Evaluation Metrics Overview

Without rigorous evaluation, you are guessing whether your LLM system is improving or degrading. These metrics form the foundation of every quality measurement pipeline, from offline benchmarks to production monitoring.

Text Similarity Metrics

Metric	What It Measures	Range	Strengths	Weaknesses
BLEU	N-gram precision overlap	0-1	Fast, established	Misses semantics
ROUGE-L	Longest common subsequence	0-1	Good for summaries	Surface-level only
ROUGE-1/2	Unigram/bigram overlap	0-1	Simple	No semantic understanding
METEOR	Alignment with synonyms + stems	0-1	Better than BLEU for short text	Slower
BERTScore	Cosine similarity of BERT embeddings	0-1	Semantic awareness	Compute-heavy
BLEURT	Learned metric (fine-tuned BERT)	-1 to 1	Best correlation with humans	Needs training

LLM-as-Judge

Approach	Description	Pros	Cons
Pointwise scoring	Rate output 1-5 on criteria	Simple	Scale inconsistency
Pairwise comparison	A vs B, which is better?	More reliable	2x cost, position bias
Reference-based	Compare to gold answer	Grounded	Needs reference answers
Rubric-based	Score against detailed rubric	Consistent	Rubric design takes effort

LLM-as-Judge Prompt Template

```
You are an expert evaluator. Rate the following response on a scale of 1–5
for each criterion.

Criteria:
– Relevance (1–5): Does the response address the question?
– Accuracy (1–5): Are the facts correct?
– Completeness (1–5): Does it cover all important aspects?
– Clarity (1–5): Is it well–written and easy to understand?

Question: {question}
Response: {response}
Reference Answer: {reference}
```

```
Provide scores as JSON:
{"relevance": N, "accuracy": N, "completeness": N, "clarity": N, "reasoning": "..."}
```

RAG-Specific Metrics

Metric	What It Measures	How to Compute
Faithfulness	Is the answer grounded in context?	LLM checks each claim against retrieved docs
Answer relevancy	Does the answer address the question?	Semantic similarity (answer, question)
Context relevancy	Are retrieved docs relevant?	LLM rates relevance of each chunk
Context recall	Did we retrieve enough?	Compare answer against reference
Answer correctness	Is the final answer right?	Compare to ground truth

Evaluation Frameworks

Do not build your evaluation infrastructure from scratch -- these frameworks handle the boilerplate of running test suites, scoring outputs, and comparing experiments so you can focus on defining what "good" means for your use case.

Framework	Key Feature	Language
RAGAS	RAG-specific metrics (faithfulness, relevancy)	Python
DeepEval	LLM-as-judge with many metrics	Python
Promptfoo	Config-driven eval, CI/CD friendly	Node.js
LangSmith	Tracing + evaluation in LangChain	Python
Braintrust	Logging, scoring, experiments	Python/TS
Arize Phoenix	Tracing, evals, embeddings viz	Python

Guardrail Types

Guardrails are the safety net between your LLM and your users. Each type trades off speed, cost, and accuracy -- production systems layer multiple types together for defense in depth.

Type	Speed	Cost	Accuracy	Use Case
Rule-based (regex, keyword)	Very fast	Free	Low-medium	Known bad patterns
Classifier (ML model)	Fast	Low	Medium-high	Topic/toxicity detection
LLM-based	Slow	High	High	Nuanced judgment
NER + PII detection	Fast	Low	High for PII	Data privacy
Embedding similarity	Fast	Low	Medium	Off-topic detection

Input Guardrails

Guardrail	Detects	Implementation
Prompt injection	Attempts to override system prompt	Classifier + LLM check
Jailbreak	Bypass attempts	Pattern matching + classifier
PII detection	SSN, email, phone, names	Regex + NER model (Presidio)
Topic restriction	Off-topic requests	Classifier or embedding distance
Input length	Extremely long inputs	Token counter + hard limit
Language detection	Unsupported languages	Language ID model
Rate limiting	Abuse, DDoS	Token bucket per user

Output Guardrails

Guardrail	Detects	Implementation
Hallucination check	Unsupported claims	Compare output vs source docs
Toxicity filter	Harmful content	Classifier (Perspective API, etc.)
PII leakage	Model outputs PII	Regex + NER on output
Format validation	Wrong output structure	JSON schema validation
Factual grounding	Unverifiable claims	Citation check, source matching
Brand safety	Off-brand content	Keyword list + classifier
Code safety	Dangerous code patterns	AST analysis, sandbox

Guardrail Implementation Patterns

A single guardrail layer will always have blind spots. The layered defense pattern catches cheap, obvious problems first and reserves expensive LLM-based checks for what slips through.

Layered Defense

```
Input
  │
  ├── Layer 1: Rule-based (fast, cheap)
  │   ├── Length check
  │   ├── Keyword blocklist
  │   └── Regex patterns
  │
  ├── Layer 2: Classifier (fast, moderate cost)
  │   ├── Toxicity classifier
  │   ├── Topic classifier
  │   └── PII NER
  │
```

```
   ├── Layer 3: LLM-based (slow, expensive)
   │   ├── Injection detection
   │   └── Nuanced policy check
   │
   ▼
LLM Call
   │
   ▼
Output
   │
   ├── Layer 1: Format validation (fast)
   ├── Layer 2: PII/toxicity scan (fast)
   └── Layer 3: Faithfulness check (if RAG)
```

Prompt Injection Detection

```
# Simple pattern-based check
INJECTION_PATTERNS = [
    r"ignore (?:all )?(?:previous|above|prior) instructions",
    r"you are now",
    r"new instructions:",
    r"system prompt:",
    r"forget (?:everything|your instructions)",
    r"disregard (?:all|your|the)",
]

def check_injection(text: str) -> bool:
    text_lower = text.lower()
    return any(re.search(p, text_lower) for p in INJECTION_PATTERNS)
```

PII Detection with Presidio

```
from presidio_analyzer import AnalyzerEngine

analyzer = AnalyzerEngine()
results = analyzer.analyze(
    text="My SSN is 123-45-6789 and email is john@example.com",
    language="en",
    entities=["PHONE_NUMBER", "EMAIL_ADDRESS", "US_SSN", "PERSON"]
)
# Returns: [type=US_SSN, start=10, end=21, score=0.85, ...]
```

Output Validation

```
import json
from jsonschema import validate, ValidationError

expected_schema = {
    "type": "object",
    "properties": {
        "answer": {"type": "string"},
        "confidence": {"type": "number", "minimum": 0, "maximum": 1},
        "sources": {"type": "array", "items": {"type": "string"}}
    },
```

```
    "required": ["answer", "confidence"]
}

def validate_output(llm_output: str) -> dict:
    try:
        parsed = json.loads(llm_output)
        validate(instance=parsed, schema=expected_schema)
        return {"valid": True, "data": parsed}
    except (json.JSONDecodeError, ValidationError) as e:
        return {"valid": False, "error": str(e)}
```

Guardrail Frameworks

These frameworks provide pre-built validators and scanners so you do not have to implement injection detection, PII scanning, and toxicity filtering from scratch.

Framework	Type	Key Features
Guardrails AI	Python SDK	Validators, structured output, retry
NeMo Guardrails	NVIDIA	Dialog rails, topic control, COLANG
LLM Guard	Open source	Input/output scanners, PII, toxicity
Lakera Guard	API	Prompt injection, PII, content
Rebuff	Open source	Multi-layer injection detection

Evaluation Pipeline Design

Evaluation is not a one-time event -- it is a continuous pipeline that runs offline before deployment and online after. Without both, you are either shipping untested changes or ignoring production degradation.

Offline Evaluation

```
1. Curate test dataset (100+ examples with ground truth)
2. Run model on test set
3. Compute automatic metrics (BLEU, ROUGE, BERTScore)
4. Run LLM-as-judge evaluation
5. Compute pass rates for guardrails
6. Human eval on sample (20-50 examples)
7. Compare against baseline
```

Online Monitoring

Metric	What to Track	Alert Threshold
Guardrail trigger rate	% of requests blocked	Sudden spike (> 2x baseline)
User feedback	Thumbs up/down ratio	Below 80% positive
Latency	P50, P95, P99	P95 > 5s

Metric	What to Track	Alert Threshold
Token usage	Avg tokens per request	> 2x expected
Error rate	% of failed requests	> 1%
Hallucination rate	% of ungrounded claims	> 10% (sample-based)

Common Pitfalls

The most dangerous pitfall is having no evaluation at all -- followed closely by evaluating the wrong thing. This list covers the mistakes that lead to false confidence in your LLM system.

Pitfall	Problem	Fix
Only using BLEU/ROUGE	Misses semantic quality	Add BERTScore + LLM-as-judge
No eval dataset	Can't measure improvements	Create before building features
Guardrails too strict	High false positive rate, bad UX	Tune thresholds, add fallback responses
Guardrails too loose	Harmful content leaks through	Layer multiple methods
Only testing happy path	Fails on adversarial input	Red-team with injection attempts
No monitoring in prod	Silent degradation	Log guardrail triggers, sample outputs
LLM-as-judge position bias	Prefers first/last option	Randomize order, average across positions
Eval on training data	Misleading results	Strict train/eval split
One-time eval	Model or data drifts	Automate eval in CI/CD

Quick Decision: Which Metric?

Different tasks demand different metrics -- using BLEU for open-ended QA or human eval for classification wastes time and gives misleading results. Match the metric to the task.

Task	Primary Metric	Secondary
Summarization	ROUGE-L	BERTScore, human eval
Translation	BLEU	COMET, human eval
Classification	Accuracy, F1	Precision, Recall
Open QA	LLM-as-judge	BERTScore vs reference
RAG	Faithfulness, Answer Relevancy	Context Relevancy, Recall
Code generation	pass@k (execution)	CodeBLEU
Chat/dialog	Human eval, LLM-as-judge	User satisfaction metrics