AI Evals for Everyone - Complete Course

Created by Aishwarya Naresh Reganti & Kiriti Badam

Table of Contents

Chapter 1: WTH are AI Evals?

What Are Evals and Why Do They Suddenly Matter?

If you have been following recent AI updates, especially in the product space, you have probably heard the term *evals* come up repeatedly. It shows up in conversations, blog posts, product reviews, and conference talks. Everyone seems to use it casually, yet everyone also seems to mean something different by it.

The usefulness of evals is debated heavily, often without clarity on what people are actually referring to or where they are coming from. This lack of precision has led to a growing number of misconceptions as teams build AI solutions in the real world.

We put together this guide as a practical starting point. Think of it as a 101. The goal is to explain *why* evaluation is needed, *what* it actually refers to in practice, and *where* people commonly misunderstand it when building AI products.

Our hope is that by the end of this course, you are able to separate noise from signal, understand how practitioners on the ground think about evaluation, and start building evaluations using a first principles approach.

The Shift That Makes Evals Unavoidable

Before getting into evaluation itself, we need to build intuition for a much larger shift. AI systems and AI products are fundamentally *non-deterministic*. We will use this term often throughout this chapter and the rest of the course, because it is the single biggest reason evals exist in the first place.

Most of us have spent our careers working with traditional software products. In those systems, the number of actions a user can perform is usually limited. Users click buttons, fill out forms, upload a photo, submit a request, or complete a predefined flow. In most cases, both the input and the expected output are known ahead of time. If a user uploads a photo, the system should store it. If they submit a form, the backend should validate and process it. The code is written to explicitly enforce these expectations.

To make sure the product behaves as intended, teams rely on unit tests and integration tests. The core assumption is simple. Given an input x, the system should reliably produce output y. If you can verify this offline, you can be reasonably confident the product will behave the same way in production.

The standard software lifecycle follows a familiar pattern. You build version one of the product, add tests to ensure it works, test it with different users, fix bugs, and then continue building on top of that foundation.

The expectation is that if you have tested x to y thoroughly before shipping, production issues will be relatively rare and manageable. When problems do occur, they are often clear outliers that can be debugged and fixed.

How AI Products Break Classical Assumptions

AI products break these assumptions in two fundamental ways.

First, the input space becomes effectively unbounded. Most AI products accept text, voice, images, or video as input. Users are no longer selecting from predefined flows or filling structured forms. They are expressing intent in natural language, often ambiguously, incompletely, or in ways the product team never anticipated. You no longer control how users frame their requests. You only control how the system attempts to respond.

Second, the output is no longer guaranteed. The same input, or even small changes in phrasing, can yield different responses across runs. This is a property of the models powering these products. They are highly sensitive to context and phrasing, and they produce probabilistic outputs rather than fixed answers.

Traditional unit and integration tests answer a narrow question. *Did the system do exactly what we expected?* In AI products, there are two unknowns that make this question insufficient.

First, you do not fully know how end users will interact with your system.

Second, you do not have direct visibility into how the model arrives at its answers.

Large language models are black boxes. There is no simple pass or fail signal.

So How Do Teams Build with Confidence?

In practice, teams start by estimating how users might interact with the system. For example, if you are building an agent to help answer customer queries for a large retail company like Amazon or Walmart, you can look at historical customer support data to understand commonly asked questions. You can then test how your system responds to those questions before launch.

A simple way to think about this is a table like the following:

User Question	Expected Correct Answer	Agent Generated Answer
I can't seem to refund my shoes, it's been 45 days	Explain return policy and escalate	[System Response]
I requested a refund a week ago and haven't gotten it	Check status and provide update	[System Response]

This is often where people first encounter the idea of evaluation. It is also where confusion usually starts.

Why the Word "Evals" Causes Confusion

A major source of confusion is that the term *evals* is used loosely to refer to very different things. In this course, we will use the word *evaluation* intentionally, because *evals* has become a catch-all term that hides important distinctions.

Broadly, there are two kinds of evaluations.

Model Evaluations

Model evaluations are primarily conducted by frontier labs and research teams. Their goal is to answer a specific question. *How capable is this model in general compared to others?*

These evaluations rely on standardized benchmarks that test reasoning, factual recall, coding ability, or performance on academic style tasks. They are usually run on fixed datasets with predefined expected answers, and the model's outputs are scored across multiple dimensions using evaluation metrics.

Model evaluations are valuable. They help researchers measure progress, help infrastructure teams choose base models, and help vendors communicate improvements.

However, they are intentionally broad and domain agnostic. They are not designed to tell you whether a model will work well inside a specific product, workflow, or business context.

AI Product Evaluations

AI product evaluations are what practitioners should care about most when building real products. Product evaluations focus on whether a system behaves acceptably in a specific domain, for a specific workflow, and for real users.

Real world data is far more nuanced than benchmark datasets. Domain rules, edge cases, risk tolerance, and downstream consequences matter deeply. A model that performs well in general may still fail in ways that are unacceptable for your product.

Even if frontier labs have done extensive model evaluations, product teams still need their own evaluation process. Model evaluations tell you what a model can do in general. Product evaluations tell you whether it should be used in your system.

In the rest of this course, we focus on AI product evaluations and product evaluation metrics, not model evaluations. This is the level at which product teams actually make decisions and manage risk.

Clearing Up the Terminology

Before moving on, let us clearly define the terms we will use throughout this course. These words are often used interchangeably in conversations, which is one of the main reasons people get confused about evaluation.

Evaluation refers to the overall process of assessing how an AI system behaves. It is not a single test, score, or dashboard. Evaluation is the act of checking whether a system's outputs meet certain expectations under specific conditions. This can happen before launch, after launch, or continuously in production.

A benchmark or evaluation harness is the setup used to run evaluations in a repeatable way. This usually includes a dataset of example inputs, any required context, and a defined execution process. Benchmarks exist to ensure consistency across different evaluation runs and make results comparable.

Evaluation metrics are the dimensions along which system behavior is judged. A metric answers the question, *what does good mean in this context?* Common examples include correctness, relevance, completeness, safety, tone, or helpfulness. Metrics can be objective or subjective, but they are always context dependent.

The same metric can mean very different things in different domains. Take *helpfulness* as an example. In a real estate product, helpfulness might mean summarizing listings clearly, surfacing relevant comparables, or asking clarifying questions when user intent is vague. Over explaining or speculating would be harmful, even if the response sounds articulate.

In an insurance or healthcare workflow, helpfulness might mean knowing when *not* to answer. Escalating uncertainty, flagging missing information, or deferring to a human can be more helpful than attempting to provide a complete answer.

Because of this, evaluation metrics must almost always be guided by explicit *rubrics*. A rubric defines what good looks like and what failure looks like in a given context. Without rubrics, metrics like helpfulness, correctness, or safety become vague labels that different people interpret differently.

Model evaluations assess general model capability independent of any specific product.

AI product evaluations assess whether a model behaves acceptably inside a real product.

Throughout this course, when we talk about evaluation, we are referring to AI product evaluations unless stated otherwise. Our goal is not to measure how intelligent a model is in the abstract, but to understand whether a system is behaving well enough for the product we are trying to build.

Key Takeaways

As you can see, *evals* is an overloaded term that means different things to different stakeholders. When someone says PMs should do evals, they often mean defining rubrics and evaluation metrics and expectations for product behavior. When someone says a model's evals look good, they usually mean benchmark scores on popular datasets. When data labeling companies talk about writing evals, they typically mean creating training datasets and annotation guidelines.

Understanding these distinctions is crucial for building effective evaluation systems that actually help you ship better AI products.

At this point, you should have a clearer mental map. In the next chapter, we will look at the different ways evaluations are implemented in practice, and the tradeoffs each approach introduces.

Chapter 2: Model vs Product Evaluations

From Model Creation to Product Implementation

In the previous chapter, we established that AI evaluation is unavoidable. Now we need to understand how evaluation works across the AI ecosystem.

When AI companies build models, they evaluate them to understand their general capabilities. But these same models get used in thousands of different applications, from customer support chatbots to legal document analysis to medical diagnosis tools. Each application has its own requirements, constraints, and success criteria.

This creates a natural progression: models are evaluated for their general abilities, then they need additional evaluation when you use them for specific purposes. The first tells you what a model can do in theory. The second tells you whether it actually works for your particular use case.

Model creators have their own perspective on evaluation worth understanding first.

Model Evaluations: Measuring General Capability

When AI companies develop models, they need to understand and communicate what their models can do. **Model evaluations** serve this purpose, measuring general capability across broad domains. Their goal is straightforward: *How capable is this model compared to others?*

You see these evaluations in research papers, vendor marketing materials, and leaderboards.

These evaluations use standardized benchmarks that test different aspects of model capability:

• **MMLU (Massive Multitask Language Understanding)**: Tests knowledge across 57 academic subjects from elementary math to professional law

• **HumanEval**: Measures coding ability by testing whether models can write Python functions that pass unit tests

• **GSM8K**: Tests grade-school level mathematical reasoning

• **GPQA**: Tests graduate-level reasoning in physics, chemistry, and biology

Model evaluations serve as a competitive landscape for AI providers. When companies release new models, they publish benchmark scores to demonstrate improvements and establish market positioning. A model that scores 85% on MMLU versus 78% on the previous version signals meaningful progress to potential customers and the research community.

The industry benefits from this standardization in several ways. Teams can track scientific progress over time, organizations can compare and choose between different models, and everyone gets objective measures that cut through marketing claims.

The benchmarks are carefully designed to be objective, repeatable, and comparable across different models. Model evaluations can test both general capabilities and domain-specific knowledge, but they're designed to assess what models can do in standardized conditions, not how they'll perform in your specific business context with your particular constraints and requirements.

Why Model Evaluations Don't Predict Product Success

Here's a concrete example. Suppose you're building an AI system to help insurance agents process claims. You're choosing between two models:

• **Model A** scores 92% on MMLU and 85% on HumanEval

• **Model B** scores 87% on MMLU and 79% on HumanEval

Based on benchmark scores, Model A looks clearly superior. When you test them on actual insurance claims with your specific data, workflows, and business constraints, Model B might perform significantly better.

Why? Because your insurance use case has specific requirements that general benchmarks don't capture:

• **Domain knowledge**: Understanding insurance terminology, regulations, and claim types

• **Risk tolerance**: The cost of approving a fraudulent claim versus denying a legitimate one

• **Business constraints**: Processing time requirements, escalation policies, compliance needs

• **Real-world messiness**: Incomplete forms, ambiguous language, edge cases specific to insurance

Model B might have seen more insurance-related data during training, or its architecture might be better suited to the structured reasoning required for claims processing. The benchmark scores can't tell you this.

Real-World Context Is Far More Complicated

The data that powers your business lives in industry-specific silos that benchmark creators never see. Healthcare data has different patterns than financial data. Legal documents follow different structures than customer support conversations. Manufacturing quality reports contain domain knowledge that doesn't exist in academic datasets.

This means benchmark performance often fails to predict real-world behavior. Consider a customer support AI that scores well on standard helpfulness benchmarks. When a frustrated customer types "this is the third time I'm contacting you about my broken order and nobody seems to care," the AI needs to:

• Recognize the emotional context and escalation history

• Know when to apologize versus when to escalate immediately

• Understand your company's specific policies and capabilities

• Balance being helpful with managing expectations appropriately

These nuanced requirements emerge from your specific business context, customer base, and operational constraints. They don't appear in general benchmarks, but they're critical for your product's success.

AI Product Evaluations: What Actually Matters for Your Business

AI product evaluations focus on a different question: *Does this system behave acceptably for our specific use case, with our users, in our domain?*

Product evaluations are context-dependent by design. They test whether the AI system:

• Handles your specific user inputs appropriately

• Follows your business rules and constraints

• Escalates correctly when uncertain

• Maintains appropriate tone and style for your brand

• Manages risk according to your tolerance levels

The metrics you track in product evaluation often look very different from model evaluation metrics. Instead of general correctness, you might measure:

• **Escalation accuracy**: Does the system correctly identify when it should hand off to a human?

• **Policy compliance**: Does it follow your company's specific guidelines and constraints?

• **Risk management**: How often does it make decisions you later have to reverse?

• **User experience**: Are users able to complete their tasks efficiently?

A Practical Example: Legal Document Analysis

Imagine you're building an AI system to help lawyers review contracts. Two different evaluation approaches would look completely different:

Model Evaluation Approach

• Test general reading comprehension on legal text

• Measure accuracy on standardized legal reasoning benchmarks

• Compare performance to other models on academic legal datasets

Product Evaluation Approach

• Test on your firm's actual contract types and templates

• Measure how often it catches the specific risk patterns your lawyers care about

• Evaluate whether it flags clauses that your legal team would want to review

• Test escalation behavior when it encounters unusual or high-risk terms

• Measure time savings for your lawyers while maintaining quality standards

The model evaluation tells you the AI can understand legal language in general. The product evaluation tells you whether it can actually help your lawyers do their job better.

Focus on Product Evaluation for Builders

Now that we understand both approaches, it becomes clear that model evaluation alone isn't really useful for builders. While model evaluations help with initial model selection, the real work happens at the product evaluation level.

Baseline capability assessment: If a model performs poorly on relevant general benchmarks, it's unlikely to work well in your specific domain. Model evaluations can help you eliminate obviously unsuitable options.

Comparative analysis: When choosing between models with similar architectures, benchmark scores can provide useful signals about relative capability, especially when combined with product-specific testing.

Progress tracking: If you're fine-tuning or customizing a model, general benchmarks can help you verify that you're not degrading core capabilities while adding domain-specific knowledge.

But model evaluations should be just the starting point, not the endpoint, of your evaluation process.

Building Your Product Evaluation Strategy

Given this distinction, how should you approach evaluation for your AI product?

Start with model evaluations as a filter. Use benchmark scores to eliminate models that lack the basic capabilities your application requires. If you need strong reasoning ability, look for models that perform well on reasoning benchmarks. If you need multilingual support, check language-specific evaluations.

But invest your time in product evaluations. This is where you'll discover whether the AI actually works for your use case. Design evaluations that test:

• Your specific user inputs and edge cases

• Your business constraints and requirements

• Your risk tolerance and escalation needs

• Your quality standards and success metrics

Use real data whenever possible. Synthetic test cases are useful for getting started, but nothing beats evaluating on actual examples from your domain with real user inputs and expected outputs.

Make it an ongoing process. Unlike model evaluations, which are typically done once during model selection, product evaluation should be continuous. User behavior evolves, business requirements change, and your AI system needs to adapt.

The Evaluation Hierarchy

Think of evaluation as a hierarchy:

1. **Model capability**: Can this model handle the type of task I need? (Model evaluation)

2. **Domain fit**: Does it work well with my specific data and requirements? (Basic product evaluation)

3. **Production readiness**: Does it behave safely and reliably with real users? (Comprehensive product evaluation)

4. **Continuous improvement**: How do I maintain and improve performance over time? (Ongoing product evaluation)

Most teams spend too much time on level 1 and not enough on levels 2-4. The companies that succeed with AI products flip this priority.

Key Takeaways

Model evaluations and product evaluations serve fundamentally different purposes. Model evaluations help you understand general capability and compare different models. Product evaluations tell you whether an AI system will actually work for your business.

The benchmark illusion (assuming strong model evaluations guarantee product success) is one of the most common reasons AI projects fail to translate from demos to production.

Your evaluation strategy should use model evaluations as an initial filter but invest most of your effort in product-specific evaluation that tests real use cases, real data, and real business requirements.

In the next chapter, we'll dive into a systematic framework for thinking about AI system behavior that will help you design product evaluations that actually predict real-world performance.

Chapter 3: The Evaluation Framework

Setting Up Evaluation for Your AI Product

In the previous chapters, we covered why evaluation matters and the difference between model and product evaluation. Now you understand you need to evaluate your AI products.

If you want to evaluate a new product you're building, where do you start?

We'll cover the basic concepts that help you approach evaluation strategically. This will set you up to build datasets and evaluation systems that actually help you improve your product.

What You're Actually Evaluating

When you evaluate any AI system, you're looking at three things - **Input** (what goes into the system), **Expected** (what should happen), and **Actual** (what actually happens).

Sounds simple, but each piece is more complex than it appears.

Input: Everything That Affects Your System

"Input" isn't just the user's question. It includes everything that influences how your system behaves - the user's actual question or request, previous conversation history and context, data your system retrieves (documents, database entries, API calls), and system configuration (prompts, parameters, business rules).

This matters because many evaluation problems happen when teams only test the obvious user inputs but ignore how context and configuration changes affect behavior.

Expected: What Good Looks Like

Defining what should happen is often the hardest part. What should your system actually do?

Expected behavior depends on your specific requirements:

• Accuracy of information

• Completeness of the response

• Appropriate tone and style

• Safety and compliance

• Following your business rules

Take a healthcare AI. When someone asks "Is this medication safe for children?", good behavior isn't just giving accurate information. It includes:

• Noting that medical advice should come from doctors

• Suggesting they talk to their pediatrician

• Providing general information without specific medical recommendations

• Escalating if the situation seems urgent

Defining expected behavior requires input from both technical teams and people who understand the domain and business context.

Actual: What Your System Really Does

This is what your system produces: the response, the actions, the decisions.

But "actual" includes more than just the final output:

• The content and quality of responses

• Which tools or data sources were used

• Reasoning and decision-making process

• Performance metrics like response time

Understanding what actually happens often requires logging different parts of your system.

Why Generic Metrics Don't Work

Once you understand these three pieces, you can see why simple metrics like "helpfulness" or "correctness" don't work for real AI products.

The same metric means completely different things depending on your context and requirements.

Context Changes Everything

Take "helpfulness." What's helpful depends entirely on the situation:

Customer service: Helpful means solving problems quickly and escalating when needed. Explaining too much when someone just wants a refund isn't helpful.

Education: Helpful means guiding students to understanding, not just giving answers. A direct solution without explanation isn't helpful even if it's correct.

Medical information: Helpful means providing accurate general information while being clear about limitations. Being too specific about medical advice would be harmful.

This is why you can't just copy evaluation metrics from other applications. You need to define what quality means for your specific situation.

Multiple Dimensions Matter

Real systems need evaluation across several specific areas. Here's why:

Say a customer asks "Can I return my shoes after 45 days?" and your system responds:

"Unfortunately, our return policy only allows returns within 30 days of purchase. However, since you're clearly frustrated about this situation and have been a loyal customer, I understand your disappointment. While I cannot process the return myself, I recommend contacting our customer care team who may be able to offer alternative solutions or exceptions based on your purchase history and the specific circumstances of your case."

You need to evaluate this across several areas:

• **Policy accuracy**: Does it correctly state the 30-day policy?

• **Escalation**: Does it properly refer to the right team?

• **Tone**: Is it professional and empathetic without over-apologizing?

• **Business risk**: Does it avoid making unauthorized promises?

A single "correctness" score would miss important problems. The response could be accurate about the policy but fail to escalate properly, or escalate correctly but use the wrong tone.

The key is finding the minimum set of areas that give you the most signal about what matters for your specific product.

These areas are what we call **evaluation metrics**. Some people call these "evals," but we prefer to be more precise. We use "evaluation" for the overall process and "evaluation metrics" for the specific measures you use to judge quality.

Making Subjective Assessment Consistent

Many important quality areas require subjective judgment. Different people might evaluate the same response differently when looking at tone, appropriateness, or escalation decisions.

Rubrics solve this by providing explicit criteria for judgment.

Building Simple Rubrics

A good rubric defines:

• What counts as acceptable versus not acceptable performance

• Specific things to look for

• Examples of responses in each category

• How to handle edge cases

For example, a rubric for "appropriate escalation" might specify:

Acceptable: Correctly identifies situations that need human intervention (policy exceptions, billing disputes, complex technical issues) and provides appropriate context when escalating

Not Acceptable: Fails to escalate when human intervention is needed, escalates unnecessarily for routine questions, or escalates without sufficient context

Rubrics make subjective evaluation more consistent and help different team members align on quality standards.

Why Evaluation Requires Team Collaboration

Effective AI evaluation isn't just a technical problem. It requires collaboration between different roles:

Subject matter experts understand what good behavior looks like in the domain. They know the edge cases, risks, and nuances that technical metrics might miss.

Product teams understand user needs and business priorities. They know what trade-offs matter and how evaluation connects to user experience.

Engineers understand system capabilities and constraints. They know what's measurable, what's technically feasible, and how to implement evaluation systems.

This collaboration matters because evaluation decisions affect every aspect of your AI product. The metrics you choose influence what behaviors you optimize for and how you measure success.

Building Team Alignment

One of the most valuable outcomes of systematic evaluation is getting everyone aligned on quality. When engineering, product, and domain expert teams can look at the same examples and agree on good versus poor performance, you can move much faster.

This process often reveals hidden assumptions and disagreements. The product team might prioritize user satisfaction while the legal team prioritizes risk management. Working through evaluation examples helps surface and resolve these tensions before they affect the product.

How Do You Identify the Right Dimensions?

So far we've talked about why you need specific evaluation metrics and why collaboration matters. But how do you actually figure out which dimensions to focus on?

The process usually starts with understanding your specific failure modes. What could go wrong with your AI system that would be unacceptable for your users or business? What behaviors would make you pull the system offline immediately?

Different stakeholders will have different answers:

• **Domain experts** worry about accuracy, compliance, and safety risks specific to their field

• **Product teams** focus on user experience, completion rates, and satisfaction

• **Business stakeholders** care about liability, brand risk, and operational costs

The key is starting with these concerns and translating them into observable, measurable behaviors. Instead of "the system should be safe," you might define "the system should escalate medical questions to qualified professionals" or "the system should not provide financial advice without appropriate disclaimers."

You also need to consider your specific user context. A chatbot for customer service has different quality requirements than one for technical support or educational tutoring. The same AI technology needs completely different evaluation approaches depending on who's using it and for what purpose.

The Pre-Deployment Validation Process

The approach we're describing helps you validate your AI system before you put it in front of real users. This pre-deployment validation is essential because it's much easier to catch and fix issues in controlled testing than after users start depending on your system.

Think of this as building confidence in your system's behavior before the stakes get high. When you're working with reference datasets and controlled examples, you can iterate quickly, test edge cases thoroughly, and refine your approach without worrying about user impact. You can have domain experts review outputs carefully, engineers can debug issues systematically, and product teams can ensure the behavior aligns with user needs.

This validation process involves several key activities:

Building comprehensive test scenarios: You'll create examples that represent the full range of situations your system needs to handle, from common user requests to edge cases that could cause problems.

Establishing clear quality criteria: You'll work with stakeholders to define exactly what good behavior looks like in your specific context, creating rubrics that everyone can agree on.

Testing system behavior systematically: You'll run your AI system against your test scenarios and evaluate whether it meets your quality standards across different dimensions.

Iterating based on findings: When you discover issues, you'll fix them and re-test to ensure the problems are resolved without creating new ones.

Once you deploy and real users start interacting with your AI, you'll need to adapt these same concepts for ongoing monitoring. Real-world conditions introduce new challenges like unpredictable user behavior, scale issues, and evolving requirements that require different approaches while building on the same evaluation foundation.

Where This Leads

Understanding what goes into your system, what should happen, and what actually happens helps you see why AI evaluation is more complex than traditional software testing. The challenge isn't just technical - it's about getting alignment across different perspectives on quality.

Generic metrics like "helpfulness" mean different things in different contexts. Effective evaluation requires specific metrics that reflect your domain, users, and business requirements.

AI evaluation is inherently collaborative. Subject matter experts, product teams, and engineers each bring essential perspectives to defining what good performance looks like.

But this raises an important question: how do you actually come up with all these metrics? How do you know which ones matter most for your specific situation? How do you balance different team perspectives to create evaluation criteria everyone can agree on?

In the next chapter, we'll talk about building a reference dataset so you can understand how to apply this framework and improve your system once you've built a version of your product. We'll cover how you could set this up in a more systematic way.

Chapter 4: Building Reference Datasets

Getting Started with Systematic Evaluation

In the previous chapter, we covered why you need specific evaluation metrics and how different stakeholders bring different perspectives to defining quality. Now let's talk about how to set this up systematically.

You've built a version of your AI product and you want to evaluate it properly before putting it in front of real users. Where do you start?

The most practical approach is building a reference dataset. Think of this as a small, carefully chosen set of examples that represent the scenarios you care most about. It's not meant to be comprehensive - it's meant to be useful for validating your system's behavior in a controlled environment before deployment.

A note on system complexity: In this guide, we'll focus on single-step AI interactions (where the user asks something and the system responds). Many complex AI systems involve multiple steps like calling tools, multi-turn conversations, or reasoning chains, but the core ideas we'll cover can be translated to those more complex scenarios as well.

What Is a Reference Dataset?

A reference dataset is your first concrete representation of how the system should behave when deployed. It's a collection of realistic inputs paired with what you expect the system to do in those situations.

The key word here is "realistic." These aren't made-up test cases. They're examples that reflect how real users will actually interact with your system, including the messy, ambiguous, and edge-case scenarios that always happen in production.

Each example in your dataset typically includes:

• **Input**: A realistic user request or scenario

• **Expected output**: What the system should do (written in plain language)

• **Context**: Any additional information the system needs

The expected output doesn't have to be a perfect response. It can be a description of the right behavior, like "escalate to human agent" or "ask for clarification about the user's budget range."

Why Start Small and Specific

Teams often make the mistake of trying to build comprehensive test coverage from day one. This doesn't work well for AI systems.

Instead, start with a small set of examples that represent scenarios you absolutely cannot get wrong. These are usually:

• High-risk situations where failure would be unacceptable

• Common user workflows that need to work smoothly

• Edge cases that reveal important system limitations

• Examples that expose different evaluation dimensions you care about

For a customer support AI, this might include:

• A billing dispute that requires human escalation

• A simple return request that should be handled automatically

• An angry customer message that needs careful tone handling

• A request that's outside your company's service scope

Starting small lets you focus on quality over quantity. It's better to have 20 well-chosen examples with clear expected behaviors than 200 generic test cases.

Step 1: Generate Your Initial Examples

The best source for examples is usually existing data from your domain. If you have historical customer support tickets, user queries, or domain-specific scenarios, start there.

If you don't have existing data, this is where collaboration becomes essential:

Subject matter experts should contribute the majority of initial examples. They know the edge cases, the high-risk scenarios, and the subtle requirements that technical teams might miss. Don't rely on engineers to generate domain-specific examples because they'll miss important nuances.

Product teams can contribute examples based on user research, feature requirements, and common user journeys they've observed.

Engineers can help identify technical edge cases and system boundary conditions.

We recommend avoiding AI-generated synthetic examples at this stage. AI can help with formatting or expanding on human-written examples, but it tends to create shallow scenarios that miss real-world complexity.

Example: Customer Support Dataset

Here's what an initial reference dataset might look like for a customer support system:

Input	Expected Behavior
"I want to return my shoes but I lost the receipt"	Ask for order number or email, explain receipt alternatives, process if sufficient info available
"Your service is terrible and I'm switching to a competitor"	Acknowledge frustration, apologize professionally, escalate to retention team
"How do I track my order?"	Ask for order number, provide tracking information, explain delivery timeline
"I was charged twice for the same order"	Apologize, escalate immediately to billing team with all available details

Notice these examples cover different scenarios (returns, complaints, tracking, billing) and different required behaviors (self-service, escalation, information gathering).

Step 2: Run Your System on the Examples

Once you have your initial set of examples, run your AI system on these inputs. Keep the system simple at this stage. Don't add complexity before you understand baseline behavior.

This step often reveals immediate problems with your system design, prompts, or configuration. That's valuable information.

Document both what the system outputs and any intermediate steps (which tools it used, what data it retrieved, how confident it was). This helps you understand not just what went wrong, but why.

Step 3: Evaluate Alignment with Domain Experts

Now you have three columns: input, expected behavior, and actual system output. The next step is getting domain experts to evaluate the alignment.

Keep this process simple. Ask domain experts to review each example and answer: "Was this response satisfactory? If not, why not?"

Avoid asking for numerical scores at this stage. Scores are harder to interpret and don't give you actionable feedback. Simple yes/no judgments with explanations are more useful.

Make it easy for domain experts to participate. Don't ask them to learn complex rubrics or scoring systems. Frame questions in their language and focus on their expertise.

Example Evaluation Session

Continuing with our customer support example:

Input	System Output	Expected	Satisfactory	If not, why?
"Lost receipt for shoe return"	"I can help you process a return. Please provide your order confirmation email."	Ask for order number or email, explain alternatives	Yes	Handled appropriately
"Service is terrible, switching"	"I understand you're frustrated. Let me see how I can help resolve your issue."	Acknowledge, apologize, escalate to retention	No	Didn't escalate to retention team

This gives you specific, actionable feedback about where your system is failing.

Step 4: Identify Error Patterns

Now bring the engineering perspective back in. Look at the annotations from domain experts and identify patterns in the failures.

Many issues that look different on the surface come from the same root cause. The goal is clustering errors into a small number of underlying problems that you can actually fix.

Add two more columns to your analysis:

• **Error category**: What type of failure is this?

• **Potential cause**: Why might this be happening?

Common error patterns include:

• **Missing context**: System doesn't have access to information it needs

• **Prompt issues**: Instructions aren't clear or specific enough

• **Business rule failures**: System doesn't follow domain-specific policies

• **Escalation problems**: Doesn't recognize when human intervention is needed

Example Error Analysis

Input	System Output	Satisfactory	Error Category	Potential Cause
"Service terrible, switching"	Generic help offer	No	Missing escalation	No escalation logic for retention cases
"Charged twice"	"Let me help with that"	No	Missing urgency	Billing issues not flagged as high-priority

This helps you prioritize fixes and understand whether issues are implementation problems or deeper design issues.

Step 5: Decide Which Metrics You Need

Here's a key insight: if an issue can be fixed once and is unlikely to return, fix it and move on. If an issue represents a behavior that can reappear in different forms, you need an ongoing metric to track it.

For example:

• A missing instruction in a prompt is usually a one-time fix

• Appropriate escalation behavior is an ongoing concern that needs monitoring

Create metrics for recurring risks, not one-off bugs.

Be ruthless about what you measure. You want the minimum number of metrics that give you the maximum amount of signal. If there are issues in your use case that you're not really worried about (small things that don't significantly impact your users or business), don't add a metric for them.

Only create metrics for behaviors you actually care about and can take action on. If you wouldn't change your system based on a metric, don't track it.

Based on your error analysis, identify 2-4 key behaviors that need ongoing measurement. These become your evaluation metrics. More than that becomes difficult to manage and act upon effectively.

For our customer support example, you might end up with:

• **Escalation accuracy**: Does the system correctly identify when human intervention is needed?

• **Information gathering**: Does it ask for the right information to resolve requests?

• **Tone appropriateness**: Does it match the professional, helpful brand voice?

Focus on what matters, not implementation. At this stage, don't worry about how you'll measure these behaviors. Just identify which behaviors are most important for your use case. We'll cover implementation approaches in the next chapter.

For now, examples of metrics you might track include:

• Response time stays under acceptable limits

• Required legal disclaimers appear in financial advice responses

• Billing-related queries get properly flagged for escalation

• System outputs maintain valid structure for downstream processing

• Appropriate tone and empathy in customer interactions

• Accurate assessment of query complexity for escalation decisions

• Relevant information gathering without being repetitive

What matters most is identifying which failure modes are critical for your specific system and user needs.

Step 6: Iterate and Expand

Your reference dataset isn't static. As you fix issues and learn more about system behavior, add new examples that represent:

• Edge cases discovered in production

• New failure modes that emerge

• Additional scenarios your system needs to handle

The dataset grows into a record of hard-won understanding about what good behavior looks like in your domain.

Common Pitfalls to Avoid

Don't make it too big too fast: Start with 10-20 high-quality examples rather than 100 mediocre ones.

Don't rely entirely on synthetic data: AI-generated examples often miss real-world complexity and edge cases.

Don't skip domain expert involvement: Technical teams alone cannot define what good behavior looks like in specialized domains.

Don't create metrics for every issue: Focus on recurring risks that need ongoing monitoring.

Don't make rubrics too complex: Simple "acceptable/not acceptable" categories work better than elaborate scoring systems.

What You End Up With

After following this process, you'll have:

• A reference dataset that represents scenarios you care about

• Clear definitions of what good behavior looks like

• Specific metrics that track the most important behavioral dimensions

• Rubrics that make subjective evaluation consistent

• A process for expanding and refining your evaluation over time

This becomes the foundation for ongoing evaluation and improvement. Every time you make changes to your system, you can run it against your reference dataset to check for regressions. Every time you discover new edge cases in production, you can add them to improve your evaluation coverage.

The goal isn't perfect evaluation - it's systematic improvement. Your reference dataset helps you move from vague concerns about system behavior to concrete, measurable criteria you can act on.

In this chapter, we've identified what metrics are important to track based on your specific failure modes and business requirements. In the next chapter, we'll talk about how to implement these metrics, from simple code-based checks to more sophisticated evaluation approaches.

Chapter 5: Implementing Evaluation Metrics

From What to How

In the previous chapter, we walked through building reference datasets and identifying which metrics matter for your system. You now have a clear list of behaviors you want to track, such as escalation accuracy, response time, or tone appropriateness.

Now comes the practical question: how do you actually measure these behaviors?

This chapter covers the different approaches you can use to implement your metrics. We'll explore when each approach works well, their trade-offs, and how to choose the right mix for your specific situation.

Three Ways to Measure AI Behavior

There are three main approaches to implementing evaluation metrics:

Human evaluation: Having people assess AI system behavior based on their expertise and judgment

Code-based metrics: Deterministic checks written in code that look for specific patterns or properties

LLM judges: Using one model to evaluate another model's behavior

Each approach has strengths and weaknesses. Most effective evaluation systems use a combination of multiple approaches.

Human Evaluation: The Gold Standard

Human evaluation is exactly what we did when building the reference dataset in the previous chapter. You show examples to domain experts, product managers, or other stakeholders and ask them to judge whether the AI system's behavior was acceptable.

This approach has major advantages:

• **Nuanced judgment**: Humans can assess complex, subjective qualities like appropriateness, empathy, and contextual correctness

• **Domain expertise**: Subject matter experts understand subtleties that automated systems miss

• **Flexibility**: Humans can adapt their evaluation criteria on the fly when they encounter edge cases

• **Ground truth**: Human judgment often serves as the standard that other metrics try to approximate

Why Human Evaluation Doesn't Scale

The problem with human evaluation is obvious: it's slow and expensive. If you had to have a human evaluate every single conversation your AI system has in production, you'd need an army of evaluators working around the clock.

Imagine a customer support AI that handles 10,000 interactions per day. Having domain experts review each one would be impractical and cost-prohibitive. Even sampling 1% would require evaluating 100 interactions daily.

This is why we need the other automated approaches. They're attempts to capture human-like judgment at scale. The goal is finding automated methods that correlate well with human evaluation while being fast and cost-effective enough to run in production.

When to Use Human Evaluation

Human evaluation still has important roles:

• **Calibrating automated metrics**: Use human judgment to test whether your LLM judges or other metrics align with expert assessment

• **Edge case analysis**: When automated metrics flag something as problematic, humans can investigate whether it's a real issue

• **Periodic sampling**: Regularly evaluate a small sample of interactions to ensure your automated systems are working correctly

• **High-stakes decisions**: For critical interactions or when the cost of errors is very high

Code-Based Metrics: When Rules Work

Code-based metrics are deterministic checks you can implement with regular programming. They're fast, reliable, and easy to understand.

These work well when you can define success clearly and objectively:

Structure validation: Check if the response contains required fields, follows JSON format, or includes mandatory disclaimers

Performance metrics: Measure response time, token count, or API call frequency

Content detection: Verify specific phrases appear (like "please consult your doctor" in medical responses) or don't appear (like specific banned words)

Classification flags: Check if the system correctly tagged a query as "billing," "technical support," or "escalation needed"

Example: Structured Output Validation

Say you're building an AI system that helps sales teams qualify leads. The system needs to extract key information from customer conversations and output it in a structured format for the CRM system.

Your AI system should output JSON like this:

```json

{

"customer_name": "John Smith",

"company": "TechCorp",

"budget_range": "50000-100000",

"timeline": "Q2 2024",

"decision_maker": true,

"contact_email": "john@techcorp.com"

}

```

A code-based metric can easily verify:

• Is the output valid JSON?

• Are all required fields present (customer_name, company, budget_range)?

• Is the budget_range in the expected format?

• Is the decision_maker field a boolean?

• Is the contact_email field a valid email format?

This kind of check is perfect for code-based metrics because the requirements are objective and well-defined.

When Code-Based Metrics Fall Short

Code-based metrics struggle with subjective qualities like tone, appropriateness, or nuanced decision-making. You can't easily write code to detect whether a customer service response shows appropriate empathy or whether an escalation decision was justified.

They also miss nuanced meaning and context. A response might pass all the structural checks but still be unhelpful, inappropriate, or incorrect in ways that matter to users.

LLM Judges: Automating Human-Like Evaluation

LLM judges use one model to evaluate another model's behavior. The idea is to replace the manual human evaluation process we used when building reference datasets with an automated system that can make similar judgments at scale.

Instead of having domain experts review every response, you give an LLM the same criteria and ask it to assess whether the behavior was appropriate. This lets you evaluate thousands of interactions with the same standards a human expert would apply.

This approach works for subjective or complex evaluations:

Tone assessment: Is the response professional and empathetic?

Escalation decisions: Should this query have been escalated to a human?

Reasoning quality: Does the explanation make logical sense?

Safety evaluation: Does the response avoid harmful content?

Example: Customer Service Tone

For evaluating whether a customer service response shows appropriate empathy, an LLM judge can assess nuanced qualities like tone, professionalism, and contextual appropriateness that would be difficult to capture with code-based metrics.

Whether you're using LLM judges or human evaluators, you need clear criteria for what constitutes good and poor performance. This is where rubrics become essential.

A good rubric defines:

• **Acceptable performance**: Specific characteristics of good behavior

• **Not acceptable performance**: Clear failure criteria

• **Examples**: Concrete instances of each category

• **Edge case guidance**: How to handle ambiguous situations

Example: From Error Pattern to LLM Judge Rubric

Here's how you'd build an LLM judge based on the customer support example from Chapter 4.

The Problem: In your reference dataset evaluation, you found that when customers expressed frustration and mentioned switching providers (like "Service is terrible, switching"), your system gave generic help offers instead of escalating to the retention team.

The Pattern: Analysis revealed this was part of a broader "escalation accuracy" issue. The system wasn't recognizing when situations required specialized human intervention.

The Metric: You decided to track "escalation accuracy" as an ongoing metric since this behavior could reappear in many different forms.

The LLM Judge Rubric:

Acceptable:

• Correctly identifies customer retention situations (mentions switching, canceling, competitor comparisons, dissatisfaction with service)

• Escalates billing disputes over significant amounts ($100+)

• Recognizes technical issues beyond basic troubleshooting scope

• Provides relevant context when escalating (customer sentiment, issue details, urgency level)

Not Acceptable:

• Misses clear retention signals and attempts generic problem-solving

• Fails to escalate high-value billing disputes

• Tries to handle complex technical issues that require specialized expertise

• Escalates routine questions that could be resolved automatically

• Escalates without sufficient context for the human agent

Examples:

• **Acceptable**: "Your service is terrible and I'm switching to CompetitorX" → Escalates to retention team noting customer dissatisfaction and competitor mention

• **Not Acceptable**: "I want to cancel my subscription to save money" → Provides generic retention offer instead of escalating to retention specialists

• **Acceptable**: "I was charged $500 for services I never ordered" → Escalates to billing team with charge amount and dispute details

• **Not Acceptable**: "How do I reset my password?" → Escalates to technical support instead of providing standard reset instructions

This rubric now gives you a measurable way to track the escalation behavior that was failing in your reference dataset, turning the discovered error pattern into an ongoing monitoring capability.

LLM Judge Prompt Example:

Here's how you might structure a prompt for an LLM judge using this rubric:

```

You are evaluating customer service responses for escalation accuracy. Your job is to determine if the AI system correctly identified when human intervention was needed.

EVALUATION CRITERIA:

Acceptable Performance:
```

• Correctly identifies customer retention situations (mentions switching, canceling, competitors)

• Escalates billing disputes over $100

• Recognizes complex technical issues beyond basic troubleshooting

• Provides relevant context when escalating (sentiment, details, urgency)

Not Acceptable Performance:

• Misses clear retention signals and tries generic problem-solving

• Fails to escalate high-value billing disputes

• Attempts to handle complex technical issues requiring specialized expertise

• Escalates routine questions that could be resolved automatically

• Escalates without sufficient context for human agents

EXAMPLES:

• Customer: "Your service is terrible and I'm switching to CompetitorX"

Acceptable Response: Escalates to retention team noting dissatisfaction and competitor mention

Not Acceptable: Offers generic troubleshooting help

• Customer: "I was charged $500 for services I never ordered"

Acceptable Response: Escalates to billing team with charge details

Not Acceptable: Asks customer to verify their account information

TASK:

Review the customer input and AI response below. Determine if the escalation decision was:

• Acceptable

• Not Acceptable

Provide a brief explanation for your judgment.

Customer Input: [INPUT]

AI Response: [RESPONSE]

```
Your Evaluation:
```

This prompt gives the LLM judge the same detailed criteria that human evaluators would use, allowing it to make consistent assessments at scale.

A Note on LLM Judge Calibration

While we've shown you how to build an LLM judge with clear criteria and rubrics, remember that in practice, calibrating an LLM judge is a much longer and more data-driven process than what we've demonstrated here. LLM judges are powerful but challenging to implement well. They can be inconsistent, biased, or misaligned with human judgment. They're also more expensive and slower than other approaches.

Effective LLM judge calibration requires extensive testing against human judgment across hundreds of examples, not just a few. You need to systematically identify where the LLM judge disagrees with human evaluators, understand why those disagreements happen, and iteratively refine your prompts and criteria until alignment is acceptable for your specific use case.

Calibration is Essential

The biggest challenge with LLM judges is ensuring they actually align with human judgment. Just because you write detailed criteria doesn't mean the LLM will interpret them the same way a human expert would. In fact, if not calibrated properly they can add more problems to your system because they add another layer of non-determinism.

You need to test your LLM judge against human evaluations:

• Have humans evaluate a sample of examples using your rubric

• Run your LLM judge on the same examples

• Compare the results to see where they agree and disagree

• Refine your prompt and criteria based on the differences

• Repeat until alignment is acceptable

We leave you here since this is a 101 course, but building reliable LLM judges can be a course on its own. Remember to dig deeper on learning them well.

What You End Up With

After implementing your metrics, you'll have a measurement system that can:

• Automatically track the behaviors you care about most

• Run consistently across different examples

• Provide actionable feedback for system improvement

• Scale with your evaluation needs

This system becomes the foundation for continuous improvement. You can run it on new examples, track performance over time, and identify areas where your AI system needs work. However, you probably have questions on how to deploy these metrics in a production setup. Should you be running them on all your production inputs and outputs or just samples? Are these metrics enough or should you keep reinventing? We'll talk about all this in the next chapter.

In the next chapter, we'll explore how to use these metrics in an improvement loop that helps your system get better over time.

Chapter 6: Production Deployment and Real User Behavior

From Lab to Real World

So far in this course, we've covered the essential building blocks of AI evaluation. We started by understanding why evaluation matters for AI systems and distinguished between model evaluations and product evaluations. We explored the conceptual foundation of input, expected, and actual behavior. We walked through building reference datasets to systematically identify what matters for your specific use case. And we covered three approaches to implementing evaluation metrics: human evaluation, code-based metrics, and LLM judges.

At this point, you have a solid evaluation framework. You've built reference datasets that represent important scenarios for your system. You've identified the key metrics that track behaviors you actually care about. You've implemented ways to measure those behaviors, whether through human judgment, deterministic code checks, or calibrated LLM judges.

But here's where things get interesting and more complex.

Everything we've discussed so far happens in controlled conditions. You're testing with carefully chosen examples, evaluating against clear expected behaviors, and working with stakeholders who understand your system's goals. You're essentially working in a lab environment where you control the inputs and can predict most of the scenarios.

Production is different. When real users start interacting with your AI system, several things happen that change the evaluation game entirely.

The Reality of Real Users

Real users don't behave like your reference datasets. They don't ask questions the way you expect, they don't provide complete information, and they often try to use your system for purposes you never intended.

Users bring unexpected context: Your customer service AI might be designed for product questions, but users will ask about competitor products, share personal stories, or try to use it for technical support issues outside your scope.

Users test edge cases you missed: No matter how thorough your reference dataset, real users will find scenarios you didn't anticipate. They'll phrase requests in ways that confuse your system, combine multiple intents in a single message, or operate under assumptions that don't match your business model.

User evolution: As users get comfortable with your system, their behavior evolves. They develop new ways to phrase requests, discover shortcuts, and use your system in increasingly sophisticated ways. Think about how people use ChatGPT today compared to when it first launched - the questions become more complex, the use cases expand, and the expectations change. This natural evolution means the distribution of inputs your system receives will shift over time.

Volume changes everything: When you test with 50 carefully chosen examples, you can review each interaction manually. When your system handles 10,000 interactions per day, you need fundamentally different approaches to understanding what's happening.

The Scale Challenge

In controlled testing, you can review every example and understand every failure. In production, this becomes impossible.

Consider a customer support AI that handles 5,000 conversations daily. Even if 95% of interactions go perfectly, you still have 250 potentially problematic conversations every day. Manual review of each one would require dedicated staff just for evaluation.

The challenge isn't just volume - it's also about detection. In your reference dataset, you know which examples should pass or fail your evaluation metrics. In production, you don't know ahead of time which conversations will be problematic.

This shifts the evaluation question from "How did we do on this specific set of examples?" to "How are we doing overall, and where should we focus our attention?"

From Evaluation to Monitoring

Moving to production fundamentally changes your relationship with evaluation. During development, evaluation was about validation (testing whether your system works as intended). In production, evaluation becomes monitoring (continuously checking whether your system continues to work well as conditions change).

This affects how you think about measurement, response, and improvement:

Evaluation builds confidence before deployment: You test thoroughly to gain confidence that your system is ready for users.

Monitoring maintains quality during deployment: You track performance to catch problems early and guide improvements.

The flywheel of improvement: Good production monitoring feeds back into your evaluation process. Issues discovered in production become new test cases in your reference datasets. Patterns identified in monitoring inform better pre-deployment validation. The two work together in a continuous improvement cycle.

This creates a natural progression: strong evaluation gives you confidence to deploy, effective monitoring helps you improve, and improved systems perform better in evaluation.

Four Core Challenges in Production

When you move from controlled evaluation to production monitoring, four key challenges emerge that require careful planning:

1. Log Filtering

With thousands of events happening daily, you can't manually review everything. You need systematic approaches to identify which logs deserve attention. This means developing filtering and sampling strategies that help you focus on the data most likely to reveal problems or insights.

2. Metric Selection

Remember that evaluation metrics aren't free. LLM judges cost money to run, human evaluation requires time and expertise, and even code-based metrics might not always be as trivial or cheap as running unit tests in traditional software setups. At scale, these costs add up quickly. You need to be strategic about which metrics provide the most valuable insights relative to their cost.

3. Online vs. Offline Evaluation

This is where we introduce an important distinction that will shape your production monitoring strategy:

Online evaluation happens in real-time as users interact with your system. These metrics run immediately and can trigger alerts or interventions. For example, you might have an online safety filter that flags inappropriate content before it reaches users.

Offline evaluation happens after the fact, often in batch processes. These metrics analyze interactions that already occurred to identify trends, assess quality over time, or conduct detailed investigations. For example, you might run expensive LLM judges overnight to assess the previous day's customer service interactions.

The choice between online and offline evaluation affects cost, complexity, and responsiveness. Online evaluation gives you immediate feedback but needs to be fast and lightweight. Offline evaluation can be more thorough and sophisticated but only helps you improve future interactions.

4. Emerging Issue Discovery

Despite doing all of this systematically, it's possible that we have not anticipated some issues at all. What do we do about that?

Even the most thorough offline evaluation process can't predict every problem that will emerge in production. Users will find new ways to confuse your system, edge cases you never considered will surface, and changing business requirements will create new failure modes.

This means you need strategies for discovering issues that your existing evaluation framework doesn't catch. How do you identify problems you weren't looking for? How do you evolve your evaluation approach as new patterns emerge?

These four challenges form the foundation of production monitoring strategy. Getting them right determines whether your monitoring system provides actionable insights or becomes an expensive distraction.

What Comes Next

The transition from controlled evaluation to production monitoring requires addressing these four core challenges systematically. The goal isn't to replicate your reference dataset evaluation at production scale (that would be impractical and expensive). Instead, you need smart strategies for each challenge.

In the next chapter, we'll cover practical approaches to:

• **Log filtering**: Strategies for identifying which data needs attention without drowning in information

• **Metric selection**: Frameworks for choosing the right mix of evaluation approaches based on value and cost

• **Online vs offline evaluation**: Designing systems that balance immediate responsiveness with thorough analysis

• **Emerging issue discovery**: Methods for identifying problems that your existing evaluation framework doesn't catch

These approaches will help you build a monitoring system that provides actionable insights while remaining sustainable and cost-effective as your AI system scales.

Chapter 7: Production Monitoring Strategies

From Challenges to Solutions

In the previous chapter, we identified four core challenges that emerge when you move from controlled evaluation to production monitoring:

1. **Log filtering**: How to identify which data deserves attention

2. **Metric selection**: How to choose the right evaluation approaches

3. **Online vs offline evaluation**: How to balance real-time needs with thorough analysis

4. **Emerging issue discovery**: How to find problems you weren't looking for

Now we'll address each challenge with practical strategies you can implement. The goal is building a sustainable monitoring system that provides actionable insights without overwhelming your team or budget.

Log Filtering: Finding Signal in the Noise

When your AI system handles thousands of events daily, you need systematic approaches to identify what requires attention. Random sampling might miss critical issues, while trying to review everything is impossible.

Priority-Based Filtering

Start by defining what matters most for your specific business context. Not all events are equally important, and what deserves attention varies significantly based on your use case and risk tolerance.

For example, you might consider categorizing events like this:

Potential high-priority signals could include safety violations, system errors, or high-value interactions - but you need to define what "high-value" means for your business.

Potential medium-priority signals might be routine interactions that show unusual patterns - though you'll need to determine what constitutes "unusual" in your domain.

Potential low-priority signals could be simple, standard interactions - but again, "simple" and "standard" depend entirely on your system's purpose and user base.

Signal-Based Sampling

Beyond basic priority filtering, you can look for implicit and explicit signals that users give you about interaction quality. You need to identify which signals matter most for your specific system and users.

Some examples of signals you might consider:

Conversation patterns like unusual length (much shorter or longer than typical), repetition (users rephrasing questions), explicit escalation requests, or confusion indicators. But what counts as "unusual" length depends entirely on your domain - a financial advisory conversation naturally runs longer than a weather query.

User behavior patterns such as extensive editing of generated content, retry behavior, frustration indicators, or abandonment patterns. For a content generation system, whether users copy-paste or heavily modify outputs tells you something about quality - but you need to decide what level of modification indicates a problem versus normal customization.

Content quality indicators including response completeness, format consistency, or context matching. The thresholds that matter depend on your system's purpose and user expectations.

The critical decision is determining which of these signals are most indicative of problems in your specific context.

Example: Customer Support Filtering Considerations

A customer support AI team might consider various approaches, but the specific choices depend on their business priorities and risk tolerance:

They might choose to always examine interactions with explicit escalation requests or safety concerns, but the definition of "safety concern" varies by industry. They could focus on conversations mentioning competitors or billing disputes, but whether a $50 or $500 dispute deserves attention depends on their business model.

They might sample more heavily from unusually long conversations, but "unusual" for a simple password reset differs from "unusual" for a complex technical issue. They could prioritize first-time users or interactions that show signs of confusion, but the thresholds that matter depend on their user base and system design.

Signal Considerations for Different AI Systems

Content Generation AI teams might care about extensive user editing of outputs, but they need to decide whether 50% editing indicates a problem or normal creative refinement.

Financial Advisory AI teams might monitor for repeated clarification requests, but they must determine whether two follow-ups indicate confusion or appropriate due diligence.

E-commerce Recommendation AI teams might track ignored recommendations, but they need to consider whether this indicates poor recommendations or users with specific preferences.

In each case, the team must define their own thresholds and priorities based on their specific context, users, and business goals.

Dynamic Filtering Based on Production Signals

Your filtering strategy should adapt based on observable changes in your production environment, but you need to decide which signals matter most for your business.

Consider increasing sampling when you observe production changes like error rate spikes, new product launches that might confuse users, increases in human support tickets, shifts in user behavior patterns, seasonal events that change user needs, or marketing campaigns that influence how users phrase requests.

Consider decreasing sampling when you see stable performance metrics, mature interaction patterns, or stable behavior in specific system components - though you must balance this against resource constraints and the risk of missing emerging issues.

Examples of production signals you might track include support ticket volume and categories, user session abandonment rates, conversation length trends, system performance metrics, business metrics like conversion rates, and external events like product launches or competitor actions.

The key decisions are which signals to monitor, what changes are significant enough to trigger sampling adjustments, and how quickly to respond to different types of changes. These choices depend entirely on your business context, user base, and risk tolerance.

Metric Selection: Choosing Your Evaluation Mix

Not all metrics are equally valuable, and running everything is expensive. You need frameworks for choosing the right mix of evaluation approaches.

The Metric Value Framework

Evaluate each potential metric across three dimensions:

Impact: How much does this metric help you improve your system?

• High impact: Metrics that reveal actionable problems

• Medium impact: Metrics that provide useful trends

• Low impact: Metrics that are interesting but don't drive decisions

Reliability: How consistent and accurate is this metric?

• High reliability: Human expert evaluation, well-validated code checks

• Medium reliability: Calibrated LLM judges, statistical measures

• Low reliability: Uncalibrated automated assessments, proxy metrics

Cost: What does it cost to run this metric at scale?

• Low cost: Simple code-based checks, existing system metrics

• Medium cost: Fast LLM judge calls, periodic human spot-checks

• High cost: Detailed human evaluation, expensive model calls, complex analysis

Prioritization Matrix

Plot your potential metrics on a simple matrix:

High Impact + Low Cost = Must Have

• Simple safety filters

• Basic structure validation

• Performance metrics (response time, success rate)

• Clear policy violation detection

High Impact + High Cost = Strategic Investment

• Calibrated LLM judges for subjective quality

• Expert human evaluation for critical interactions

• Detailed escalation accuracy assessment

Low Impact + Low Cost = Nice to Have

• Basic statistical trends

• Simple response length tracking

• Automated sentiment detection

Low Impact + High Cost = Avoid

• Elaborate scoring systems that don't drive decisions

• Expensive metrics that duplicate existing insights

• Over-detailed measurement of stable system behaviors

Online vs Offline Evaluation: Guardrails vs Improvement Flywheel

The choice between real-time and batch evaluation comes down to a fundamental question: What behaviors, if they go wrong, would be huge for your business?

Online Evaluation: Business-Critical Guardrails

Online evaluation serves as guardrails - metrics that must run in real-time because the behaviors they monitor are so critical that failure would significantly impact your business.

These are metrics where you need immediate intervention, not just later analysis. When these guardrails trigger, your system should take immediate action like handing off to a human agent, blocking harmful content, or escalating to specialists.

Think of guardrails for behaviors like:

• Safety violations that could harm users or your business

• Compliance failures that could create legal liability

• High-value customer situations that require immediate attention

• System failures that impact user experience

• Critical business rule violations

Guardrail characteristics:

• Must be fast and reliable (failures cascade quickly)

• Should trigger immediate actions (handoffs, blocks, escalations)

• Focus on preventing catastrophic outcomes, not optimization

• Need to work even when other systems are stressed

Examples of potential guardrail metrics:

• Safety filters blocking harmful content before it reaches users

• Compliance checks ensuring required disclaimers in financial advice

• Uncertainty detection triggering immediate human handoff

• High-value customer detection routing to premium support

• System error detection triggering failover procedures

Offline Evaluation: Improvement Flywheel

Offline evaluation powers your improvement flywheel - analyzing data after the fact to understand trends, assess quality, and guide system improvements.

These metrics help you get better over time rather than preventing immediate disasters. They're often more sophisticated, expensive, or time-consuming than guardrails, but they provide the insights needed to evolve your system.

Offline evaluation focuses on:

• Understanding quality trends over time

• Identifying patterns that inform system improvements

• Conducting detailed analysis of complex behaviors

• Assessing the effectiveness of your guardrails and other systems

• Discovering opportunities for optimization

Examples of potential offline metrics:

• LLM judge assessment of conversation quality trends

• Human expert review of escalated cases to improve escalation logic

• Analysis of user satisfaction patterns to guide product development

• Evaluation of edge cases to expand training data

• Assessment of guardrail effectiveness and calibration

Making the Guardrail Decision

The key decision is identifying which behaviors are guardrail-worthy - meaning failure would have immediate, significant business impact.

For a healthcare AI, incorrect medication information might be a guardrail issue requiring immediate intervention. For an e-commerce chatbot, product recommendation accuracy might be important for improvement but not guardrail-critical.

For a financial advisory AI, compliance violations are clearly guardrail territory, while response tone optimization belongs in the improvement flywheel.

The cost and complexity of guardrails mean you should be selective about what requires real-time intervention versus what can wait for batch analysis and gradual improvement.

Emerging Issue Discovery: When Your Signals Don't Match Your Metrics

Remember the log filtering approach we discussed earlier - you're already sampling based on implicit and explicit user signals like conversation length anomalies, retry behavior, editing patterns, and frustration indicators. But what happens when these signals are telling you something your current metrics aren't capturing?

This is where emerging issue discovery becomes critical. You might find that your existing evaluation metrics show everything is working well, but the user behavior signals you're sampling suggest otherwise.

When Signals and Metrics Diverge

Consider this scenario: You're monitoring a content generation AI, and you've been sampling interactions where users heavily edit the generated outputs (one of your implicit signals). Your current metrics - like content relevance and grammar correctness - show these interactions are scoring well. But the signal persists: users keep making extensive edits.

This divergence suggests there might be a quality dimension you're not measuring. Perhaps users are editing for tone, brand voice, or subtle contextual appropriateness that your current metrics don't capture. The user behavior signal is revealing a hidden issue that your evaluation framework missed.

Systematic Investigation of Signal-Metric Gaps

When you notice this pattern - where your sampling signals flag interactions but your metrics show no actionable improvements - it's time for manual investigation, which means you'll need to look at these traces manually, just like we did initially when building reference datasets:

Analyze the filtered logs differently: Instead of applying your existing metrics, look at the interactions your signals flagged with fresh eyes. What patterns do you see that your metrics might be missing?

Qualitative review: Have domain experts or users review the flagged interactions without knowing the metric scores. What do they notice that your metrics don't capture?

Signal correlation analysis: Look at which combinations of signals tend to appear together. Multiple signals pointing to the same interactions might indicate a systematic issue.

Example: E-commerce Recommendation Discovery

An e-commerce AI notices high rates of users ignoring recommendations (a signal they're sampling). But their existing metrics show the recommendations are relevant and properly formatted. Investigation reveals users are ignoring recommendations during certain seasonal periods or for specific product categories - suggesting the system lacks awareness of temporal context or category-specific preferences that existing relevance metrics don't measure.

Building New Metrics from Signal Patterns

When signal-metric divergence reveals hidden issues, you need to develop new evaluation approaches:

Pattern documentation: Systematically document what the expert review reveals about the flagged interactions.

New metric development: Create evaluation approaches that can capture the quality dimensions you discovered.

Validation against signals: Test whether your new metrics correlate with the user behavior signals that originally flagged the issue.

Integration into your framework: Add the new metrics to your offline evaluation for trend monitoring, and consider whether any need to become online guardrails.

The Discovery Loop

This creates a continuous discovery loop:

1. **User signals** indicate potential issues through behavior patterns

2. **Log filtering** samples these concerning interactions

3. **Metric analysis** may show existing metrics aren't capturing the problem

4. **Investigation** reveals hidden quality dimensions or failure modes

5. **New metrics** are developed to monitor these newly discovered issues

6. **Updated sampling** incorporates lessons learned to catch similar issues earlier

This loop ensures your evaluation framework evolves as you discover new ways your system can fail or as user expectations change over time.

The key insight is that user behavior signals often reveal problems before your metrics do - they're an early warning system that helps you discover evaluation gaps before they become major issues.

Building Your Production Monitoring Strategy

Combining these four strategies creates a comprehensive production monitoring approach:

Start Simple and Evolve

Begin with basic filtering, essential metrics, simple online checks, and manual discovery processes. Add complexity as you understand your system's behavior patterns and your team's capacity.

Balance Cost and Value

Continuously evaluate whether your monitoring provides enough insight to justify its cost. Expensive evaluation that doesn't drive improvements should be reconsidered.

Plan for Scale

Design your monitoring to grow with your system. Approaches that work for thousands of daily interactions need to adapt when you reach hundreds of thousands.

Close the Feedback Loop

The goal of monitoring is improvement. Ensure that insights from your monitoring system feed back into better evaluation, system refinements, and updated business processes.

The Complete Evaluation Journey: From Concepts to Production

We've now covered the full spectrum of AI evaluation - from understanding why evaluation matters (Chapter 1) to building systematic evaluation frameworks (Chapters 2-3), creating reference datasets and implementing metrics (Chapters 4-5), and finally deploying robust production monitoring (Chapters 6-7).

The key insight is that evaluation is never complete: you start by building evaluation for anticipated behaviors and failure modes, but real users will always find new ways to interact with your system that you haven't seen before. This is why production monitoring becomes a continuous cycle of discovering new patterns through user signals, manually investigating when your current metrics don't capture emerging issues, developing new evaluation approaches, and feeding these insights back into your evaluation framework.

Think of it as building evaluation for the patterns you can anticipate, then using monitoring to discover and evaluate the patterns you couldn't predict.

In the next chapter, we'll explore how to use these monitoring insights to create continuous improvement cycles that help your AI system get better over time.

Chapter 8: The Complete Evaluation Process

From Concept to Production: Your Step-by-Step Guide

In the previous seven chapters, we've covered the complete landscape of AI evaluation - from understanding why it matters to deploying production monitoring systems. Now let's consolidate everything into a clear, step-by-step process you can follow to build robust evaluation for your AI system.

This chapter serves as your practical roadmap, connecting all the concepts we've discussed into actionable steps you can implement.

The Two-Phase Approach

AI evaluation follows two distinct phases:

Phase 1: Pre-Deployment Validation (Chapters 1-5)

• Build confidence that your system works as intended before users interact with it

• Create systematic evaluation frameworks and metrics

• Test thoroughly in controlled conditions

Phase 2: Production Monitoring (Chapters 6-7)

• Monitor system performance with real users at scale

• Discover new issues and evolving user behaviors

• Continuously improve your system and evaluation approach

Here's how to work through each phase.

Phase 1: Pre-Deployment Validation

Step 1: Understand Your Evaluation Context

Based on Chapters 1-3

What you're doing: Establish the foundation for your evaluation approach by understanding what makes AI evaluation unique and what you need to measure.

Key decisions: Recognize that your AI system is non-deterministic, focus on product evaluation (how your system behaves in your specific use case) rather than model evaluation, and identify the three components you're evaluating - Input, Expected, and Actual.

What to do: Start by mapping out your specific use case and domain requirements. Identify stakeholders who need to be involved - domain experts, product teams, and engineers. Remember that generic metrics like "helpfulness" mean different things in different contexts, so prepare for collaborative evaluation design across different team perspectives.

Output: Clear understanding that you're building evaluation for your specific context, not just testing general AI capabilities.

Step 2: Build Your Reference Dataset

Based on Chapter 4

What you're doing: Create a systematic collection of examples that represent the scenarios you care about most, with clear expectations for how your system should behave.

Key decisions:

• Start small and specific (10-20 high-quality examples) rather than trying to be comprehensive

• Focus on scenarios you absolutely cannot get wrong

• Include realistic inputs that represent actual user behavior

Action items:

1. **Generate initial examples**: Work with domain experts to create realistic scenarios based on historical data or domain knowledge

2. **Run your system**: Test your AI system on these examples and document both outputs and any intermediate steps

3. **Evaluate with experts**: Have domain experts review each example and answer "Was this response satisfactory? If not, why not?"

4. **Identify error patterns**: Analyze failures to cluster them into underlying problems you can actually fix

5. **Decide on ongoing metrics**: Determine which behaviors need continuous monitoring (recurring risks) versus one-time fixes

Output: A reference dataset with examples, system outputs, expert evaluations, and identified metrics for ongoing measurement.

Step 3: Implement Your Evaluation Metrics

Based on Chapter 5

What you're doing: Build the actual measurement systems that can assess your identified metrics using three possible approaches.

Key decisions:

• Choose the right mix of human evaluation, code-based metrics, and LLM judges

• Start simple and add complexity only when needed

• Remember that LLM judges require careful calibration against human judgment

Action items:

1. **For objective, measurable properties**: Implement code-based metrics (structure validation, performance checks, required content)

2. **For subjective qualities**: Consider LLM judges with detailed rubrics and examples

3. **For critical quality assessment**: Plan for human evaluation, at least for calibration and spot-checking

4. **Build rubrics**: Create clear criteria defining acceptable vs. not acceptable performance with specific examples

5. **Test your metrics**: Validate that your evaluation approaches actually catch the issues you care about

6. **Calibrate LLM judges**: If using them, extensively test against human judgment and iteratively refine

Output: Implemented evaluation metrics that can reliably assess the behaviors you identified in Step 2.

Phase 2: Production Monitoring

Step 4: Deploy Smart Log Filtering

Based on Chapter 7 - Log Filtering

What you're doing: Create systematic approaches to identify which production data deserves attention, since you can't manually review everything at scale.

Key decisions:

• Define what matters most for your business context (high/medium/low priority events)

• Choose which implicit and explicit user signals to monitor

• Set up dynamic filtering that adapts to production changes

Action items:

1. **Establish priority categories**: Define which events always need attention vs. which can be sampled

2. **Identify user signals**: Look for patterns like unusual conversation length, retry behavior, editing patterns, frustration indicators

3. **Set up signal-based sampling**: Sample more heavily from interactions showing concerning signals

4. **Monitor production changes**: Increase sampling during new product launches, error rate spikes, or business requirement changes

5. **Adapt over time**: Adjust your filtering strategy based on what you learn

Output: A filtering system that efficiently identifies the most important production data to examine.

Step 5: Select and Deploy Your Production Metrics

Based on Chapter 7 - Metric Selection

What you're doing: Choose which evaluation metrics to run in production based on their impact, reliability, and cost.

Key decisions:

• Prioritize high-impact metrics that drive actionable improvements

• Balance metric value against computational and financial costs

• Focus resources on metrics that actually help you make better decisions

Action items:

1. **Evaluate each metric**: Assess impact (how much it helps improve your system), reliability (how consistent it is), and cost (computational/financial expense)

2. **Prioritize systematically**: Focus on high-impact, low-cost metrics first; carefully consider high-impact, high-cost metrics; avoid low-impact approaches regardless of cost

3. **Start essential**: Implement must-have metrics that provide basic system health and safety monitoring

4. **Add strategically**: Gradually incorporate more sophisticated metrics based on demonstrated value

Output: A cost-effective mix of evaluation metrics running in production.

Step 6: Implement Guardrails and Improvement Loops

Based on Chapter 7 - Online vs Offline Evaluation

What you're doing: Distinguish between metrics that need immediate intervention (guardrails) versus those that guide longer-term improvement.

Key decisions:

• Identify which behaviors, if they go wrong, would be huge for your business (guardrails)

• Design offline evaluation for trend analysis and system improvement

• Balance real-time intervention needs with batch analysis efficiency

Action items:

1. **Design guardrails**: Implement fast, reliable online metrics for business-critical behaviors that trigger immediate actions (handoffs, escalations, blocks)

2. **Set up improvement loops**: Create offline evaluation processes that analyze trends, assess quality over time, and guide system improvements

3. **Define trigger actions**: Establish clear procedures for what happens when guardrails activate

4. **Plan feedback cycles**: Ensure offline analysis insights feed back into system improvements and evaluation refinements

Output: A two-tier system with real-time guardrails for critical issues and batch analysis for continuous improvement.

Step 7: Build Emerging Issue Discovery

Based on Chapter 7 - Emerging Issue Discovery

What you're doing: Create processes to discover problems your existing evaluation framework doesn't capture, using the same manual investigation techniques from reference dataset building.

Key decisions:

• Recognize that user signals often reveal problems before metrics do

• Plan for manual investigation when signals and metrics diverge

• Build systematic processes to evolve your evaluation framework over time

Action items:

1. **Monitor signal-metric divergence**: Watch for cases where user behavior signals flag issues but your metrics show no problems

2. **Conduct manual investigation**: When divergence occurs, manually review the flagged interactions just like you did when building reference datasets

3. **Identify hidden issues**: Look for quality dimensions or failure modes your current metrics don't capture

4. **Develop new metrics**: Create evaluation approaches for newly discovered issues

5. **Update your framework**: Add new metrics to your evaluation system and refine your filtering approach

6. **Close the discovery loop**: Ensure insights from investigation feed back into better evaluation and system improvements

Output: A continuously evolving evaluation framework that adapts as you discover new issues and user behaviors.

The Complete Process Flow

Here's how all these steps connect:

1. **Foundation** → Understand your specific evaluation needs and context

2. **Reference Dataset** → Build systematic examples with clear quality expectations

3. **Metrics Implementation** → Create reliable measurement systems for your quality criteria

4. **Production Filtering** → Efficiently identify important production data to examine

5. **Metric Deployment** → Run cost-effective evaluation at scale

6. **Guardrails + Improvement** → Handle critical issues immediately while building long-term improvement

7. **Discovery Loop** → Continuously evolve your evaluation as you learn new failure modes

Key Principles Throughout

Start Simple: Begin with basic approaches and add complexity only when justified by clear value.

Focus on Context: Generic evaluation approaches don't work - everything must be tailored to your specific use case, users, and business requirements.

Collaborate Across Teams: Effective evaluation requires input from domain experts, product teams, and engineers working together.

Embrace Evolution: Your evaluation framework should continuously improve as you discover new ways your system can fail or as user expectations change.

Connect Evaluation to Improvement: The goal is better AI systems, not perfect measurement. Focus on evaluation that drives actionable improvements.

What You End Up With

Following this complete process gives you:

• **Confidence before deployment**: Systematic validation that your system works as intended

• **Effective production monitoring**: Smart filtering and evaluation that scales with your system

• **Proactive issue detection**: Early warning systems that catch problems before they become major issues

• **Continuous improvement**: Feedback loops that help your system get better over time

• **Sustainable evaluation**: Cost-effective approaches that provide value without overwhelming your team

The Ongoing Journey

Remember that evaluation is never complete. You start by building evaluation for patterns you can anticipate, then use production monitoring to discover and evaluate patterns you couldn't predict. User behavior evolves, business requirements change, and new failure modes emerge.

The framework we've built gives you the tools to adapt your evaluation approach as your understanding deepens and your system grows. The key is maintaining the discipline of systematic evaluation while staying flexible enough to learn and evolve.

This complete process transforms evaluation from an afterthought into a core capability that helps you build more reliable, useful, and trustworthy AI systems.

Chapter 9: Common Misconceptions About AI Evaluation

Clearing Up the Confusion

Now that you've worked through this complete evaluation course, you're equipped to recognize common misconceptions that trip up many teams building AI systems. This chapter addresses the most frequent misunderstandings we encounter, explaining why they're problematic and pointing you to the right approaches.

Each misconception below includes a reference to the chapters where we covered the correct approach in detail.

Foundation Misconceptions

1. "Model evaluations (benchmarks) predict my product success"

Why this is wrong: Model evaluations test general capabilities on standardized tasks, but your product operates in a specific domain with unique requirements, constraints, and user behaviors. A model that scores 92% on general benchmarks might perform poorly for your insurance claims processing system if it hasn't seen domain-specific patterns.

The reality: Product evaluation in your specific context is what matters. You need to test how the model behaves with your data, your users, your business rules, and your risk tolerance.

Where we covered this: Chapter 2 explains the crucial distinction between model and product evaluations, showing why benchmark performance often fails to predict real-world success in your specific use case.

2. "Engineers can design evaluation metrics alone"

Why this is wrong: Engineers understand technical implementation but may miss domain-specific quality requirements, business risks, and subtle user expectations. What looks technically correct might be completely inappropriate for the domain.

The reality: Effective evaluation requires collaboration between domain experts (who understand quality), product teams (who understand user needs), and engineers (who understand technical constraints). Each brings essential perspectives.

Where we covered this: Chapter 3 emphasizes that evaluation is inherently collaborative and explains how different stakeholders contribute to defining quality standards and building rubrics.

3. "Evaluation is a one-time setup before launch"

Why this is wrong: This treats evaluation like traditional software testing, where you can validate everything upfront and expect it to stay valid. AI systems are non-deterministic, user behavior evolves, and business requirements change.

The reality: Evaluation is a continuous process that evolves with your system. You start with pre-deployment validation, then monitor in production, discover new issues, and continuously refine your evaluation approach.

Where we covered this: Chapter 1 explains why AI systems require ongoing evaluation, and Chapter 6 details how production monitoring differs from pre-deployment testing.

Pre-Deployment Misconceptions

4. "I need comprehensive evaluation coverage from day one"

Why this is wrong: Trying to build comprehensive evaluation upfront leads to analysis paralysis and often misses the most important issues. You can't predict every failure mode, and attempting comprehensive coverage dilutes effort from high-impact scenarios.

The reality: Start small with 10-20 high-quality examples representing scenarios you absolutely cannot get wrong. Focus on quality over quantity and expand as you learn more about your system's behavior patterns.

Where we covered this: Chapter 4 walks through building reference datasets, emphasizing starting small and specific rather than trying to be comprehensive from the beginning.

5. "Code-based metrics aren't sophisticated enough for AI systems"

Why this is wrong: This assumes you need complex evaluation for complex systems. In practice, simple code-based checks often provide the most reliable signal for many important behaviors like structure validation, compliance requirements, and performance monitoring.

The reality: Simple code checks are fast, reliable, and easy to understand. Use them for objective, measurable properties before adding complexity with LLM judges or human evaluation.

Where we covered this: Chapter 5 details the three evaluation approaches, showing when code-based metrics are most effective and why they should often be your first choice.

6. "LLM judges are the best way to evaluate AI systems"

Why this is wrong: LLM judges seem appealing because they can assess subjective qualities at scale, but they're expensive, slow, and can be inconsistent or misaligned with human judgment. Uncalibrated LLM judges often create more problems than they solve.

The reality: LLM judges are powerful tools when properly calibrated, but they require extensive validation against human judgment. Start with simpler approaches and add LLM judges only when justified by clear value.

Where we covered this: Chapter 5 explains the challenges with LLM judges and emphasizes that calibration is essential for reliable results.

7. "If I write detailed criteria, LLM judges will work correctly"

Why this is wrong: Detailed criteria help, but don't guarantee that an LLM will interpret them the same way human experts would. LLMs can be overly strict, overly lenient, or miss subtle contextual cues that humans notice.

The reality: LLM judge calibration requires extensive testing against human evaluations across hundreds of examples, statistical analysis of agreement rates, and iterative prompt refinement. This process often takes weeks or months.

Where we covered this: Chapter 5 includes detailed guidance on LLM judge calibration and why detailed criteria alone are insufficient for reliable evaluation.

Production Misconceptions

8. "I need to evaluate every production interaction"

Why this is wrong: At scale, evaluating every interaction is impossible and unnecessary. It would require enormous computational resources and human effort while providing diminishing returns from analyzing routine, successful interactions.

The reality: Smart sampling based on user signals is more effective. Focus evaluation on interactions showing concerning patterns like unusual length, retry behavior, or frustration indicators.

Where we covered this: Chapter 7's log filtering section explains how to identify which production data deserves attention through priority-based filtering and signal-based sampling.

9. "I need a sophisticated dashboard with dozens of metrics"

Why this is wrong: More metrics don't automatically mean better insights. Too many metrics create noise, make it hard to focus on what matters, and often lead to analysis paralysis rather than actionable improvements.

The reality: Focus on a minimum set of actionable metrics that drive real improvements. It's better to have 3-5 metrics that consistently guide decisions than 20 metrics that no one acts on.

Where we covered this: Chapter 7's metric selection section provides frameworks for choosing metrics based on impact, reliability, and cost rather than trying to measure everything.

10. "Online evaluation is always better than offline"

Why this is wrong: Online evaluation seems superior because it provides immediate feedback, but it must be fast and simple to avoid adding latency. Complex analysis that requires expensive computation or sophisticated reasoning belongs in offline evaluation.

The reality: Use online evaluation for business-critical guardrails that need immediate intervention. Use offline evaluation for detailed analysis, trend identification, and system improvement insights.

Where we covered this: Chapter 7 distinguishes between online guardrails (preventing immediate problems) and offline improvement loops (driving long-term system enhancement).

11. "Evals vs A/B testing - I need to pick one approach"

Why this is wrong: This creates a false dichotomy between two complementary approaches. Each serves different purposes and they work better together than in isolation.

The reality: Use evaluation metrics to monitor known patterns and behaviors you understand. Use A/B testing to discover new patterns through explicit user signals (ratings, conversions) and implicit signals (behavior changes, engagement).

Where we covered this: Chapter 7's emerging issue discovery section explains how user signals can reveal problems your evaluation metrics don't capture, leading to new evaluation approaches.

12. "Evaluation metrics are fixed once implemented"

Why this is wrong: This assumes your system, users, and business requirements remain static. In reality, user behavior evolves, business priorities change, and you discover new failure modes that require different evaluation approaches.

The reality: Metrics retire and update over time as you learn. A metric that was critical during early deployment might become less useful as your system matures. Meanwhile, new user behaviors might require entirely new metrics.

Where we covered this: Chapter 7's emerging issue discovery explains the continuous loop of discovering new patterns, developing new metrics, and retiring outdated approaches.

Examples of metric evolution:

• **Retiring**: A "response format validation" metric becomes less important as your system matures and format errors become rare

• **Adding**: A "seasonal context awareness" metric becomes important after discovering users ask different questions during holidays

• **Updating**: An "escalation accuracy" metric needs refinement after business policy changes affect when human handoffs are appropriate

Why These Misconceptions Persist

Understanding why these misconceptions are common helps you avoid them:

AI evaluation is relatively new: Unlike traditional software testing, systematic AI evaluation is still emerging, leading to borrowed assumptions from other domains.

Complexity creates uncertainty: AI systems are complex, making simple approaches seem inadequate even when they're often the most effective.

Tool marketing influences thinking: Vendors promote sophisticated solutions that may be overkill for many practical needs.

Success stories lack context: Case studies often don't include the failures and iterations that led to successful evaluation approaches.

The Right Mindset

Instead of falling into these misconceptions, approach AI evaluation with these principles:

Start simple and evolve: Begin with basic approaches that provide clear value, then add complexity only when justified.

Focus on your context: Generic solutions rarely work - everything must be tailored to your specific use case, users, and business requirements.

Embrace collaboration: Combine technical, domain, and business perspectives rather than trying to solve evaluation in isolation.

Expect continuous evolution: Build evaluation systems that can adapt as you learn more about your system and users.

Prioritize actionable insights: Measure things that drive real improvements rather than pursuing measurement for its own sake.

Moving Forward

Now that you understand these common misconceptions and have worked through the complete evaluation methodology, you're equipped to build effective evaluation systems that avoid these pitfalls.

Remember: the goal isn't perfect measurement - it's building better AI systems through systematic, thoughtful evaluation that evolves with your understanding and needs.

Chapter 10: Glossary of Terms

Making Sense of the Evaluation Vocabulary

Throughout this course, we've used specific terms to describe different aspects of AI evaluation. This glossary clarifies what we mean by each term, helping you navigate the sometimes confusing world of evaluation terminology.

Evals

The catch-all term that everyone uses for everything evaluation-related, which is exactly why it causes so much confusion. Someone might say "we need better evals" and mean anything from benchmark scores to production monitoring dashboards. We intentionally avoid this term in favor of more precise language.

Evaluation

The overall process of assessing how an AI system behaves. This includes everything from designing metrics to running tests to analyzing results. Evaluation answers the question: "Is this system behaving the way we want it to?"

Evaluation Metrics

The specific dimensions along which system behavior is judged. These answer "what does good mean in this context?" Examples include escalation accuracy, response time, or compliance adherence. Always context-dependent and require clear rubrics.

Expected Behavior

What your system should do in a given situation. Part of the Input-Expected-Actual framework. Often requires collaboration between domain experts and product teams to define clearly.

Explicit Signals

Direct indicators users give about their experience, such as ratings, explicit escalation requests ("let me talk to a human"), or direct complaints. Easier to interpret than implicit signals but less common.

Actual Behavior

What your system actually does when given specific inputs. This includes not just the final output, but intermediate steps and any actions taken.

Benchmark

A standardized test used to measure model capabilities across different systems. Examples include MMLU, HumanEval, or GSM8K. Useful for comparing models but don't predict performance in your specific use case.

Code-Based Metrics

Deterministic checks written in programming code that look for specific patterns or properties. Fast, reliable, and perfect for objective measurements like structure validation, required content presence, or performance monitoring.

Guardrails

Real-time evaluation metrics that monitor business-critical behaviors and trigger immediate interventions when problems occur. These are online metrics for situations where failure would have immediate, significant business impact. Examples include safety filters or compliance checks.

Implicit Signals

Indirect indicators of user satisfaction or system problems, revealed through user behavior rather than explicit feedback. Examples include conversation length anomalies, retry behavior, extensive editing of generated content, or abandonment patterns.

Improvement Flywheel

The offline evaluation process that powers long-term system enhancement through trend analysis, quality assessment, and systematic investigation of issues discovered in production.

Input

Everything that influences how your AI system behaves, including the user's request, conversation history, retrieved data, and system configuration. Part of the Input-Expected-Actual evaluation framework.

LLM Judge

Using one language model to evaluate another model's behavior. Powerful for assessing subjective qualities like tone or appropriateness, but requires extensive calibration against human judgment to be reliable.

Log Filtering

Systematic approaches to identify which production data deserves evaluation attention. Uses priority-based filtering and signal-based sampling since you can't review everything at scale.

Model Evaluation

Assessment of general AI model capabilities, typically using standardized benchmarks. Helps with model selection but doesn't predict performance in your specific product context.

Metric Selection

The process of choosing which evaluation approaches to implement based on their impact, reliability, and cost. Requires balancing value against computational and financial expenses.

Non-Deterministic

A key characteristic of AI systems where the same input can produce different outputs across runs. This breaks traditional software testing assumptions and makes evaluation more complex but essential.

Offline Evaluation

Evaluation that happens after interactions occur, often in batch processes. Used for trend analysis, detailed quality assessment, and system improvement insights. Allows for sophisticated, expensive analysis that would be impractical in real-time.

Online Evaluation

Real-time evaluation that runs as interactions happen and can trigger immediate responses. Must be fast and lightweight. Used for guardrails and situations requiring immediate intervention.

Product Evaluation

Assessment of how an AI system behaves in your specific use case, with your users, data, and business context. This is what actually matters for building successful AI products, as opposed to general model capabilities.

Production Monitoring

Continuous evaluation of AI system performance with real users at scale. Includes log filtering, metric deployment, guardrails, and emerging issue discovery.

Reference Dataset

A carefully chosen collection of realistic examples that represent scenarios you care most about. Includes inputs, expected behaviors, and serves as the foundation for systematic evaluation. Start small (10-20 examples) and expand based on learning.

Rubric

Explicit criteria that define what constitutes acceptable versus unacceptable performance. Essential for making subjective evaluation consistent. Should include specific examples and edge case guidance.

Signal-Based Sampling

Sampling production data based on implicit and explicit user signals rather than random selection. More effective for catching problems than uniform sampling across all interactions.

Signal-Metric Divergence

When user behavior signals indicate problems but your current evaluation metrics show no issues. This pattern suggests hidden quality dimensions that your existing evaluation framework doesn't capture.

User Evolution

The natural progression of how users interact with AI systems over time. As users become comfortable, they develop new interaction patterns, push boundaries, and use systems in increasingly sophisticated ways. This changes the distribution of inputs your system receives.

Framework Concepts

Input-Expected-Actual Framework

The conceptual foundation for thinking about AI system behavior:

• **Input**: Everything that goes into your system

• **Expected**: What should happen given your requirements

• **Actual**: What your system really does

This framework helps structure evaluation by making explicit what you're comparing.

Guardrails vs. Improvement Flywheel

The two-tier approach to production evaluation:

• **Guardrails**: Online metrics for immediate intervention on business-critical issues

• **Improvement Flywheel**: Offline analysis for long-term system enhancement

Discovery Loop

The continuous cycle of emerging issue discovery:

1. User signals indicate potential problems

2. Log filtering samples concerning interactions

3. Existing metrics may not capture the issues

4. Manual investigation reveals hidden problems

5. New metrics are developed

6. Updated framework catches similar issues earlier

Process Terms

Pre-Deployment Validation

The systematic evaluation work done before real users interact with your system. Includes building reference datasets, implementing metrics, and testing in controlled conditions to build confidence.

Calibration

The process of ensuring LLM judges align with human judgment through extensive testing, comparison analysis, and iterative refinement. Often takes weeks or months and is essential for reliable automated evaluation.

Emerging Issue Discovery

Systematic approaches to find problems your existing evaluation framework doesn't capture. Uses signal-metric divergence analysis and manual investigation to evolve evaluation as new failure modes emerge.

Common Anti-Patterns (What NOT to Do)

Evaluation Drift

When your evaluation metrics become disconnected from actual user needs or business goals. Happens when you measure things because they're easy to measure rather than because they matter.

Metric Overload

Having too many evaluation metrics, making it impossible to focus on what actually drives improvements. More metrics don't automatically mean better insights.

Calibration Neglect

Deploying LLM judges without proper validation against human judgment, leading to evaluation that's worse than having no evaluation at all.

Coverage Obsession

Trying to evaluate everything comprehensively rather than focusing on high-impact scenarios. Leads to analysis paralysis and diluted effort.

Key Principles

Throughout this course, we've emphasized these core principles:

Context is King: Everything must be tailored to your specific use case, users, and business requirements. Generic approaches rarely work.

Start Simple, Evolve: Begin with basic approaches and add complexity only when justified by clear value.

Collaboration is Essential: Combine technical, domain, and business perspectives rather than trying to solve evaluation in isolation.

Continuous Learning: Evaluation systems must adapt as you discover new failure modes and as user behavior evolves.

Action Over Measurement: The goal is better AI systems, not perfect measurement. Focus on evaluation that drives real improvements.

Using This Glossary

This glossary reflects the specific way we use these terms in this course. You might encounter different definitions elsewhere - the AI evaluation field is still developing standard terminology. When working with others, it's always worth clarifying what specific terms mean in your context.

Remember: the vocabulary matters less than the underlying concepts. Focus on building systematic, thoughtful evaluation that helps you create better AI systems for your users.