EVALUATION FOR LLM APPLICATIONS

TABLE OF CONTENTS

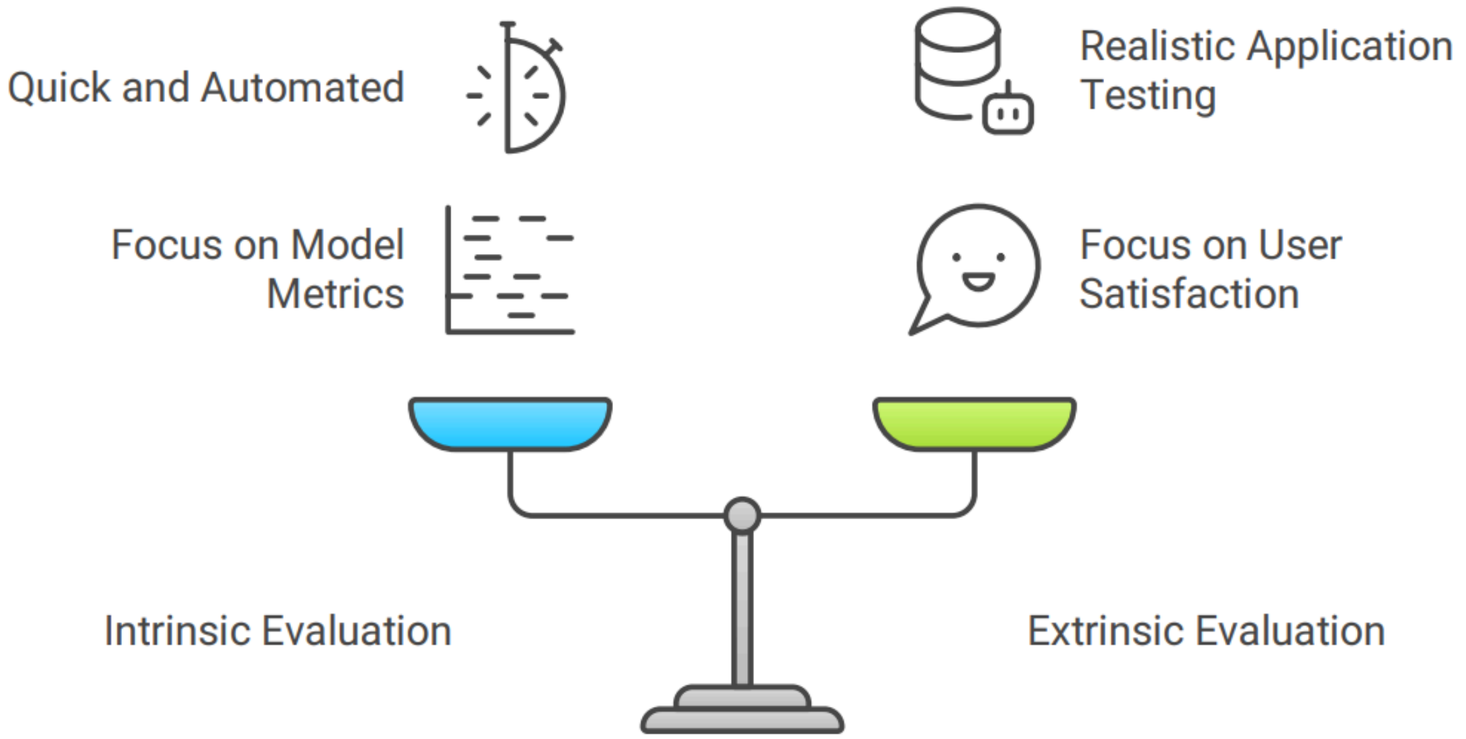
I. Foundations of LLM Evaluation

1. Types of evaluations – intrinsic vs extrinsic

- **Intrinsic Evaluation**
  - Measures the model **in isolation**, without external tasks.
  - Focuses on metrics like *perplexity, accuracy on benchmarks, BLEU scores*.
  - Quick and automated, but may not reflect real-world usefulness.
- **Extrinsic Evaluation**
  - Tests the model **within a specific application or task**.
  - Example: using an LLM in a chatbot → measure *user satisfaction, task success, latency*.
  - More realistic, but harder and more expensive to run.
- **Key Difference**
  - Intrinsic = *"Does the model generate good outputs by itself?"*
  - Extrinsic = *"Does the model help solve the real problem effectively?"*

I. Foundations of LLM Evaluation

1. Types of evaluations – intrinsic vs extrinsic



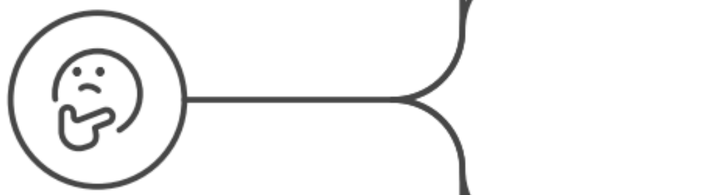
I. Foundations of LLM Evaluation

2. What makes an LLM "good"?

- **Accuracy** – Produces correct and reliable answers
- **Helpfulness** – Provides relevant, useful, and clear information
- **Safety** – Avoids harmful, biased, or inappropriate outputs
- **Latency** – Responds quickly and efficiently

I. Foundations of LLM Evaluation

2. What makes an LLM "good"?



Accuracy

Ensures correct and reliable answers, building trust.

Helpfulness

Provides relevant and useful information, enhancing user experience.

Safety

Avoids harmful or biased outputs, ensuring ethical use.

Latency

Responds quickly, improving efficiency and user satisfaction.

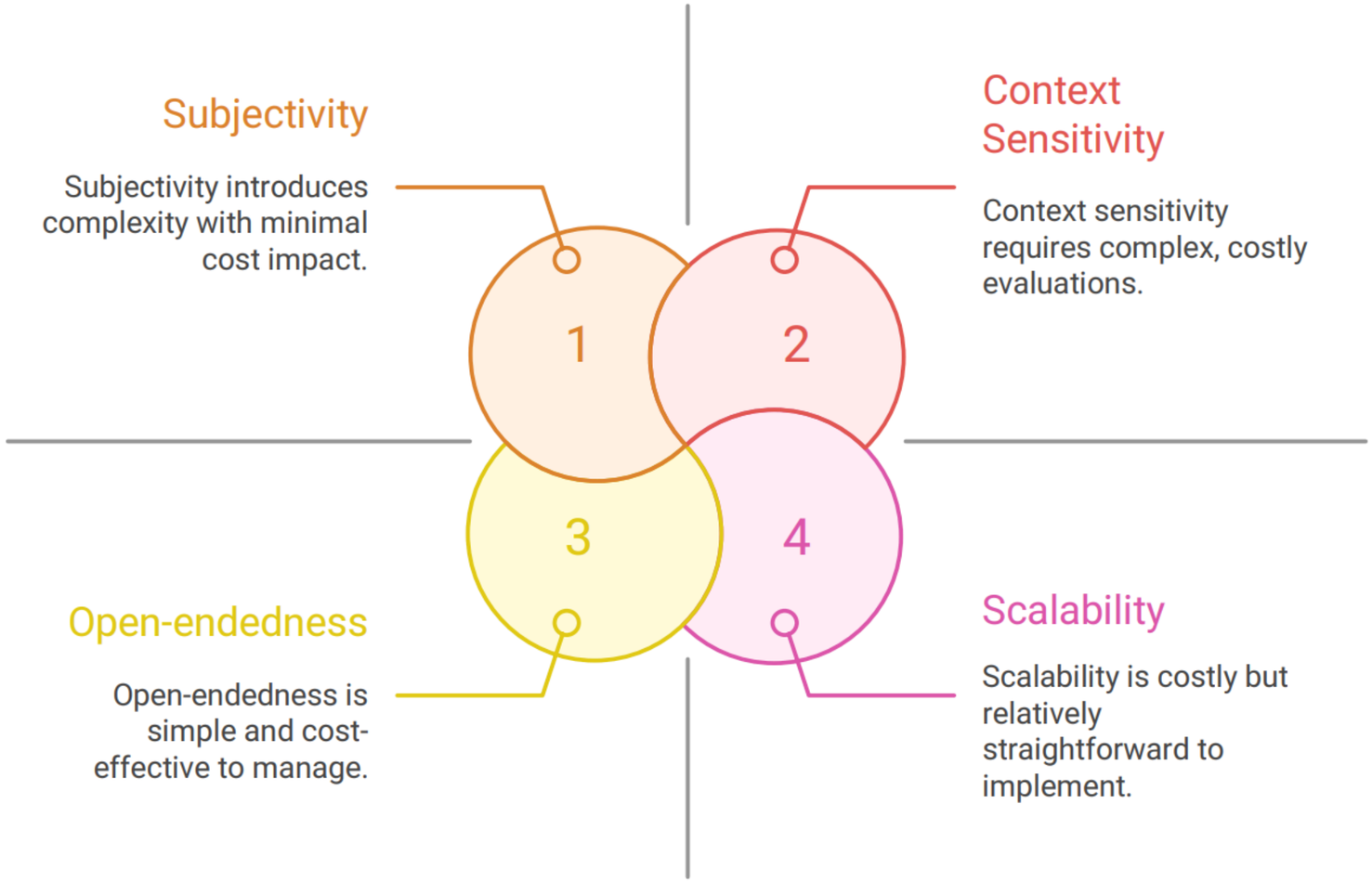
I. Foundations of LLM Evaluation

3. Challenges in evaluating generative outputs

- **Subjectivity** – Different humans may judge answers differently
- **Open-endedness** – Many valid outputs for the same prompt
- **Context sensitivity** – Quality depends on situation or user need
- **Scalability** – Human review is costly and slow

I. Foundations of LLM Evaluation

3. Challenges in evaluating generative outputs



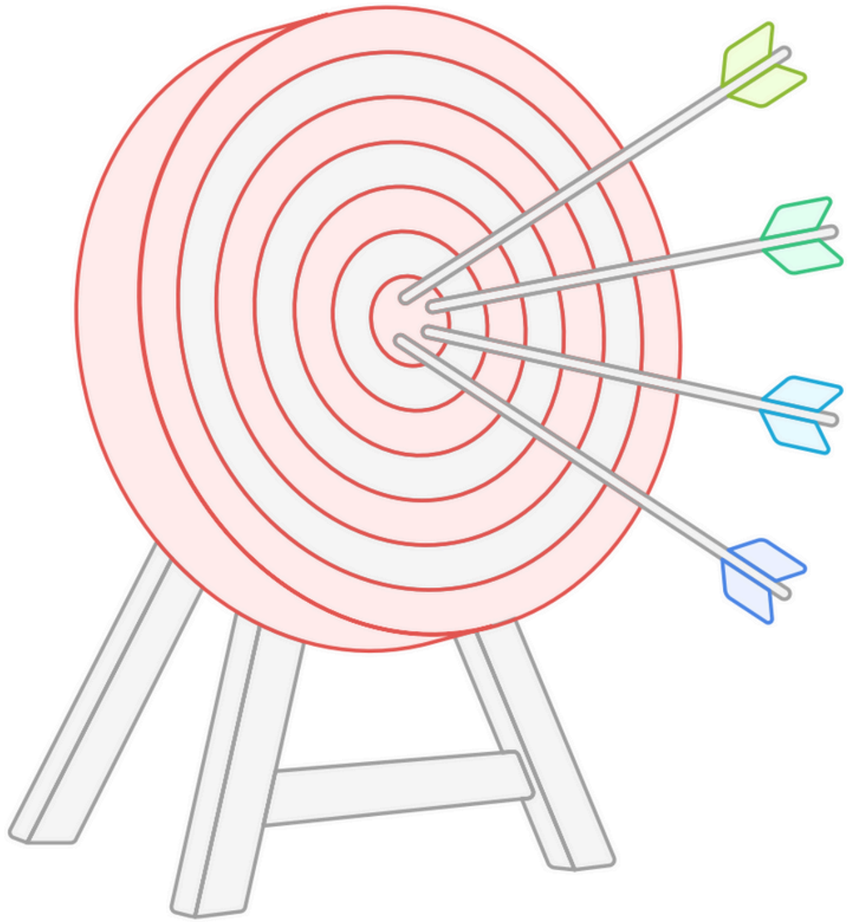
II. Instrumentation & Observability

1. Logging LLM inputs, outputs, and metadata

- **Inputs (What goes in)**
  - User prompts, system instructions, context documents
  - Model parameters: temperature, max tokens, top-k/top-p
  - Purpose: helps reproduce and debug errors
- **Outputs (What comes out)**
  - Full generated text or response
  - Confidence scores or ranking (if available)
  - Purpose: allows review of correctness, tone, safety
- **Metadata (Surrounding context)**
  - Technical: timestamps, latency, token usage, API version
  - User-related: session ID, user role, application context
  - Purpose: connect errors to conditions (e.g., peak traffic, specific users)
- **Overall Goal**
  - Create a **complete trace** of interactions
  - Enable systematic error analysis and continuous improvement

II. Instrumentation & Observability

1. Logging LLM inputs, outputs, and metadata



Complete Trace

Enables error analysis and improvement

Outputs

Generated text and confidence scores

Inputs

User prompts and model parameters

Metadata

Contextual information like timestamps

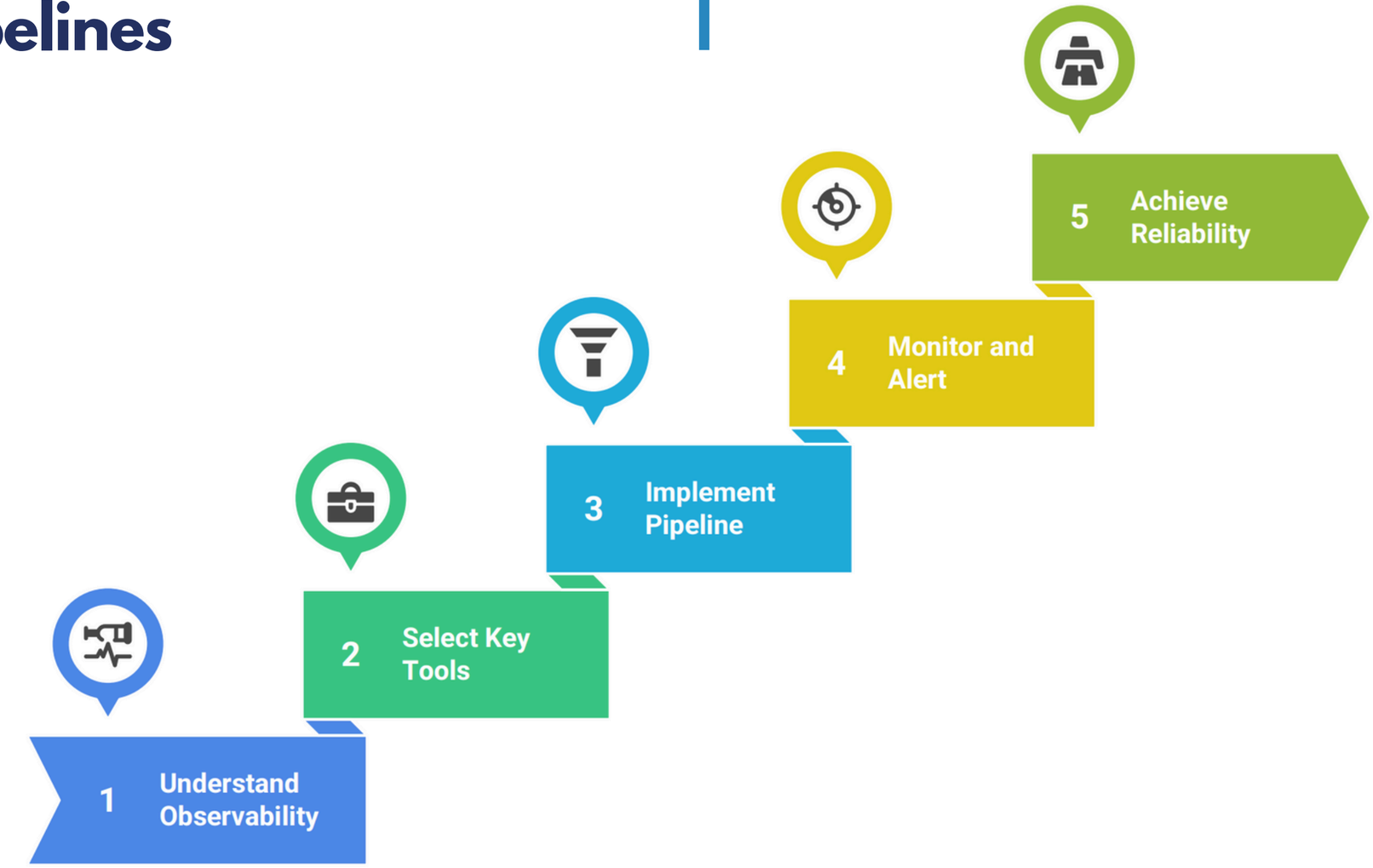
II. Instrumentation & Observability

2. Setting up observability pipelines

- **Why Observability?**
  - Detect errors, performance bottlenecks, safety issues in real time
  - Provide transparency for debugging and optimization
- **Key Tools**
  - **OpenTelemetry** → Collects traces, logs, and metrics from LLM services
  - **Prometheus** → Stores and queries metrics (latency, token usage)
  - **Grafana** → Dashboards for visualization and alerting
- **How It Works**
  - LLM service sends logs and metrics → Observability pipeline → Monitoring dashboard
  - Engineers receive alerts when anomalies occur (e.g., high latency, unusual token spikes)
- **Goal**
  - Ensure LLM systems are **trackable, measurable, and reliable** in production

II. Instrumentation & Observability

2. Setting up observability pipelines



II. Instrumentation & Observability

3. Metrics to track (latency, token usage, satisfaction)

- **Latency (Speed of response)**
  - Measure request → response time
  - Important for chatbots, search, and real-time apps
- **Token Usage (Cost & efficiency)**
  - Count input + output tokens per request
  - Helps monitor expenses and optimize prompts
- **User Satisfaction (Quality & usefulness)**
  - Ratings, feedback surveys, implicit signals (e.g., did user re-ask the same question?)
  - Captures real-world value beyond technical metrics
- **Additional Metrics**
  - Error rates (timeouts, API failures)
  - Safety violations (flagged content, harmful outputs)
- **Goal**
  - Combine **technical metrics** and **human-centric metrics** to measure overall system quality

II. Instrumentation & Observability

3. Metrics to track (latency, token usage, satisfaction)

Which metrics should be tracked to ensure optimal LLM system quality?

Latency

Measures response speed, crucial for real-time applications.

Token Usage

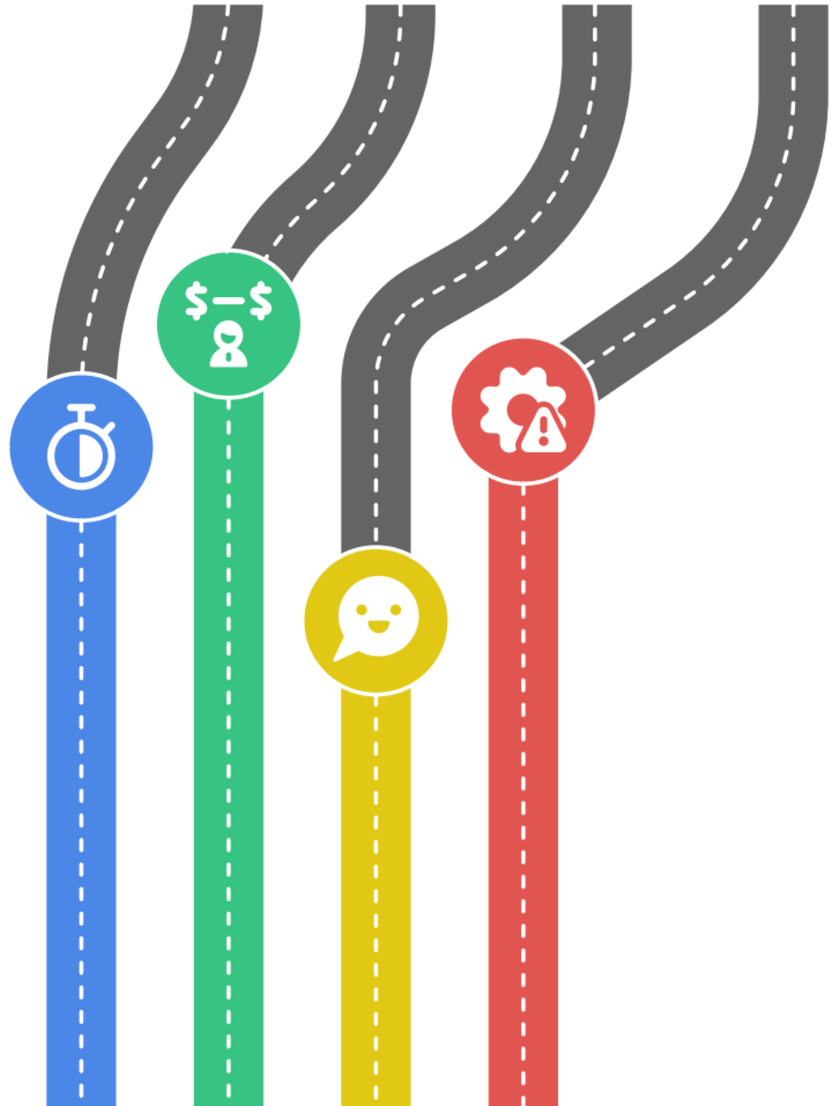
Monitors cost and efficiency by counting tokens.

User Satisfaction

Captures real-world value through feedback and ratings.

Additional Metrics

Includes error rates and safety violations for comprehensive evaluation.



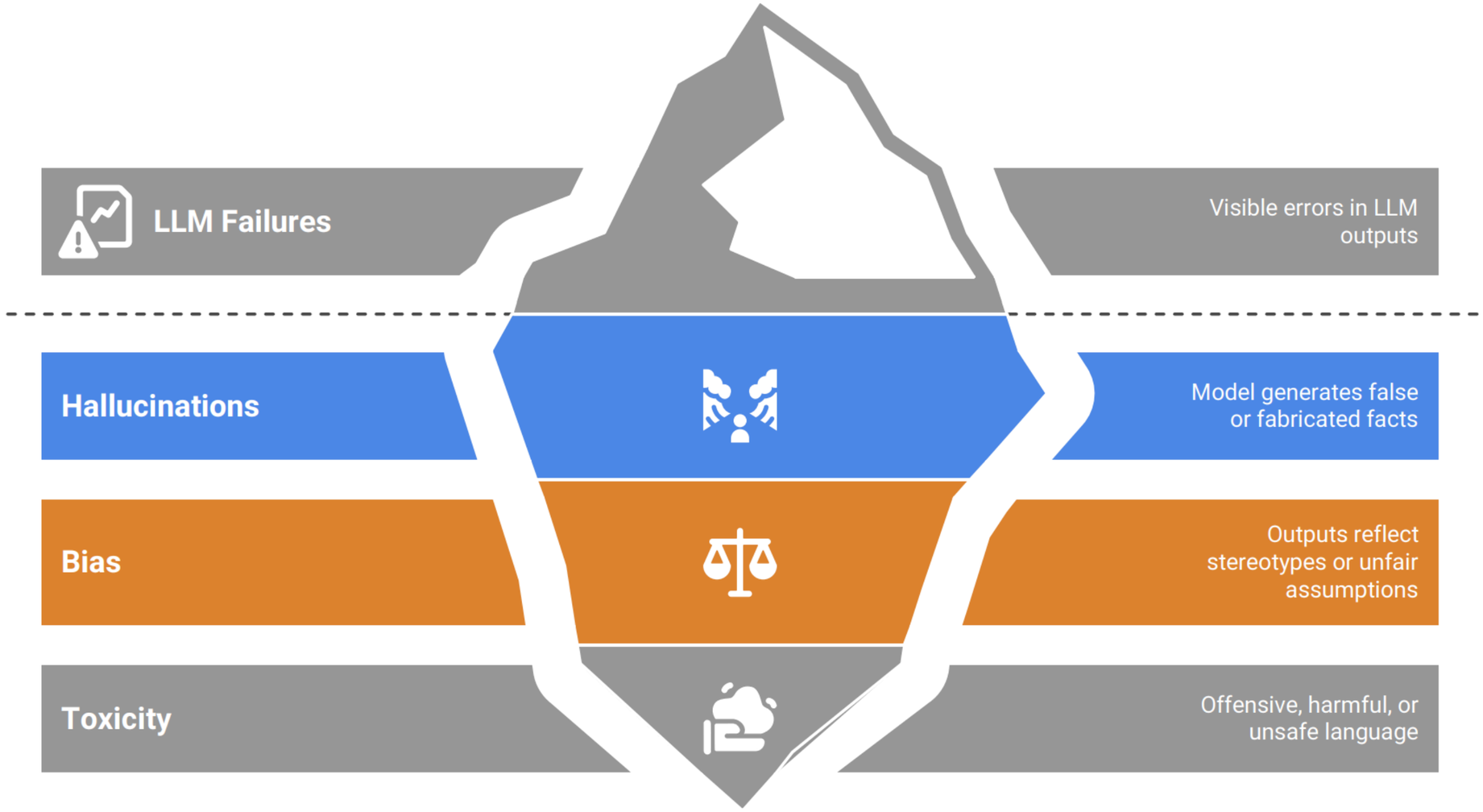
Systematic Error Analysis

1. Categorizing LLM Failures

- **Hallucinations**
  - Model generates false or fabricated facts
  - Example: Inventing references, wrong data in summaries
- **Bias**
  - Outputs reflect stereotypes or unfair assumptions
  - Example: Gender or racial bias in hiring suggestions
- **Toxicity**
  - Offensive, harmful, or unsafe language
  - Example: Abusive responses, unsafe advice
- **Why Categorize?**
  - Identifies patterns of failure
  - Guides targeted mitigation strategies (filters, retraining, guardrails)

Systematic Error Analysis

1. Categorizing LLM Failures

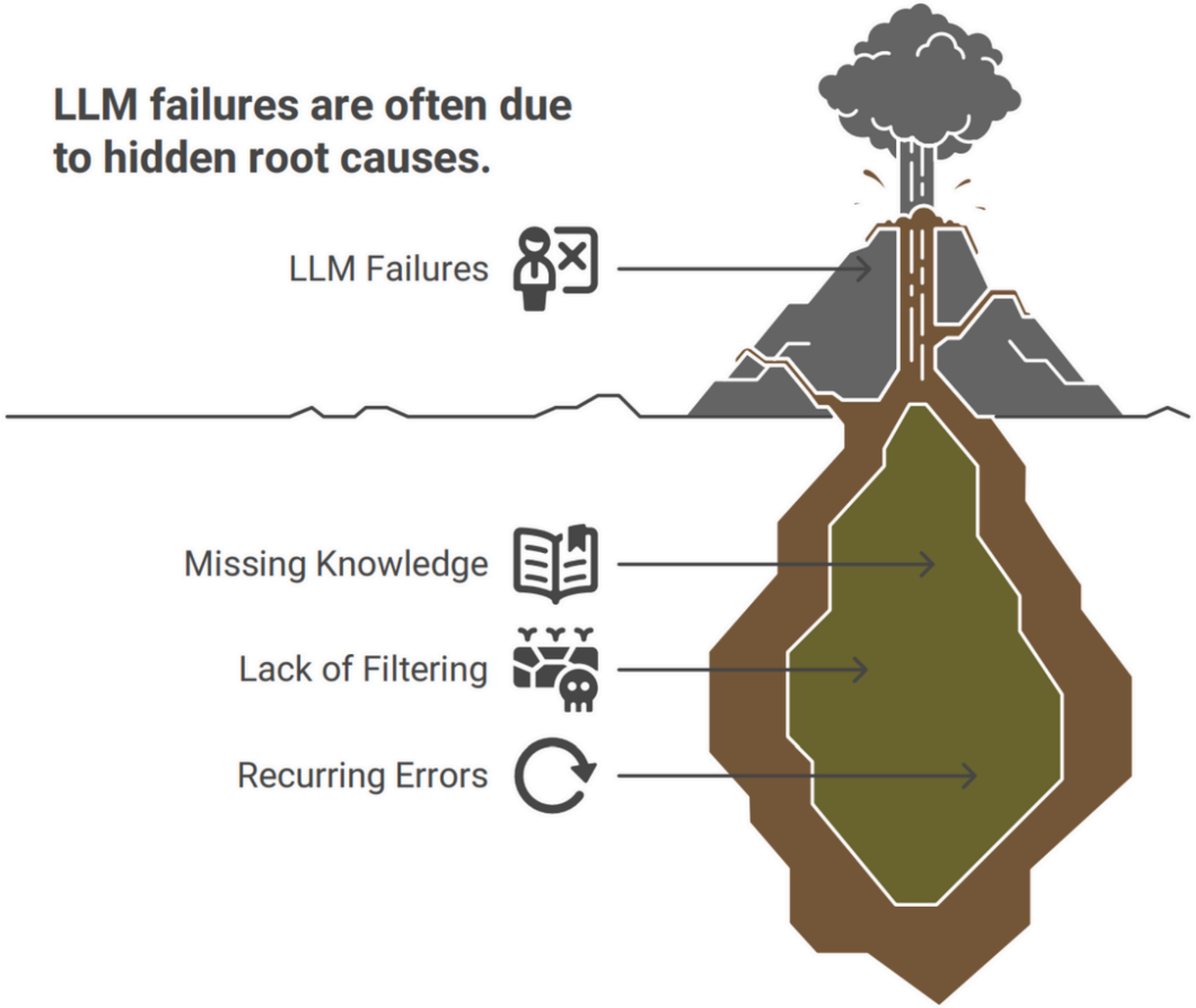


2. Root cause analysis frameworks

- **Why Root Cause Analysis?**
  - Go beyond symptoms → identify underlying reasons for LLM failures
  - Prevent recurring errors
- **Common Frameworks**
  - **5 Whys** → Ask "Why?" repeatedly until root cause is found
  - **Fishbone Diagram (Ishikawa)** → Categorize causes: data, model, deployment, user behavior
  - **Fault Tree Analysis** → Map how small failures combine into larger issues
- **Examples**
  - Hallucination traced back to missing knowledge in training data
  - Toxic output traced to lack of filtering in deployment pipeline
- **Goal**
  - Build systematic processes for diagnosing failures and guiding fixes

Systematic Error Analysis

2. Root cause analysis frameworks



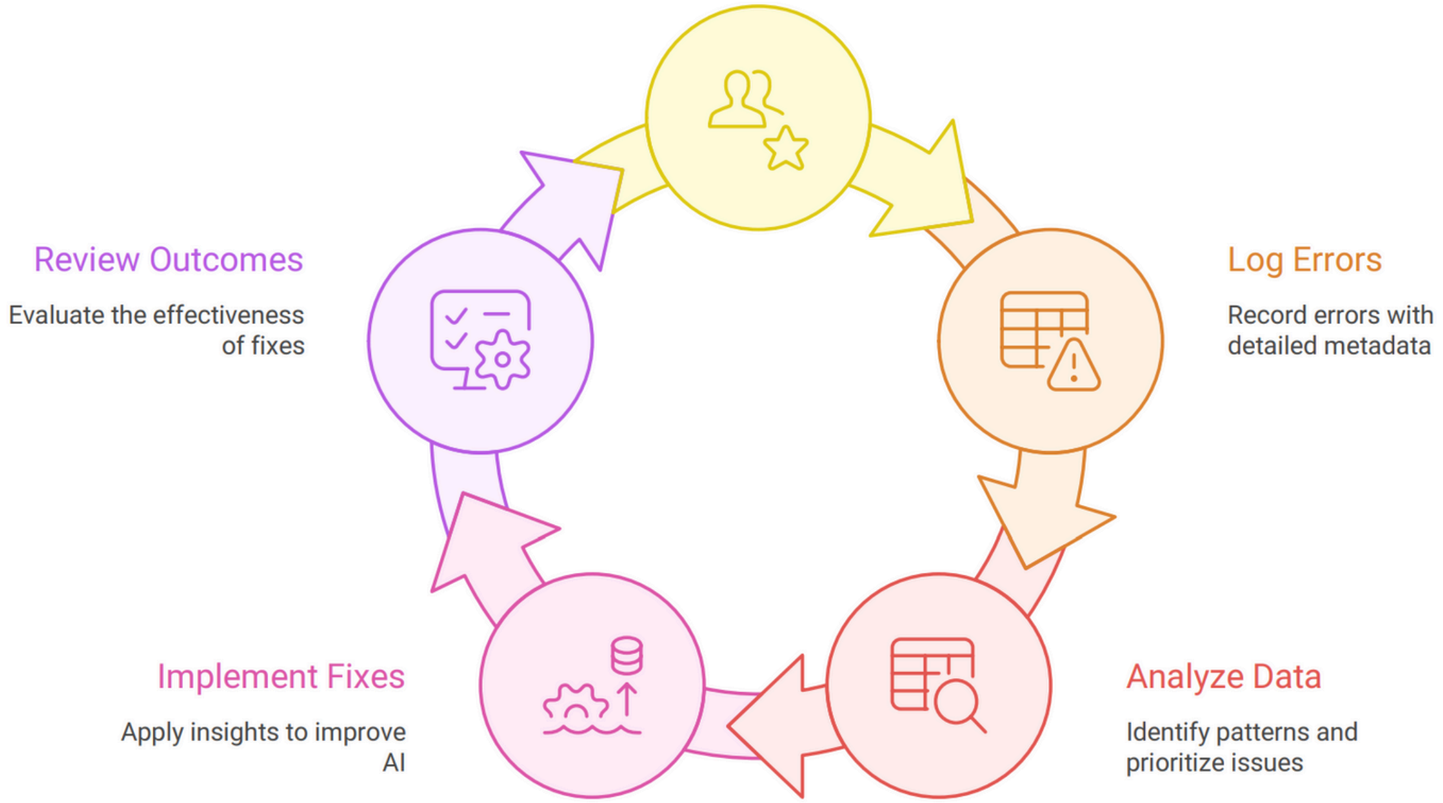
Systematic Error Analysis

3. Feedback loops and error logging strategies

- **Why Feedback Loops?**
  - Continuous improvement requires human + automated feedback
  - Prevents repeating the same mistakes
- **Feedback Sources**
  - **Explicit:** User ratings, thumbs up/down, surveys
  - **Implicit:** User re-asking same question, abandonment, session length
- **Error Logging Strategies**
  - Store errors with full trace: input, output, metadata
  - Tag errors by category (hallucination, bias, toxicity, latency)
  - Prioritize by frequency and impact
- **Closing the Loop**
  - Feed error data back into retraining, prompt optimization, or guardrails
  - Regular reviews to ensure fixes actually reduce failure rates
- **Goal**
  - Turn **errors into actionable insights** for continuous learning

Systematic Error Analysis

3. Feedback loops and error logging strategies



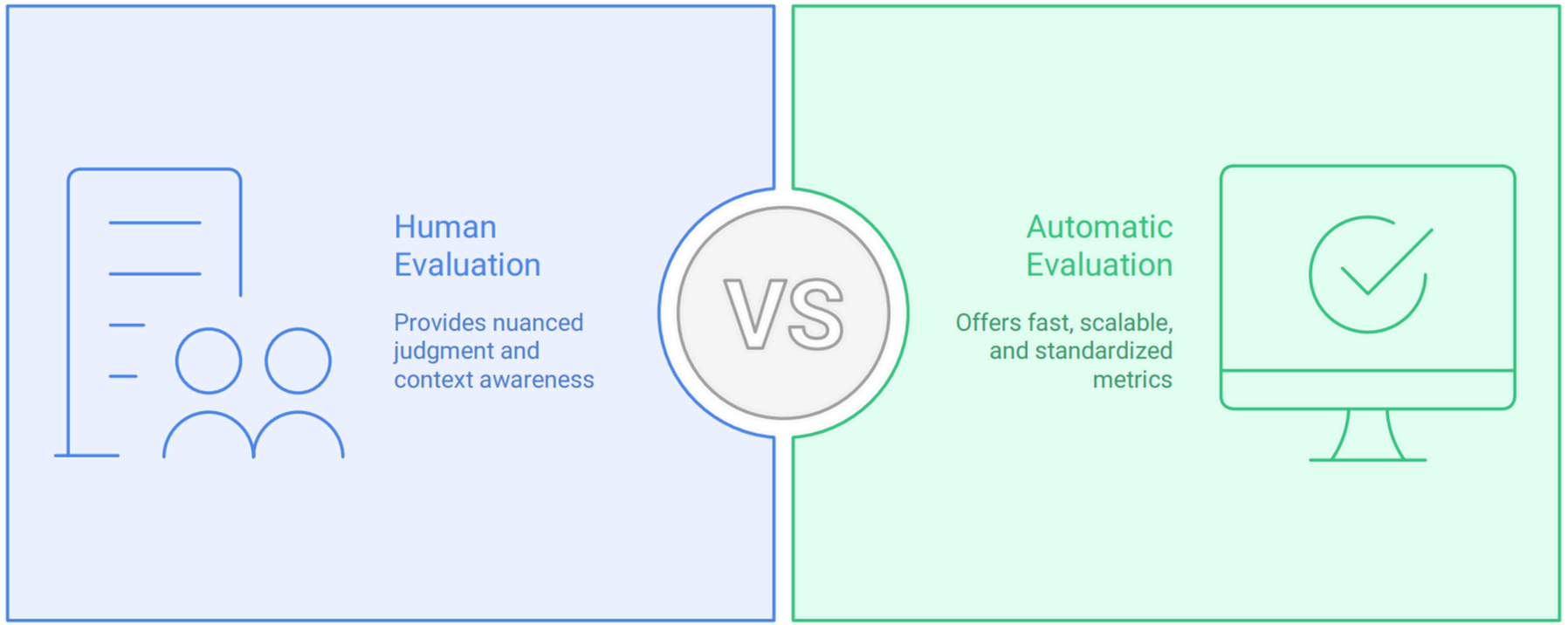
IV. Evaluation Techniques & LLM-Judge Approaches

1. Human evaluation vs automatic evaluation

- **Human Evaluation**
  - Humans review outputs for quality, helpfulness, safety
  - Pros: nuanced judgment, context awareness
  - Cons: costly, slow, not scalable
- **Automatic Evaluation**
  - Metrics: BLEU, ROUGE, accuracy, perplexity
  - Pros: fast, scalable, standardized
  - Cons: may miss context, subjective quality not captured
- **Trade-Off**
  - Human = depth, but expensive
  - Automatic = scale, but less nuance
  - Best practice: combine both for balanced evaluation

IV. Evaluation Techniques & LLM-Judge Approaches

1. Human evaluation vs automatic evaluation



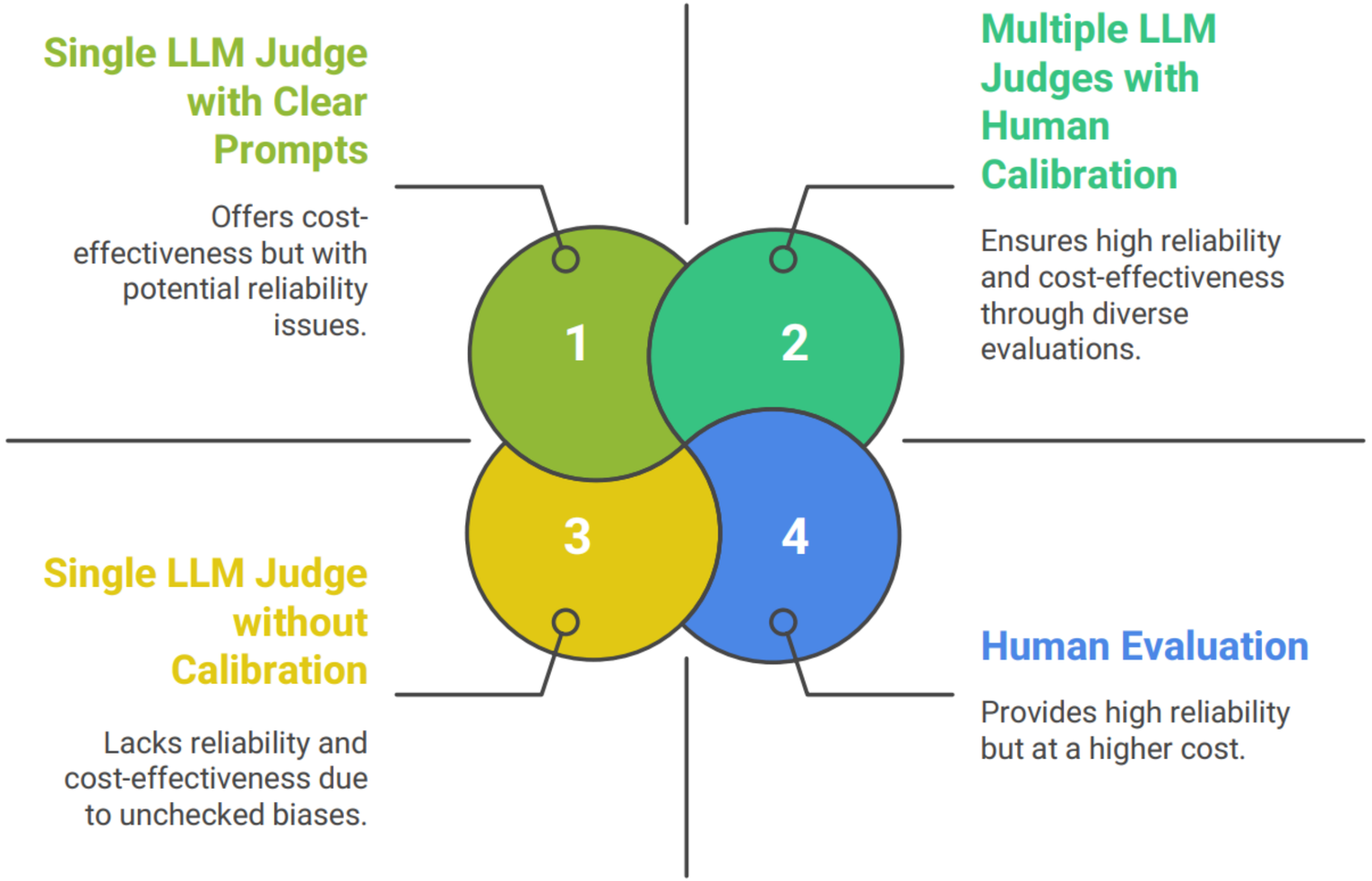
IV. Evaluation Techniques & LLM-Judge Approaches

2. Using LLMs to grade other LLMs

- **Concept**
  - Use one LLM to evaluate the outputs of another LLM
  - Often framed as grading, ranking, or scoring
- **Advantages**
  - Scalable: can process thousands of outputs automatically
  - Flexible: can assess dimensions like relevance, tone, style
  - Cost-effective compared to human evaluation
- **Limitations**
  - Judges can inherit **bias** from training data
  - Risk of **hallucinated judgments** (inconsistent scoring)
  - Still less reliable than expert human evaluators
- **Best Practices**
  - Combine with **human spot checks** for calibration
  - Use **clear evaluation prompts** to guide the judging LLM
  - Employ multiple judges for higher reliability

IV. Evaluation Techniques & LLM-Judge Approaches

2. Using LLMs to grade other LLMs



IV. Evaluation Techniques & LLM-Judge Approaches

3. Pairwise comparison and scoring methods

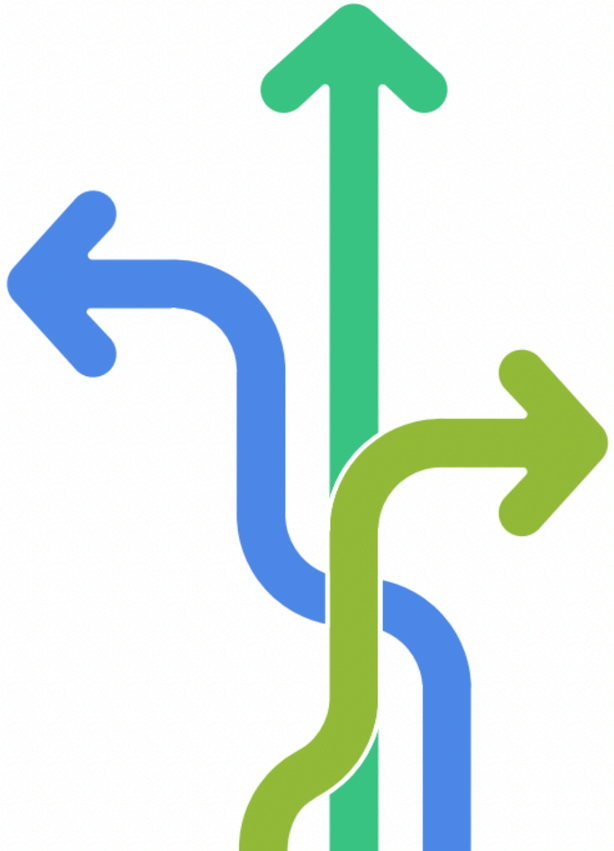
- **Pairwise Comparison**
  - Compare two outputs side by side
  - Judge selects which one is better
  - Useful for A/B testing between models
- **Scoring Methods**
  - Assign numerical scores (e.g., 1–5 scale) on dimensions like accuracy, clarity, safety
  - Allows aggregation into overall quality metrics
- **Advantages**
  - Pairwise: reduces subjectivity, easier for humans & LLM judges
  - Scoring: provides fine-grained insights across dimensions
- **Limitations**
  - Pairwise: does not measure absolute quality, only relative
  - Scoring: risks inconsistency between evaluators
- **Best Practice**
  - Combine pairwise for **relative comparisons** and scoring for **detailed evaluation**

IV. Evaluation Techniques & LLM-Judge Approaches

3. Pairwise comparison and scoring methods

Scoring Methods

Offers fine-grained insights across dimensions but risks inconsistency between evaluators.



Pairwise Comparison

Reduces subjectivity and is easier for judges but only provides relative quality assessments.

Combine Methods

Provides both relative and detailed evaluations, leveraging the strengths of both methods.

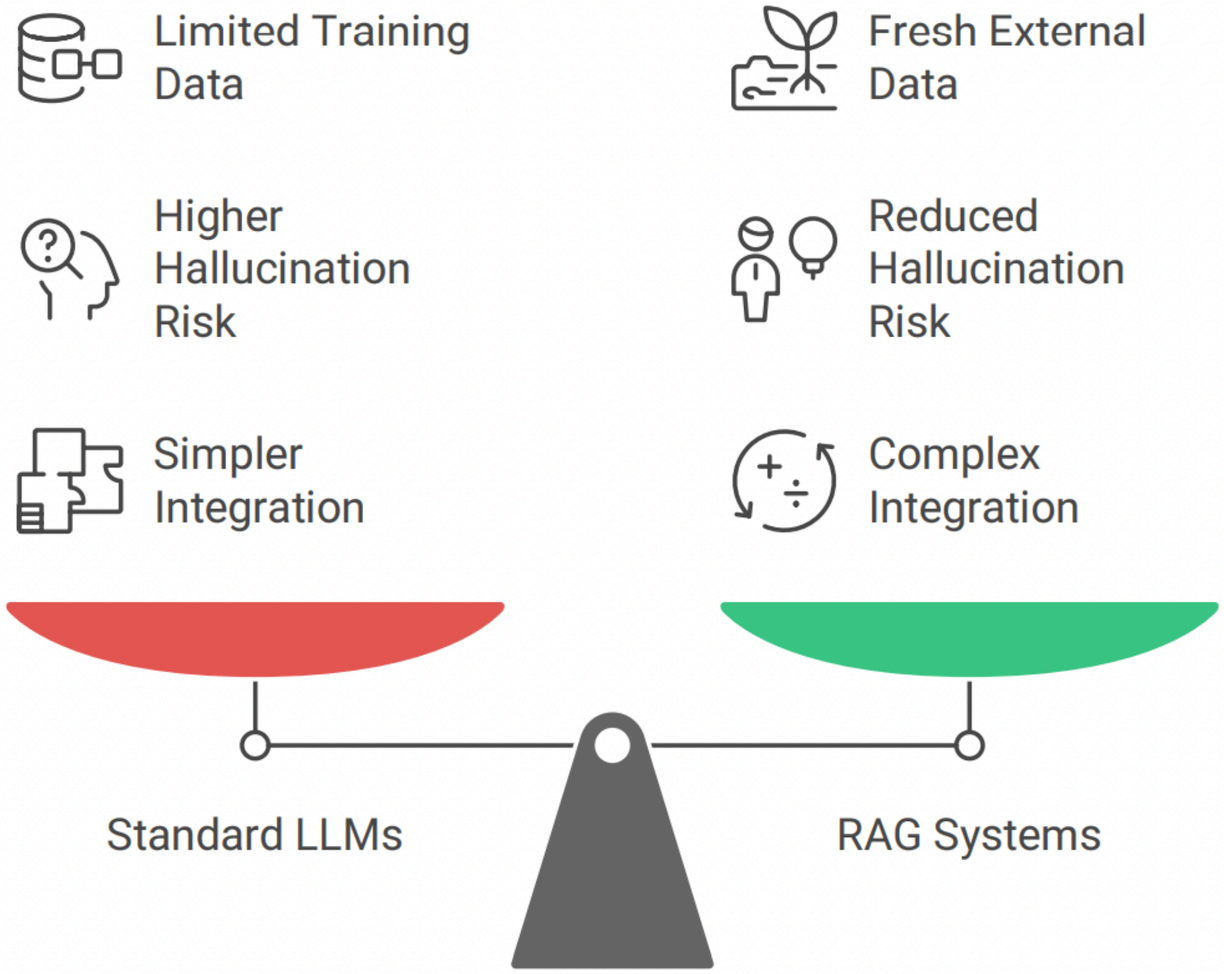
V. Evaluating RAG Systems

1. What Makes RAG Different?

- **Definition**
  - RAG = Retrieval-Augmented Generation
  - Combines **external knowledge retrieval** + **LLM generation**
- **Why Different from Plain LLMs?**
  - Standard LLMs rely only on training data → limited, may hallucinate
  - RAG retrieves **fresh, domain-specific, factual data** from external sources
- **Advantages**
  - Reduces hallucinations
  - Provides up-to-date and domain-specific knowledge
  - Improves trustworthiness and factual grounding
- **Challenges**
  - Retrieval errors → wrong or missing context
  - Integration complexity between retriever and generator

V. Evaluating RAG Systems

1. What Makes RAG Different?



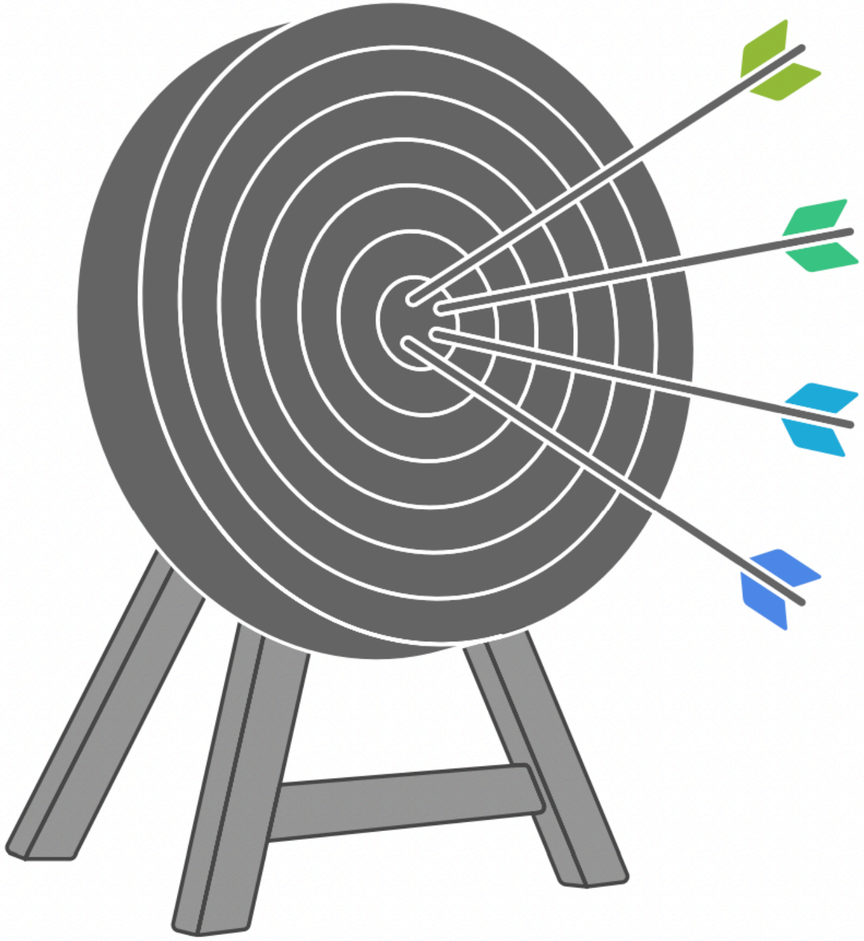
Evaluating RAG Systems

2. Evaluating Retrieval Quality

- **Recall** – Measures how many of the relevant documents were retrieved
  - High recall = fewer missed documents
  - Example: retrieving 8 out of 10 relevant papers → 80% recall
- **Precision** – Measures how many retrieved documents are actually relevant
  - High precision = fewer irrelevant results
  - Example: 7 relevant docs out of 10 retrieved → 70% precision
- **Relevance (Quality of match)**
  - Beyond raw numbers → how useful are the retrieved documents?
  - Example: retrieved doc is technically related but not answering the query
- **Why Important?**
  - Poor retrieval = poor generation
  - Strong retrieval = factual, grounded, trustworthy answers

Evaluating RAG Systems

2. Evaluating Retrieval Quality



Strong Retrieval

Factual, grounded, trustworthy answers

Relevance

Usefulness of retrieved documents

Precision

Proportion of relevant documents retrieved

Recall

Proportion of all relevant documents retrieved

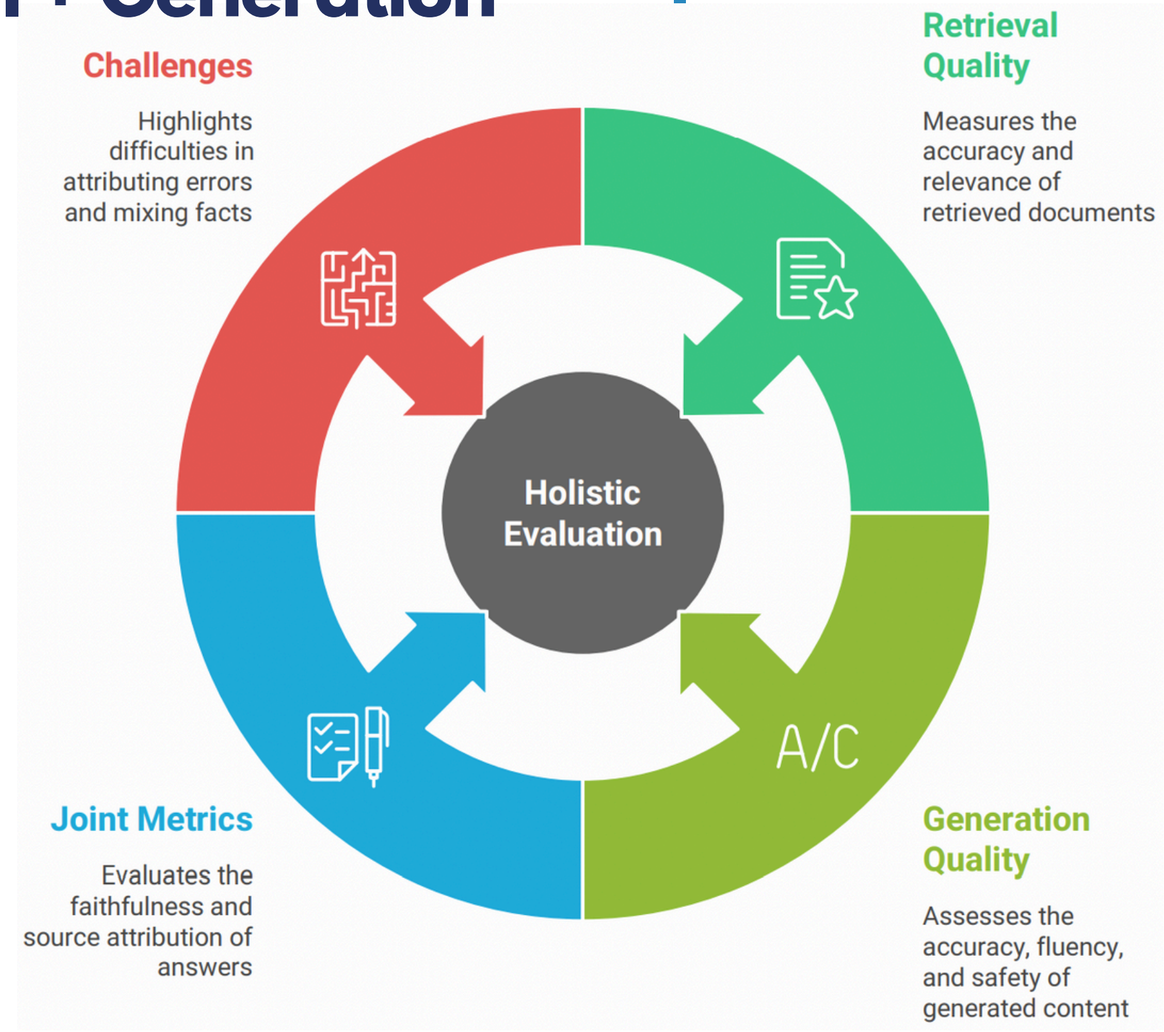
V. Evaluating RAG Systems

3. Combined Evaluation of Retrieval + Generation

- **Why Combined?**
  - Retrieval and generation are interdependent
  - Poor retrieval leads to poor answers, even with a strong LLM
- **Evaluation Layers**
  - **Retrieval Quality:** recall, precision, relevance of documents
  - **Generation Quality:** accuracy, fluency, safety, helpfulness
- **Joint Metrics**
  - Answer grounded in retrieved sources (faithfulness)
  - Source attribution: can user trace back facts to documents?
  - End-to-end success rate: does the system solve the user's task?
- **Challenges**
  - Errors can come from retrieval, generation, or both
  - Attribution is difficult when outputs mix facts with model knowledge
- **Goal**
  - Holistic evaluation of **the full pipeline**, not just isolated parts

V. Evaluating RAG Systems

3. Combined Evaluation of Retrieval + Generation



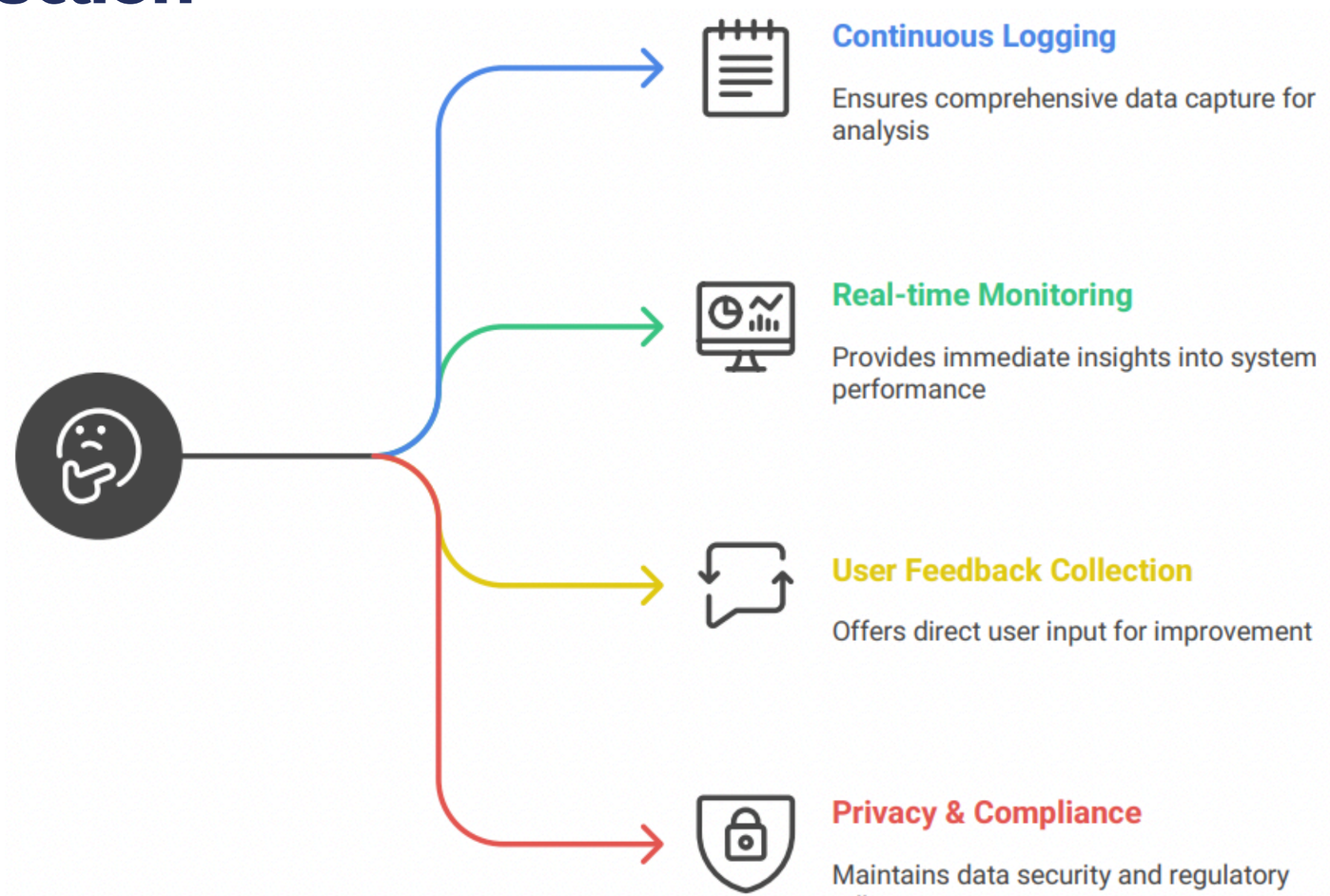
VI. Production Monitoring & Continuous Evaluation

1. Designing Evaluation in Production

- **Why Evaluate in Production?**
  - Lab benchmarks ≠ real-world behavior
  - Users, context, and scale reveal new issues
- **Key Considerations**
  - **Continuous logging** of inputs, outputs, metadata
  - **Real-time monitoring** of latency, errors, safety violations
  - **User feedback collection** (explicit & implicit signals)
- **Trade-offs**
  - Balance depth of evaluation with system performance & costs
  - Must ensure privacy & compliance when storing data
- **Goal**
  - Build an evaluation framework that runs **live, continuously, and safely**

VI. Production Monitoring & Continuous Evaluation

1. Designing Evaluation in Production



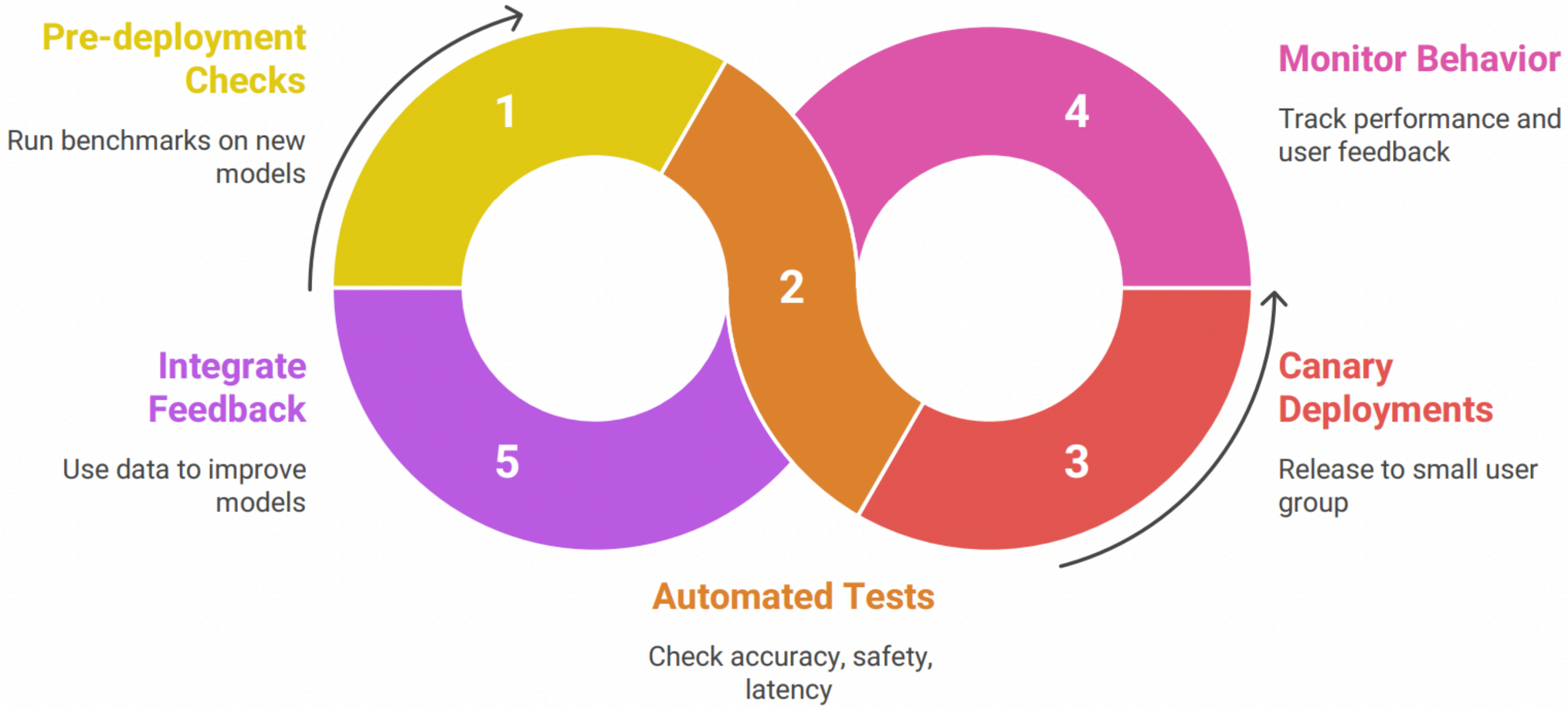
VI. Production Monitoring & Continuous Evaluation

2. Integrating Evaluation into CI/CD Pipelines

- **Why Integrate?**
  - Prevent regressions before deployment
  - Automate testing for faster, safer releases
- **CI/CD Evaluation Steps**
  - **Pre-deployment checks:** run evaluation benchmarks on new models
  - **Automated tests:** check accuracy, safety, latency thresholds
  - **Canary deployments:** release to small % of users, monitor behavior
- **Workflow Integration**
  - Embed evaluation into MLOps pipelines
  - Use monitoring tools (e.g., Prometheus, Grafana) in staging & production
- **Benefits**
  - Early detection of problems
  - Continuous quality assurance across updates
  - Faster iteration without sacrificing reliability

VI. Production Monitoring & Continuous Evaluation

2. Integrating Evaluation into CI/CD Pipelines



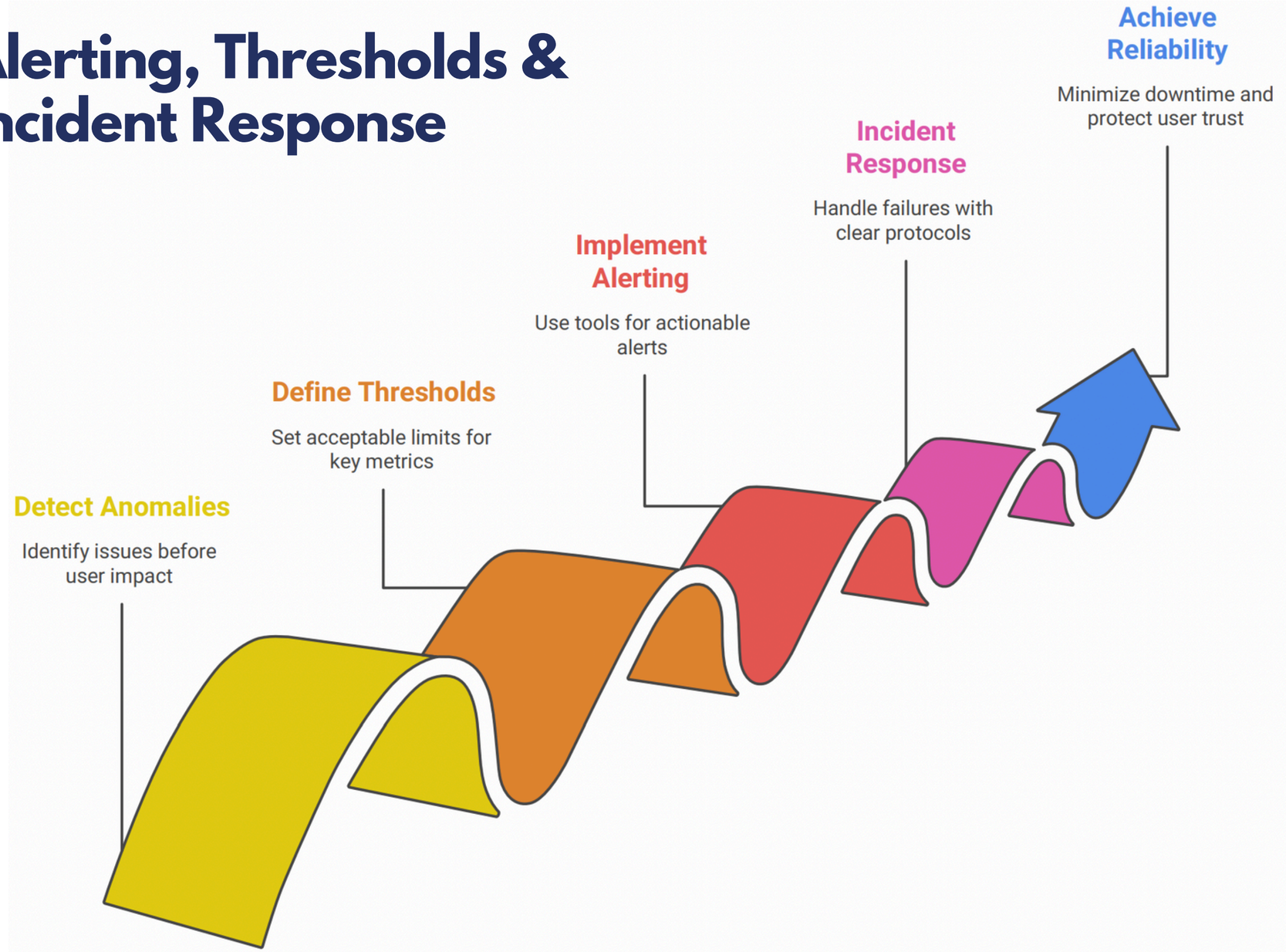
VI. Production Monitoring & Continuous Evaluation

3. Alerting, Thresholds & Incident Response

- **Why Alerts?**
  - Detect anomalies before users are impacted
  - Enable fast response to critical issues
- **Thresholds**
  - Define acceptable limits for latency, error rates, token usage, safety violations
  - Example: latency > 3s triggers alert, toxicity score > 0.2 flags content
- **Alerting Systems**
  - Use tools like Prometheus + Grafana, PagerDuty, or Slack integrations
  - Alerts should be actionable, not overwhelming ("alert fatigue")
- **Incident Response**
  - Playbooks for handling model failures (rollback, disable feature, escalate)
  - Clear ownership: who responds, how fast, escalation path
- **Goal**
  - Minimize downtime, ensure reliability, protect user trust

VI. Production Monitoring & Continuous Evaluation

3. Alerting, Thresholds & Incident Response



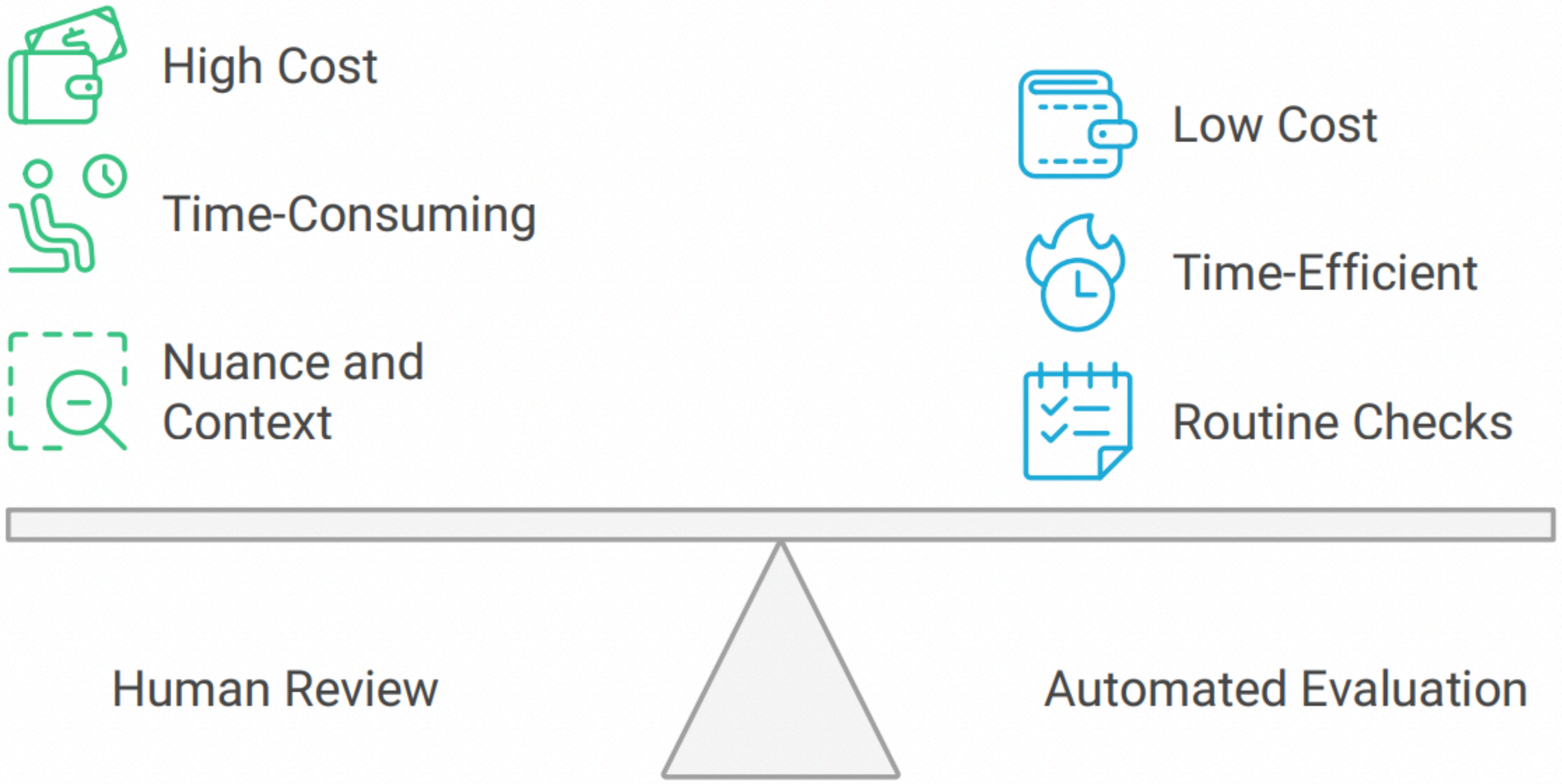
VII. Human Review & Cost Optimization

1. Scalable Human-in-the-Loop Review

- **Why Human Review?**
  - Automated evaluation misses nuance, context, and safety risks
  - Humans provide judgment for edge cases and sensitive tasks
- **Challenges**
  - Costly and time-consuming
  - Difficult to scale for millions of outputs
- **Scalable Approaches**
  - **Sampling:** review a subset of outputs instead of all
  - **Triage:** prioritize high-risk tasks (e.g., medical, financial, safety)
  - **Hybrid systems:** automation for routine checks, humans for critical ones
- **Best Practice**
  - Design workflows where humans validate **only where most valuable**

VII. Human Review & Cost Optimization

1. Scalable Human-in-the-Loop Review



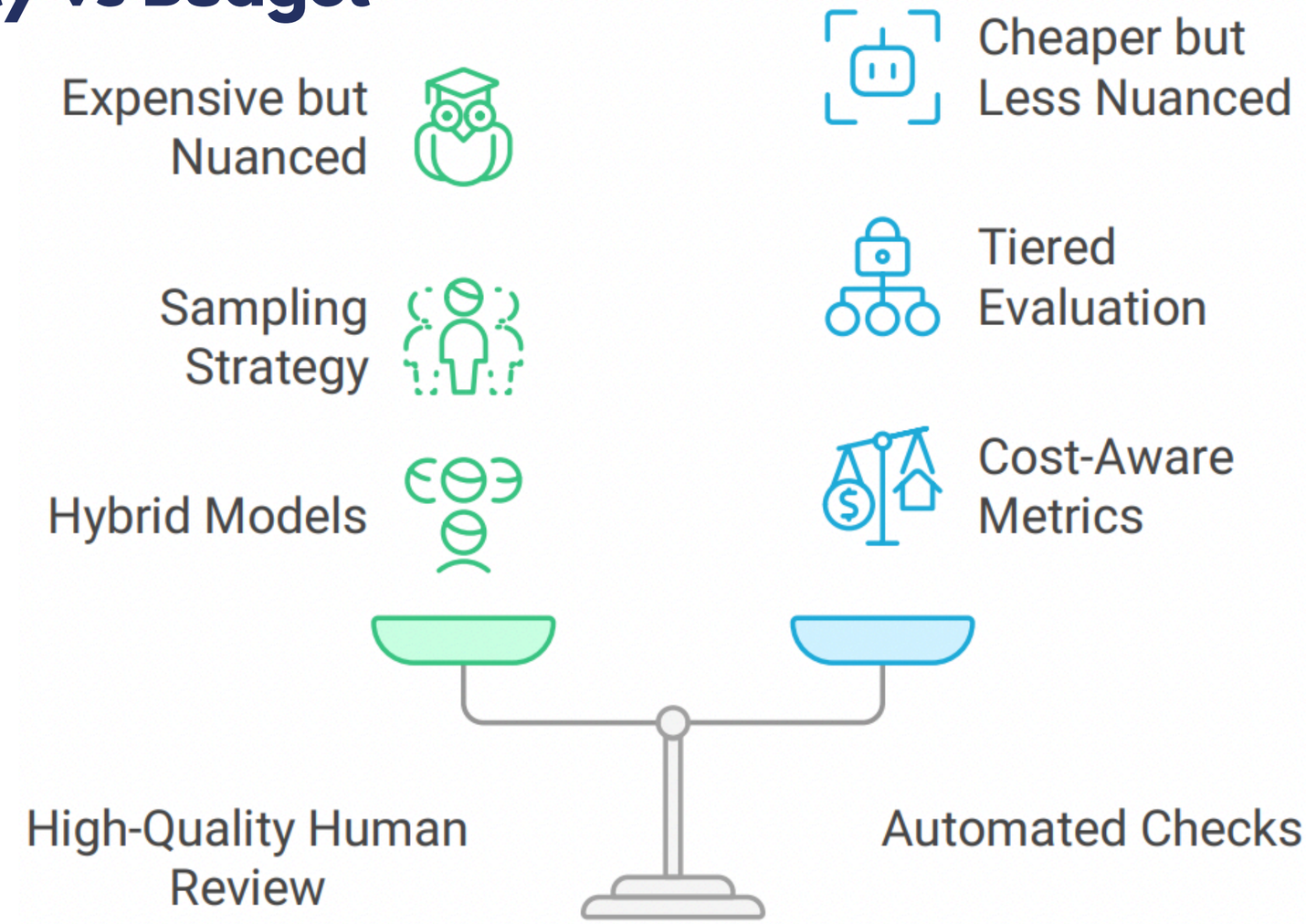
VII. Human Review & Cost Optimization

2. Balancing Evaluation Quality vs Budget

- **The Trade-off**
  - High-quality human review = expensive
  - Automated checks = cheaper, but less nuanced
- **Strategies to Balance**
  - **Sampling:** review a fraction of outputs to control costs
  - **Tiered evaluation:** more review for high-risk domains, less for low-risk
  - **Hybrid models:** combine automated scoring with human spot checks
- **Cost-Aware Metrics**
  - Prioritize metrics that impact business outcomes (safety, satisfaction)
  - Avoid over-spending on low-impact evaluations
- **Goal**
  - Optimize resources: achieve reliable evaluation **without overspending**

VII. Human Review & Cost Optimization

2. Balancing Evaluation Quality vs Budget



Human Review & Cost Optimization

3. Token & Model Strategies for Cost Optimization

- **Token Efficiency**
  - Shorter prompts → fewer tokens = lower cost
  - Use prompt engineering: avoid unnecessary context
  - Summarize past interactions before feeding back to model
- **Model Selection**
  - Match model size to task:
    - Lightweight models (e.g., GPT-3.5) for routine queries
    - Larger models (e.g., GPT-4, Claude Opus) for complex reasoning
  - Consider open-source models for specialized domains
- **Hybrid Strategies**
  - Route queries: cheap model for simple tasks, advanced model for critical cases
  - Use retrieval (RAG) to cut down context size and reduce token usage
- **Goal**
  - Minimize cost **without sacrificing quality**

VII. Human Review & Cost Optimization

3. Token & Model Strategies for Cost Optimization

Course Conclusion

1: Foundations of LLM Evaluation

2: Instrumentation & Observability

3: Systematic Error Analysis

4: Evaluation Techniques & LLM-Judge

5: Evaluating RAG Systems

6: Production Monitoring

7: Human Review & Cost Optimization

THANK YOU