, text embeddings, image embeddings, graph embeddings, and contextual features within the same ranker. An industrial system can combine Computer Vision to detect defects, Time Series to analyze sensors, and Graph ML to represent dependencies among equipment. These systems make it clear why AI Engineering must be understood as the engineering of compound AI systems.

The objective is not to select a winning technology. It is to assign each part of the problem to the most appropriate mechanism and define clear interfaces among those mechanisms.

13.6.2. Paradigm-Specific Observability

Monitoring only CPU, memory, latency, and errors is not sufficient for AI systems. Each paradigm has its own functional signals and specific degradation modes.

In recommendation, important metrics include:

- **Candidate Recall reveals retrieval quality:** Declines can prevent relevant items from reaching ranking;
- **Ranking metrics measure ordering:** NDCG and other metrics help identify regressions in the list;
- **Click and conversion measure downstream interaction:** They show whether technical relevance translates into behavior;
- **Diversity measures variety:** It can detect excessive concentration;
- **Coverage shows how much of the catalog receives exposure:** A system can achieve strong CTR while virtually ignoring the long tail.

In forecasting, monitoring must observe a different set of signals:

- **Prediction error measures realized quality:** It should be analyzed over time;
- **Calibration evaluates probabilistic intervals:** A useful confidence interval must accurately represent its empirical coverage frequency;
- **Drift detects distribution changes:** Features and outcomes can diverge from historical patterns;
- **Regime changes indicate structural change:** The model may need to be reevaluated even if no code has changed;
- **Systematic bias reveals the direction of error:** Continuously underestimating demand can be far more serious than an apparently acceptable average error.

In Computer Vision, the data source itself must be monitored:

- **Lighting can alter the distribution:** Environmental changes can cause regressions;
- **The camera or sensor can change:** Different hardware can invalidate training assumptions;
- **Resolution and blur affect available information:** Capture problems can appear to be model problems;
- **Input distribution must be compared:** The system should distinguish model drift from sensor drift.

Graph ML requires structural observability:

- **Node distribution can change:** New entity types alter the domain;
- **Edge volume can increase or decrease abruptly:** Process changes can alter relationships;
- **Degree distribution reveals graph reorganization:** Unexpected structures can affect message passing;
- **Connected components help detect systemic transformations:** Model behavior can change even without changes to model weights.

In Reinforcement Learning, the behavior of the policy itself must be observed:

- **Reward shows the optimized signal:** But it cannot be the only metric;
- **Action distribution reveals behavioral changes:** A policy may begin selecting certain actions at an unexpected frequency;
- **State distribution identifies new environments:** Distribution shift can invalidate previous evaluations;
- **Safety violations measure prohibited effects:** They must exist independently of reward;
- **Exploration rate shows how much experimental risk is being introduced:** Changes must be operationally visible.

The common principle is to decompose observability according to the causal structure of the system. The team must be able to explain not only that quality declined, but which mechanism changed.

13.6.3. Cybersecurity and Data Poisoning

AI Security is not limited to prompt injection. Classical systems can also be manipulated through data, feedback, the environment, or relational structure.

Attack surfaces vary according to the paradigm:

- **Recommendation can suffer from feedback manipulation:** Bots can generate artificial clicks, ratings, follows, or purchases to increase the exposure of a particular item;
- **Ranking can suffer from feature manipulation:** External actors can adapt content to exploit known signals used by the ranker;
- **Computer Vision can receive adversarial inputs or physical manipulation:** Environmental changes can induce incorrect classifications;
- **Graph ML can suffer from the artificial insertion of nodes and edges:** The structure created by an attacker can modify representations;
- **Reinforcement Learning can have its environment or reward manipulated:** The agent can learn behavior from adversarial signals.

Recommendation is particularly sensitive to poisoning because user feedback frequently serves as training data. Mitigations can combine anomaly detection, identity signals, trusted interactions, weighting, and traditional fraud-detection mechanisms.

The architectural rule is not to assume that behavioral data represents reliable ground truth simply because it was produced by real users or appears to represent legitimate events.

13.6.4. Governance and Auditability

Each paradigm also creates specific governance requirements. Recommendation can determine which content, suppliers, or products receive exposure. For this reason, model scores and business policies must be separately observable.

The organization may need to document:

- **Model objective:** Which behavior is being optimized;
- **Business rules:** Which decisions are applied in addition to the model;
- **Fairness constraints:** Which restrictions influence distribution;
- **Prohibited content:** Which results cannot be displayed;
- **Experimentation policy:** How new strategies are tested and approved.

Forecasting requires temporal traceability:

- **Model version records which artifact produced the forecast:** This makes it possible to reconstruct decisions;
- **Prediction timestamp establishes when the forecast was available:** This is essential for point-in-time correctness;
- **Forecast horizon documents which future period was being predicted:** Comparisons must respect equivalent horizons;
- **Data version records which information was available:** Retrospective evaluation cannot incorporate later data;
- **Uncertainty must be preserved:** If it influenced the decision, it must remain auditable.

Computer Vision can process faces, documents, or private environments. Purpose, retention, access control, privacy, and human oversight are part of the architecture, not merely a subsequent review.

Graph ML introduces another risk: **derived relationships can reveal information that was not explicit in the original datasets.**

By connecting different sources, the system can infer personal, financial, or organizational relationships. Access control and purpose limitation must consider both the nodes and the information produced by the graph structure. Mature governance must therefore cover data, models, policies, and derived inferences.

13.6.5. Business Impact

These AI families are among the most economically mature applications because they have relatively clear value-creation mechanisms. Recommendation can influence different dimensions:

- **Conversion increases when relevant items reach the right users:** This effect can directly impact revenue;
- **Retention can improve through personalized experiences:** Value emerges over time;
- **Cross-sell connects complementary products:** Recommendation can increase average transaction value or the number of items per transaction;
- **Discovery distributes exposure across the long tail:** This can increase catalog utilization;
- **Advertising improves matching among inventory, users, and context:** Relevance can increase the value of advertising inventory.

However, maximizing immediate revenue can harm retention. KPIs must balance the short and long term.

Forecasting creates value when it improves future decisions:

- **Inventory can be sized with less waste:** Better forecasts reduce excess inventory and stockouts;
- **Staffing can follow expected demand:** Human capacity can be distributed more efficiently;
- **Supply chains can react earlier:** Decisions gain greater lead time;
- **Infrastructure capacity can be planned:** Forecasts can guide reservations and autoscaling;
- **Cash flow can be managed with greater predictability:** Anticipation improves financial planning.

A statistical metric only has value if it improves these decisions.

Computer Vision can reduce manual inspection, decrease losses, increase productivity, and improve quality control.

Detection accuracy is a technical metric. Prevented defects, inspection time, loss reduction, and operational cost are business metrics.

Graph ML creates value when relationships invisible in traditional representations improve decisions in fraud, recommendation, cybersecurity, supply chain, or knowledge discovery. However, entity resolution and graph pipelines can be expensive. The relational signal must justify the additional infrastructure.

Reinforcement Learning can generate significant value in resource allocation, recommendation, control systems, and dynamic optimization when sequential decisions are central. At the same time, RL may require simulators, experimentation infrastructure, safety controls, and advanced monitoring. Its ROI must justify this complexity.

13.6.6. Key Trade-Offs and Anti-Patterns

The families presented in this chapter have different decision surfaces, but several trade-offs appear repeatedly.

- **Candidate Recall vs. Ranking Cost:** More candidates increase the probability of including good items, but increase downstream processing;
- **Personalization vs. Diversity:** Intensive personalization can increase immediate relevance while reducing discovery;
- **Exploration vs. Immediate Reward:** Experimentation produces new information but can reduce short-term performance;
- **Point Forecast vs. Uncertainty:** A single forecast is simple, whereas distributions enable decisions that are better aligned with risk;
- **Forecast Horizon vs. Predictability:** The farther into the future, the greater the uncertainty tends to become;
- **Classification vs. Detection vs. Segmentation:** Greater visual detail generally requires more labeling, compute, and operational complexity;
- **CNN vs. Vision Transformer:** Different architectures balance capacity, cost, dataset characteristics, hardware, and transfer learning;
- **Tabular Features vs. Graph Structure:** Graph ML can capture important relationships while adding pipelines and infrastructure;
- **Full Graph Context vs. Sampling:** Larger neighborhoods provide additional information but increase cost;
- **Supervised Learning vs. Reinforcement Learning:** RL represents sequential decisions at the cost of exploration, reward design, and more difficult evaluation;
- **Offline Optimization vs. Online Learning:** Online adaptation enables faster learning but increases exposure to risk;
- **Model Objective vs. Ecosystem Health:** Maximizing clicks, reward, or revenue can produce systemic side effects.

These trade-offs help identify several anti-patterns:

- **Using an LLM for simple tabular ranking:** This introduces cost and latency when traditional models adequately solve the problem;
- **Applying a sophisticated ranker to poor candidates:** The best ranking model cannot recover items eliminated during retrieval;

- **Optimizing only CTR:** This can produce clickbait or behaviors that degrade long-term value;
- **Ignoring cold start:** The system fails precisely when users or items have the least evidence;
- **Failing to control feedback loops:** Popularity bias can reinforce itself indefinitely;
- **Using random splits for time series:** Evaluation can be contaminated by future information;
- **Producing deterministic forecasting when risk requires uncertainty:** A single number creates false precision;
- **Using segmentation when classification is sufficient:** Labeling and compute increase without corresponding returns;
- **Training vision models without controlling capture:** Camera or environment changes create an unstable distribution;
- **Using a GNN simply because relationships exist:** Complexity increases without guaranteed gains;
- **Building graphs on weak entity resolution:** Incorrect relationships begin to be learned as evidence;
- **Using RL when Supervised Learning solves the problem:** Exploration and reward design are introduced unnecessarily;
- **Defining a single reward for complex behavior:** The risk of reward hacking increases;
- **Allowing unrestricted exploration for critical actions:** The learning process begins to create operational risk;
- **Evaluating only the model score while ignoring the policy layer:** The behavior presented to the user can be completely different from the raw scoring.

Recognizing these patterns prevents technical sophistication from becoming complexity without return.

Chapter Summary

AI Engineering is much broader than LLM Engineering. Recommender systems, ranking, forecasting, Computer Vision, Graph Machine Learning, and Reinforcement Learning represent distinct classes of problems, each with its own mathematical, operational, and economic properties.

The central principles can be summarized as follows:

- **Recommender systems are typically multi-stage:** Candidate generation, ranking, and re-ranking solve different problems and must be evaluated separately;
- **Collaborative and content-based signals have complementary limitations:** Hybrid systems increase robustness and help address cold start;
- **Ranking must be measured according to position and consumption behavior:** NDCG, MRR, and candidate-retrieval metrics answer different questions;
- **Recommendation creates feedback loops:** The system influences the very data that it will subsequently use for learning;
- **Time Series requires point-in-time correctness:** Random splits and features containing future information can produce completely misleading results;
- **Forecast horizon must be determined by the decision:** The required lead time defines a substantial part of the architecture;
- **Probabilistic forecasting is essential when risk is asymmetric:** Distributions can be more useful than point predictions;
- **Computer Vision requires selecting the appropriate granularity:** Classification, detection, and segmentation have different capabilities and costs;
- **The capture pipeline is part of the visual system:** Changes in sensors or the environment may matter more than changes to the model;

- **Graph ML should be used when relationships carry real signal:** The existence of a graph is not sufficient justification;
- **GNNs expand context through message passing:** Sampling, compute, and noise must be balanced;
- **Graph embeddings enable composition:** Relational signal can be incorporated into traditional models without turning the entire architecture into a GNN;
- **Reinforcement Learning is appropriate for sequential decisions:** It should not replace Supervised Learning when a simpler formulation solves the problem;
- **Reward design defines incentives:** Reward hacking demonstrates that optimizing a metric does not guarantee optimization of the intended objective;
- **Exploration must be proportional to risk:** Critical environments require constraints, simulation, offline learning, or human oversight;
- **Observability must be paradigm-specific:** Candidate Recall, interval calibration, sensor drift, graph drift, and action distribution represent different problems;
- **AI Security extends beyond generative systems:** Data poisoning, adversarial inputs, graph manipulation, and reward manipulation must also be considered;
- **Governance must cover both models and policies:** The score produced by the model does not necessarily correspond to the system's final decision;
- **Technical metrics must ultimately connect to business metrics:** Better ranking, forecasting, or detection only has value when it improves real outcomes;
- **The most sophisticated technology is not automatically the best solution:** Simplicity, cost, latency, reliability, security, and operational capability remain fundamental criteria.

The key lesson is that a Staff AI Engineer must master a portfolio of paradigms and recognize the limitations of each one. The objective is not to adapt every problem to the most popular model, but to deliberately select the representation, learning mechanism, and architecture that best correspond to the decision the business actually needs to make.

As seniority increases, it becomes less important to ask only, "Which model should we use?" and more important to answer a broader question: **what is the simplest, most reliable, secure, and economically sustainable architecture capable of correctly representing this problem and producing the expected business outcome?**

Practical Framework for Selecting the AI Paradigm

A good architecture begins with the structure of the decision, not with the catalog of available models. The first step is to identify the type of problem:

- **We need to predict a class:** Traditional Classification may be sufficient;
- **We need to predict a value:** Regression or forecasting may represent the problem more effectively;
- **We need to order items:** Ranking and recommendation should be considered;
- **We need to interpret an image:** Classification, detection, and segmentation must be differentiated;
- **We need to explore relationships among entities:** Graph features or Graph ML may provide additional signal;
- **We need to take a sequence of actions:** Bandits or Reinforcement Learning may be appropriate;
- **We need to combine different sources and modalities:** A compound AI system may be necessary.

When the problem is recommendation, several questions are fundamental:

- **What is the candidate universe:** Catalog size determines whether candidate generation is required;
- **How will candidates be generated:** Collaborative, content-based, embeddings, or rules can be combined;
- **Is sufficient historical data available:** Cold start requires an explicit strategy;

- **What objective will the ranker optimize:** The proxy must reflect the desired behavior;
- **Which guardrails will be required:** Short-term metrics should not compromise the ecosystem;
- **How will feedback loops be controlled:** Current exposure modifies future data.

When the problem is ranking, it is necessary to understand how the list will be consumed:

- **The user observes only Top-K:** Metrics must place greater weight on the highest-ranked positions;
- **The first relevant result is sufficient:** MRR may better represent the experience;
- **Pointwise satisfies the objective:** More complex approaches should only be adopted if they produce material gains;
- **Pairwise or listwise justifies the cost:** Training complexity must generate measurable improvement;
- **The offline metric is related to the actual outcome:** A ranking improvement without product impact may have no value.

For forecasting, temporal structure comes first:

- **What is the forecast horizon:** The decision determines the required lead time;
- **Is there trend or seasonality:** These patterns must be represented and monitored;
- **How much does uncertainty matter:** Asymmetric risk may require probabilistic forecasting;
- **Does validation respect chronology:** Point-in-time correctness must be guaranteed;
- **Is there a risk of regime change:** The pipeline must detect when the past no longer represents the future.

In Computer Vision, granularity must follow the decision:

- **Is Classification sufficient:** There is no reason to pay for localization that will not be used;
- **Are bounding boxes necessary:** Detection adds cost but may be an operational requirement;
- **Is pixel-level segmentation actually necessary:** The additional precision must provide value;
- **How will the images be captured:** The sensor and environment are part of the architecture;
- **Is edge inference required:** Hardware and energy can constrain model selection;
- **Does the gain from larger models justify additional infrastructure:** Benchmark improvements must translate into operational outcomes.

For Graph ML, the main question is whether relationships provide additional signal:

- **Does the graph contain information that tabular features do not capture:** Without material improvement, Graph ML may be unnecessary;
- **Is entity resolution reliable:** Incorrect relationships contaminate the entire structure;
- **How many nodes and edges exist:** Scale determines sampling and partitioning strategies;
- **Can precomputed embeddings solve part of the problem:** Not every product requires an online GNN;
- **Is there temporal leakage:** The graph being used must represent the actual state at the time of the decision.

For sequential decisions, RL must be justified:

- **Does the current action change future states or rewards:** This dependency is one of the primary reasons to use RL;
- **Can a supervised policy solve the problem adequately:** If so, it will probably be simpler;
- **Does the reward truly represent the objective:** Proxies must be analyzed as incentives;
- **Is exploration safe:** The platform cannot learn through unacceptable actions;
- **Are offline data or simulation available:** These options can reduce risk;
- **Which actions require external guardrails:** Safety should not depend solely on learning.

After selecting the paradigm, the analysis must return to traditional engineering:

- **What latency is required:** Some decisions must occur within milliseconds, while others can be processed in batch;
- **What throughput must the system sustain:** Volume can completely change the architecture;
- **How much storage, memory, and compute will be required:** Cost must scale in proportion to value;
- **Is there a fallback:** The unavailability of an AI component should not necessarily bring down the product;
- **Which components can be precomputed:** Offline processing can simplify the online path;
- **Which parts genuinely need to operate in real time:** Reducing synchronous work generally improves reliability and cost.

Security and governance must be evaluated before deployment:

- **Can feedback be manipulated:** Behavioral systems are vulnerable to poisoning;
- **Is visual data sensitive:** Privacy and retention must be part of the design;
- **Does the graph reveal private relationships:** Derived information must also be protected;
- **Can a policy execute dangerous actions:** Action constraints must exist outside the model;
- **Is there an audit trail:** Important decisions must be reconstructable.

Finally, the entire architecture must return to the economic objective:

- **Which KPI will change:** The value-creation mechanism must be explicit;
- **What incremental value is expected:** Technical improvements must be compared with a baseline;
- **What is the total cost of the architecture:** Compute, data, operations, and people must be considered;
- **What additional complexity will be introduced:** Every new dependency has a maintenance cost;
- **Can a simpler solution produce a similar outcome:** If the answer is yes, the additional sophistication may not be justified.

This is the mental model expected of a Staff AI Engineer: **understand the structure of the problem before selecting the paradigm, combine techniques only when necessary, and build systems that maximize value without paying for complexity, risk, or infrastructure that the business does not need.**