



AI
Bees

Generative AI

Evaluation Metrics

Text Similarity • LLM Evaluation • RAG Assessment

BLEU • ROUGE • BERTScore • Cosine Similarity • RAGAS • DeepEval

This document builds understanding in the correct order — from the simplest scoring techniques to the most sophisticated LLM-aware evaluation frameworks:

| Section | Topic | What You Will Understand |
|---------|---|---|
| 1 | Why Evaluation Matters | How we measure whether an AI system's output is any good |
| 2 | Classification Metrics & Confusion Matrix | Precision, Recall, F1 — the foundation of all ML evaluation |
| 3 | BLEU — N-gram Overlap | Word-matching score for translation and generation tasks |
| 4 | ROUGE — Recall-Oriented Scoring | Summary evaluation via recall of reference text words |
| 5 | BERTScore — Semantic Similarity | Meaning-aware scoring using BERT embeddings |
| 6 | Cosine Similarity | Embedding-based similarity without a reference text |
| 7 | RAGAS — RAG-Specific Evaluation | Framework for scoring RAG pipelines (faithfulness, relevancy) |
| 8 | DeepEval — Production Evaluation | Comprehensive LLM evaluation with GEval custom metrics |
| 9 | Choosing the Right Metric | Decision guide for matching metric to task |


1. Why Evaluation Matters

1.1 The Core Problem

When a human writes a summary, edits a translation, or answers a question, another human can read both the original and the output and immediately judge whether it is good or bad. But when an AI system produces text, how do we measure quality automatically — at scale, without reading every single output?

This is the central challenge of NLP and LLM evaluation. We need metrics that can assign a meaningful numerical score to generated text so we can: compare different models, monitor quality in production, detect regression when we update the system, and identify specific failure modes (hallucination, irrelevance, bias).

| Evaluation Challenge | Why It Is Hard | Metric That Addresses It |
|---|--|--|
| Does the summary cover the main points? | Need to compare against a reference summary | ROUGE (recall-based) |
| Does the translation match the reference? | N-gram overlap with reference translation | BLEU |
| Do the two sentences mean the same thing? | Synonyms and paraphrases fool word-matching | BERTScore, Cosine Sim. |
| Did the LLM make up facts? | Requires understanding the retrieved context | RAGAS Faithfulness, DeepEval Hallucination |
| Does the answer address the question? | Requires reasoning about question–answer fit | RAGAS Answer Relevancy |
| Are the retrieved chunks relevant? | Requires judgement about retrieval quality | RAGAS Context Precision/Recall |

 **Key Point:** There is no single perfect metric. Every task needs a different combination. A good ML engineer understands what each metric is measuring — and more importantly, what it is NOT measuring.

2. Classification Metrics — The Foundation

2.1 The Confusion Matrix

Before diving into text similarity metrics, it is essential to understand Precision, Recall, and F1 Score — because they appear in almost every evaluation framework (ROUGE, BERTScore, RAGAS all compute these). These metrics originate from the Confusion Matrix, a 2x2 table that compares what the model predicted against what was actually true.

Consider a spam filter. Every email gets either classified as Spam or Not Spam. The four possible outcomes are:

- **True Positive (TP)**: email IS spam, model correctly predicted SPAM. ✓
- **True Negative (TN)**: email is NOT spam, model correctly predicted NOT SPAM. ✓
- **False Positive (FP)**: email is NOT spam, but model incorrectly predicted SPAM. ✗ (Type I Error)
- **False Negative (FN)**: email IS spam, but model incorrectly predicted NOT SPAM. ✗ (Type II Error)

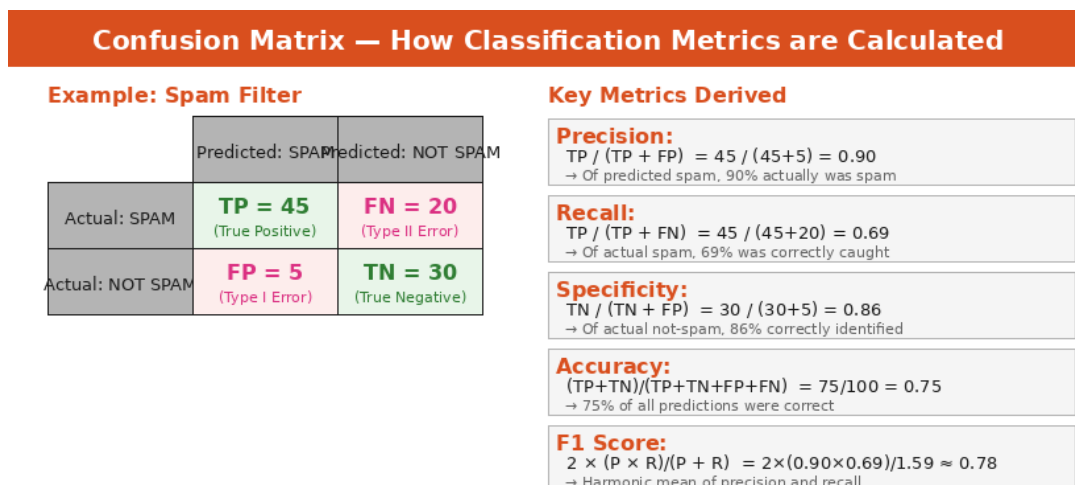


Figure 1: Confusion Matrix with spam filter example (TP=45, FN=20, FP=5, TN=30) and all derived metrics

2.2 The Four Key Metrics Explained

Precision — Quality of Positive Predictions

Precision answers: 'Of all the emails I marked as spam, what fraction actually were spam?' It measures how accurate the model is when it fires an alarm. High precision = few false alarms.

$$\text{Precision} = TP / (TP + FP) = 45 / (45 + 5) = 0.90$$

In our example: 90% of emails flagged as spam genuinely were spam. Only 5 legitimate emails were incorrectly flagged.

Recall (Sensitivity) — Coverage of Actual Positives

Recall answers: 'Of all the emails that actually were spam, what fraction did I catch?' It measures how good the model is at not missing things. High recall = few missed cases.

$$\text{Recall} = TP / (TP + FN) = 45 / (45 + 20) = 0.69$$

In our example: the filter caught 69% of actual spam emails but let 20 spam emails slip through to the inbox.

The Precision–Recall Trade-off

Precision and Recall are inversely related in most systems. If you lower the threshold to mark emails as spam (catch more spam = higher recall), you will also flag more legitimate emails (lower precision).

Choosing the right balance depends on the cost of each error type:

- **Medical diagnosis:** high recall is critical — missing a disease (false negative) is worse than an unnecessary test.
- **Spam filter:** high precision is often preferred — blocking legitimate email (false positive) is worse than missing spam.

F1 Score — Harmonic Mean of Precision and Recall


F1 Score combines Precision and Recall into a single number. It is the harmonic mean, which means it penalises extreme imbalances — a model with perfect precision (1.0) but zero recall (0.0) gets an F1 of 0, not 0.5.

$$\begin{aligned} \text{F1 Score} &= 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall}) \\ \text{F1} &= 2 \times (0.90 \times 0.69) / (0.90 + 0.69) = 1.242 / 1.59 \approx 0.78 \end{aligned}$$

Accuracy and Specificity

Accuracy is the simplest metric — percentage of all predictions that were correct. But it is misleading for imbalanced datasets. If 95% of emails are legitimate, a model that marks everything as 'Not Spam' gets 95% accuracy while being completely useless for spam detection.

$$\begin{aligned} \text{Accuracy} &= (TP + TN) / (TP + TN + FP + FN) = (45 + 30) / 100 = 0.75 \\ \text{Specificity} &= TN / (TN + FP) = 30 / (30 + 5) = 0.86 \end{aligned}$$

 **Remember:** F1 Score is almost always more informative than Accuracy for imbalanced classes. Use Accuracy only when the classes are roughly balanced.

3. BLEU — Bilingual Evaluation Understudy

3.1 What is BLEU?

BLEU (Bilingual Evaluation Understudy) was developed by IBM Research in 2002 to automatically evaluate machine translation quality. It measures how many n-grams (contiguous sequences of N words) in the generated text also appear in the reference text. BLEU was a breakthrough because it enabled automated translation evaluation without human judges.

Despite its name, BLEU is now used far beyond translation — for summarisation, question answering, and any text generation task where a reference output is available.

- **N-gram**: a contiguous sequence of N words. 'The cat' is a bigram. 'The cat sat' is a trigram.
- **Reference text**: the ground-truth or expected output, written by a human.
- **Generated text**: the text produced by the LLM or model being evaluated.

3.2 How BLEU is Calculated — Step by Step

BLEU computes precision scores for n-grams of size 1 (unigrams), 2 (bigrams), 3 (trigrams), and 4 (4-grams), then combines them using a geometric mean.

Example from your slides

```
Reference: 'The cat is on the mat'    (6 words)
Generated: 'The cat and the dog'      (5 words)
```

Step 1 — Unigram Precision (P1)

Count how many words from the generated text also appear in the reference. Use modified precision (cap by reference count to prevent repetition gaming).

```
Words in generated:  The, cat, and, the, dog
Matches in reference: 'The' appears 1× in ref → match count = 1
                    'cat' appears 1× in ref → match count = 1
                    'and' → not in ref → 0
                    'the' → (clipped to 1, already counted) → match = 1
                    'dog' → not in ref → 0
Matched unigrams = 3 | Total generated unigrams = 5
P1 = (2 + 1) / 5 = 3/5 = 0.60
```

Step 2 — Bigram Precision (P2)

```
Bigrams in generated: 'The cat', 'cat and', 'and the', 'the dog'
Bigrams in reference: 'The cat', 'cat is', 'is on', 'on the', 'the mat'
Matching bigrams:     'The cat' → 1 match
P2 = 1 / 4 = 0.25
```

Step 3 — Trigram and 4-gram Precision (P3, P4)

```
Trigrams: 'The cat and', 'cat and the', 'and the dog' — NONE match reference
4-grams:  'The cat and the', ... — NONE match reference
P3 = 0     P4 = 0
```

Step 4 — Geometric Mean

$$BLEU = \text{geometric_mean}(P1, P2, P3, P4) = \sqrt[4]{(0.60 \times 0.25 \times 0 \times 0)} = 0$$

Because P3 and P4 are zero, the entire BLEU score collapses to zero! This is a known limitation.

3.3 The Smoothing Solution

The zero-score problem is addressed with Laplace Smoothing (also called add-1 smoothing). Instead of using raw counts, each n-gram count gets 1 added to both numerator and denominator before dividing. This prevents the score from collapsing to zero just because no trigram matches were found.

$$\text{Modified Precision} = (\text{Matching N-grams} + 1) / (\text{Total Generated N-grams} + 1)$$

```
# Python - from your code
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

ref_tokens = [nltk.word_tokenize(reference_text)] # list of lists
gen_tokens = nltk.word_tokenize(generated_text)

# method1 = Laplace smoothing - most commonly used
bleu = sentence_bleu(ref_tokens, gen_tokens,
                     smoothing_function=SmoothingFunction().method1)
```

⚠ Limitation: BLEU does not understand meaning. 'The cat sat on the mat' and 'The feline rested on the rug' mean the same thing but score very low because they share almost no n-grams. BLEU rewards exact word matches, not semantic similarity.

✅ Use When: Machine translation quality evaluation, comparing two versions of a model on the same test set, and any task where an exact reference output exists. NOT suitable for open-ended generation or semantic similarity.

4. ROUGE — Recall-Oriented Understudy for Gisting Evaluation

4.1 What is ROUGE?

ROUGE was designed specifically for automatic summarisation evaluation. While BLEU is precision-oriented (how many generated words are in the reference?), ROUGE is recall-oriented (how many reference words appear in the generated summary?). This makes ROUGE more appropriate for summarisation — we care more about whether the summary covers the key points than whether it avoids irrelevant words.

| Aspect | BLEU | ROUGE |
|---------------------|--|--|
| Primary orientation | Precision — generated words matching reference | Recall — reference words covered by summary |
| Primary use case | Machine translation | Text summarisation |
| Best question asked | 'Is the generated text accurate?' | 'Does the summary cover the key content?' |
| N-gram variants | P1, P2, P3, P4 geometric mean | ROUGE-1 (unigrams), ROUGE-2 (bigrams), ROUGE-L (LCS) |
| Score range | 0 to 1 (higher = more overlap) | 0 to 1 (higher = better coverage) |

4.2 ROUGE-1 — Unigram Recall

ROUGE-1 measures the overlap of individual words (unigrams) between the generated summary and the reference summary. The key metric is Recall.

```
Reference Summary: 'The cat is on the mat.'           (6 words)
Generated Summary: 'The cat and the dog is on the mat.' (9 words)
```

$$\text{ROUGE-1 Recall} = \text{Overlapping Words} / \text{Total Words in Reference}$$

```
Reference words present in generated:
'The' (✓) 'cat' (✓) 'is' (✓) 'on' (✓) 'the' (✓) 'mat' (✓) → 6 matches

ROUGE-1 Recall = 6 / 6 = 1.0
```

The recall is 1.0 because every word from the reference appears in the generated summary. Note that the generated text has extra words ('and the dog') — recall doesn't penalise for extra content, only for missing reference content.

4.3 ROUGE-L — Longest Common Subsequence (LCS)

ROUGE-L uses the Longest Common Subsequence (LCS) — the longest sequence of words that appears in both texts in the same order, but not necessarily consecutively. This captures structural similarity better than simple word counts.

```
Reference: 'The cat is on the mat.'
Generated: 'The cat and the dog is on the mat.'

LCS: 'The cat is on the mat' — all 6 reference words appear in order
LCS Length = 6
```

$$\begin{aligned}
 \text{ROUGE-L Recall} &= \text{LCS Length} / \text{Reference Length} = 6/6 = 1.0 \\
 \text{ROUGE-L Precision} &= \text{LCS Length} / \text{Generated Length} = 6/9 \approx 0.67 \\
 \text{ROUGE-L F1} &= 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall}) \\
 &= 2 \times (0.67 \times 1.0) / (0.67 + 1.0) = 1.34 / 1.67 \approx 0.8
 \end{aligned}$$

```

# Python - from your code
from rouge_score import rouge_scorer

scorer = rouge_scorer.RougeScorer(['rouge1', 'rougeL'], use_stemmer=True)
# use_stemmer=True normalises words (running → run, better → good)

scores = scorer.score(reference_text, generated_text)
print(scores['rouge1'].fmeasure)    # F1 for ROUGE-1
print(scores['rougeL'].fmeasure)    # F1 for ROUGE-L

```

⚠ **Limitation:** ROUGE ignores synonyms. A generated summary that uses 'automobile' instead of 'car' scores zero for that word even though the meaning is identical. Like BLEU, it only matches exact words (or stems with `use_stemmer=True`).

✅ **Use When:** Evaluating extractive or abstractive summarisation. Comparing models when you have human-written reference summaries. Use ROUGE-1 for general word coverage, ROUGE-L when word order and sentence structure matter.

5. BERTScore — Semantic Similarity via Embeddings

5.1 The Problem BERTScore Solves

Both BLEU and ROUGE rely on exact word matches or n-gram overlaps. This means they completely fail to recognise that 'The feline rested on the rug' and 'The cat sat on the mat' express the same meaning. BERTScore solves this by comparing sentences at the semantic (meaning) level using contextual embeddings from BERT.

Instead of asking 'do these words match?', BERTScore asks 'are these words semantically similar based on their meaning in context?' This is a fundamentally more powerful question.

5.2 How BERTScore Works — Three Steps

Step 1 — Token Embeddings

Both the generated text and the reference text are fed through a pre-trained BERT model. BERT produces a contextual embedding (a dense vector of ~768 numbers) for every token in both sentences. These embeddings capture not just the word itself but its meaning in context — 'bank' near 'river' gets a different embedding than 'bank' near 'money'.

```
Generated: 'The automobile raced down the street'
Reference: 'The car sped along the road'

BERT produces embeddings for each token:
'The' (generated) → [0.12, -0.45, 0.33, ...] 768-dim vector
'automobile' → [0.88, -0.12, 0.71, ...] 768-dim vector
'car' (reference) → [0.85, -0.09, 0.69, ...] 768-dim vector
↑ automobile and car have very similar vectors!
```

Step 2 — Cosine Similarity Matrix

For every token in the generated sentence, BERTScore finds the most semantically similar token in the reference sentence by computing cosine similarity between their embeddings. This builds a full similarity matrix of all generated-token × reference-token pairs.

$$\text{Cosine Similarity}(A, B) = (A \cdot B) / (|A| \times |B|)$$

Cosine similarity ranges from -1 to 1. A value of 1.0 means the two vectors point in exactly the same direction (identical meaning). Values above 0.85 typically indicate highly semantically similar tokens.

Step 3 — Soft Alignment Scores

BERTScore uses greedy matching — each generated token is matched to the most similar reference token, and vice versa. This produces three scores:


- **BERTScore Precision**: for each generated token, find its best-matching reference token. Average these similarities.
- **BERTScore Recall**: for each reference token, find its best-matching generated token. Average these similarities.
- **BERTScore F1**: harmonic mean of BERTScore Precision and Recall — the most commonly used BERTScore metric.


```
# Python — from your code
import bert_score


# lang='en' uses the default English BERT model
```

```
# Returns three tensors: precision, recall, F1 (per example)
P, R, F1 = bert_score.score([generated_text], [reference_text],
                             lang='en', verbose=False)

print(f'BERTScore F1: {F1[0].item():.4f}')
# Typically 0.85-0.95 for good paraphrases
```

 **Remember:** BERTScore uses 'soft' matching — it doesn't require exact word matches, only semantic similarity. This means synonyms, paraphrases, and even different sentence structures can score highly if they convey the same meaning.

 **Limitation:** BERTScore requires a pre-trained BERT model (downloaded automatically), so it is slower and requires more compute than BLEU or ROUGE. It also still needs a reference text — it cannot evaluate generation quality without ground truth.

 **Use When:** Any task where paraphrasing is valid and meaning matters more than exact wording: evaluating abstractive summaries, dialogue systems, creative writing, or comparing semantic equivalence between two texts.

6. Cosine Similarity — Unsupervised Semantic Scoring

6.1 What is Cosine Similarity?

Cosine similarity measures the angle between two embedding vectors in high-dimensional space. Unlike BERTScore which compares token by token, cosine similarity compares the overall meaning of two entire sentences (or documents) by computing the similarity between their sentence-level embeddings.

The key advantage is that it requires NO reference text. You can compare any two pieces of text and get a similarity score — making it ideal for information retrieval, RAG systems, clustering, and deduplication.

$$\text{Cosine Similarity}(A, B) = (A \cdot B) / (|A| \times |B|)$$

The result is always between -1 and 1, but for sentence embeddings it is typically between 0 and 1:

- **1.0**: identical direction — texts have identical meaning
- **0.7–0.9**: highly similar — same topic, possibly paraphrased
- **0.3–0.6**: moderately related — overlapping concepts
- **0.0–0.3**: unrelated topics

6.2 How It Works in Practice

The process uses a sentence embedding model (like all-MiniLM-L6-v2 from SentenceTransformers) to convert each piece of text into a single dense vector representing its overall meaning, then measures the cosine of the angle between those vectors.

```
# Python — from your code
from sentence_transformers import SentenceTransformer, util


embed_model = SentenceTransformer('all-MiniLM-L6-v2')
# all-MiniLM-L6-v2: fast, lightweight, 384 dimensions — great for production


# convert_to_tensor=True returns a PyTorch tensor for GPU-accelerated comparison
desc_emb = embed_model.encode(description, convert_to_tensor=True)
summ_emb = embed_model.encode(summary, convert_to_tensor=True)

cosine_sim = util.pytorch_cos_sim(desc_emb, summ_emb).item()
# Returns a single float: 0.0 to 1.0
```

6.3 Cosine Similarity vs BERTScore

| Aspect | Cosine Similarity | BERTScore |
|-----------------------|------------------------------------|------------------------------------|
| What is compared | Full sentence/document embedding | Token-by-token embedding matching |
| Reference text needed | No — compares any two texts freely | Yes — must have a reference output |
| Granularity | Sentence-level (holistic meaning) | Word-level (precise alignment) |
| Speed | Fast — one vector per sentence | Slower — full similarity matrix |
| Best for | RAG retrieval, clustering, dedup | Evaluating quality vs ground truth |
| Score range | 0 to 1 for sentence embeddings | 0.8–1.0 typical for good outputs |

 **Use When:** RAG retrieval (finding relevant chunks), comparing a generated summary to the source document without a reference summary, deduplication, and any similarity task where you don't have ground-truth outputs.

 **Remember:** In a RAG pipeline, cosine similarity is the exact metric used by the vector database to find the most relevant chunks. The query embedding and stored chunk embeddings are compared by cosine similarity to retrieve the top-K results.

7. RAGAS — RAG-Specific Evaluation Framework

7.1 Why Traditional Metrics Are Not Enough for RAG

BLEU, ROUGE, and even BERTScore all measure the quality of a generated text compared to a reference. But RAG systems have an additional layer of complexity: the LLM's answer depends on retrieved context. Traditional metrics cannot tell you:

- **Did the LLM stay faithful to the retrieved context, or did it hallucinate?** — Traditional metrics compare output to a reference text, but they have no concept of the retrieved chunks that the LLM was supposed to use.
- **Did the answer actually address the user's question?** — BLEU and ROUGE measure overlap with a reference, not whether the response is relevant to what was asked.
- **Were the retrieved chunks relevant to the question?** — Traditional metrics only evaluate the final generated text; they cannot assess the quality of the retrieval step that preceded it.
- **Did the retrieved chunks contain all the information needed to answer?** — Even if the answer looks good on the surface, the retriever may have missed critical document sections — something BLEU, ROUGE, and BERTScore are completely blind to.

RAGAS (Retrieval-Augmented Generation Assessment) is an evaluation framework specifically designed to answer these four questions about RAG pipelines. It uses an LLM (the judge model) to evaluate these properties rather than simple n-gram counting.

7.2 The Four RAGAS Metrics

1. Faithfulness — Is the Answer Grounded?

Faithfulness measures whether every claim in the generated answer is supported by the retrieved context. A faithful answer never introduces facts that are not in the retrieved documents — it does not hallucinate.

- **High faithfulness (close to 1.0):** every sentence in the answer can be traced back to the retrieved context.
- **Low faithfulness (close to 0.0):** the LLM is making up facts not present in the documents — dangerous for production use.

$$\text{Faithfulness} = \frac{\text{Claims in answer supported by context}}{\text{Total claims in answer}}$$

Example: *Context* says 'Product X was released in 2022.' *Answer* says 'Product X was released in 2023 and won an award.' → Faithfulness is LOW because '2023' and 'won an award' are not in context.

2. Answer Relevancy — Does the Answer Address the Question?

Answer Relevancy measures whether the generated answer actually addresses the user's question. An answer can be faithful to the context but still be irrelevant — for example, answering a completely different aspect of the topic.

- **High relevancy:** the answer directly and completely addresses what was asked.
- **Low relevancy:** the answer is correct but off-topic, or vague and evasive.

Example: *Question:* 'How do I reset my password?' *Answer:* 'Our platform was founded in 2018 and has 50,000 users.' → Relevancy is ZERO — completely ignores the question.

3. Context Precision — Are Retrieved Chunks Useful?


Context Precision measures the proportion of retrieved chunks that are actually useful for answering the question. If you retrieve 5 chunks but only 2 are relevant, context precision is $2/5 = 0.4$. Low precision means the retriever is returning noise alongside signal, forcing the LLM to sort through irrelevant content.

$$\text{Context Precision} = \frac{\text{Relevant retrieved chunks}}{\text{Total retrieved chunks}}$$

4. Context Recall — Were All Needed Chunks Retrieved?

Context Recall measures whether the retrieval system found all the information needed to answer the question. It requires a ground-truth answer to compare against. If the expected answer needs information from 3 specific document sections but the retriever only returned 2, context recall is $2/3 \approx 0.67$.

$$\text{Context Recall} = \frac{\text{Retrieved chunks that support the answer}}{\text{Total chunks needed}}$$


 **Remember:** Context Precision and Recall are about the retriever (FAISS, ChromaDB, etc.), not the LLM. If these are low, the fix is to improve chunking, embeddings, or reranking — not to change the LLM.

7.3 RAGAS Code Snippet

```
from datasets import Dataset
from ragas import evaluate
from ragas.metrics import faithfulness, answer_relevancy,
                        context_precision, context_recall

data = {
    'question': ['How do I reset my password?'],
    'answer': ['Click Settings > Security > Reset Password.'],
    'contexts': [['chunk1 text...', 'chunk2 text...']], # list of lists
    'ground_truth': ['Go to Settings, Security tab, then Reset Password.'],
}
dataset = Dataset.from_dict(data)

results = evaluate(dataset,
    metrics=[faithfulness, answer_relevancy,
             context_precision, context_recall],
    llm=ragas_llm,
    embeddings=ragas_embeddings,
)
print(results.to_pandas())
```

 **Use When:** Evaluating any RAG application: document Q&A bots, knowledge base assistants, customer support systems. Run RAGAS periodically to monitor production quality and detect regressions after updates.

8. DeepEval — Comprehensive Production Evaluation

8.1 What is DeepEval?

DeepEval is an open-source LLM evaluation framework that takes a more comprehensive approach than RAGAS. It provides both built-in metrics (ready to use with configuration) and GEval — a research-backed system where you write evaluation criteria in plain English and an LLM judge scores the output against those criteria (LLM-as-a-judge).

DeepEval evaluates 11 distinct properties across three groups: Safety Checks (where lower is better), Quality Checks (where higher is better), and custom GEval metrics for properties that don't have built-in implementations.

8.2 Safety Checks — Lower Score is Safer

Hallucination — Did the LLM Invent Facts?

DeepEval's Hallucination metric measures whether the generated answer contains factual claims that are not supported by or contradict the retrieved context. Unlike RAGAS Faithfulness, DeepEval also checks for internally contradictory statements within the answer itself.

- **Threshold:** score ≤ 0.3 is acceptable. Higher means more hallucination.
- **If this fails:** the LLM is making up information not in your documents — a critical safety issue.

Toxicity — Is the Response Harmful?

Toxicity checks whether the generated response contains harmful, offensive, threatening, or inappropriate language. This is particularly important for public-facing chatbots where any toxic output could damage reputation or violate platform policies.

- **Threshold:** score ≤ 0.1 . Any toxicity detected above 10% should be investigated.

Bias — Is the Response Unfairly Skewed?

The Bias metric detects whether the generated response shows unjustified demographic, political, religious, or perspective bias. A biased answer might favour one group, political view, or option without factual justification.

- **Threshold:** score ≤ 0.2 . The LLM judge identifies the type of bias and quotes the biased sentence.

8.3 Quality Checks — Higher Score is Better

Answer Relevancy

Same concept as RAGAS Answer Relevancy — does the answer directly address the question asked? DeepEval's judge explains its reasoning, which RAGAS does not provide by default.

Faithfulness

Equivalent to RAGAS Faithfulness — every claim in the answer must be traceable to the retrieved context.

Contextual Relevancy

Measures whether the retrieved chunks are relevant to the question. Equivalent to RAGAS Context Precision but evaluated by an LLM judge rather than a scoring algorithm.

8.4 GEval — Custom LLM-as-a-Judge Metrics

GEval is DeepEval's most powerful feature. It allows you to evaluate any property that doesn't have a built-in metric by writing evaluation criteria in plain English. A judge LLM (Gemini 2.5 Flash in the code) reads your criteria and scores the output from 0.0 to 1.0 with a written explanation.

| GEval Metric | What It Measures | High Score Means | Low Score Means |
|--------------|--|--|--|
| Omission | Fraction of key facts from context included in the answer | Answer covers most important context facts | Important facts were left out of the answer |
| Fairness | Balance and neutrality — all relevant perspectives treated equally | Answer is balanced and unbiased | Answer is one-sided or shows favouritism |
| Completeness | Fraction of the question's sub-parts fully answered | All parts of the question were addressed | Answer is partial — some sub-questions ignored |

Why GEval for These Three?

DeepEval does not have built-in classes for Omission, Fairness, or Completeness. GEval fills this gap by providing a research-backed framework where:

- **You write evaluation criteria and numbered steps in plain English** — no code or ML expertise required to define what "complete" or "fair" means for your specific domain.
- **A judge LLM reads the criteria, analyses the actual output, and assigns a score from 0.0 to 1.0** — the judge reasons about the output against your custom rules, just like a human evaluator would.
- **The judge also produces a written reason explaining its scoring decision** — unlike black-box metrics, GEval tells you exactly *why* a score is low, making it actionable for debugging.
- **The evaluation_steps guide the judge's reasoning to prevent misinterpretation** — numbered steps force the judge to follow a consistent evaluation procedure, avoiding the scoring direction bugs that occur when criteria text is ambiguous.

```
# Example: how a GEval metric is configured (from your code)
from deepeval.metrics import GEval
from deepeval.test_case import LLMTestCaseParams

completeness = GEval(
    name='Completeness',
    criteria=('Measure what fraction of the question parts are',
            'fully answered. High = every part answered.'),
    evaluation_steps=[
        '1. Break the question into sub-questions.',
        '2. For each: FULLY/PARTIALLY/NOT ANSWERED.',
        '3. Score = fully_answered / total_parts.',
    ],
    evaluation_params=[LLMTestCaseParams.INPUT,
                      LLMTestCaseParams.ACTUAL_OUTPUT,
                      LLMTestCaseParams.RETRIEVAL_CONTEXT],
    model=gemini_judge,
    threshold=0.5,
)
```


8.5 Ground Truth Metrics (Optional)

Two additional DeepEval metrics run only when you provide an expected correct answer (ground truth). These are equivalent to RAGAS Context Precision and Recall:

- **Contextual Precision:** verifies that the most relevant chunks are ranked at the top of the retrieval results, not buried at position 4 or 5.
- **Contextual Recall:** verifies that all facts in the expected answer are covered by the retrieved context — no important information was missed by the retriever.

✓ **Use When:** Use DeepEval for production RAG systems that need comprehensive monitoring. The built-in safety checks (hallucination, toxicity, bias) are especially valuable for public-facing applications. GEval is ideal when you have domain-specific quality requirements that don't fit standard metrics.

9. Choosing the Right Metric — Decision Guide

9.1 Full Metrics Comparison

| LLM Evaluation Techniques — Comparison Overview | | | | | |
|---|----------------|--------------------|------------------|-------------------------|------------------------|
| Metric | Type | Needs Ground Truth | Captures Meaning | Best For | Limitation |
| BLEU | n-gram overlap | Yes (ref text) | No | Machine translation | Zero if any n-gram = 0 |
| ROUGE-1/L | n-gram / LCS | Yes (ref text) | No | Summarisation | Ignores synonyms |
| BERTScore | Embedding sim. | Yes (ref text) | Yes (contextual) | Semantic similarity | Needs BERT model |
| Cosine Sim. | Embedding sim. | No (unsupervised) | Yes (semantic) | Clustering, RAG | Needs embed model |
| Faithfulness | LLM-as-judge | No | Yes (deep) | RAG hallucination check | Needs judge LLM |
| Ans. Relevancy | LLM-as-judge | No | Yes (deep) | On-topic check | Needs judge LLM |
| Ctx Precision | LLM-as-judge | Yes (expected) | Yes | Retriever quality | Needs ground truth |
| Ctx Recall | LLM-as-judge | Yes (expected) | Yes | Coverage check | Needs ground truth |
| RAGAS (suite) | LLM framework | Mixed | Yes | Full RAG eval | Slower, LLM cost |
| DeepEval (suite) | LLM framework | Mixed | Yes | Production RAG | Custom GEval needed |

Figure 2: All evaluation techniques compared across 6 key dimensions

9.2 Metric Priority for RAG Systems

From your slides and the DeepEval/RAGAS frameworks, here is the recommended priority order for RAG evaluation in production:

| Priority | Metric | Why This Priority | Formula / Approach |
|----------------|--------------------------|--|---|
| #1 — Critical | Faithfulness | Hallucination risk. If low, the system is making up facts — unacceptable for production. | Claims supported by context / Total claims |
| #2 — High | Answer Relevancy | If the answer doesn't address the question, the system fails its core purpose. | LLM judges whether answer matches question intent |
| #3 — Important | Context Precision | Measures retriever quality — too much noise means LLM wastes context window on garbage. | Relevant chunks / Total retrieved chunks |
| #4 — Important | Context Recall | Missing chunks = incomplete answers. Important for coverage-sensitive domains. | Retrieved supporting chunks / Needed chunks |
| #5 — Safety | Hallucination (DeepEval) | Cross-validates faithfulness with explicit contradiction detection. | LLM judges factual accuracy vs context |
| #6 — Safety | Toxicity & Bias | Essential for public-facing applications and regulated industries. | LLM safety judge |
| #7 — Quality | Omission, Completeness | Ensures answers are thorough, not just technically faithful. | GEval custom criteria |

9.3 Weighted Scoring for Production

In production, you can combine metrics into a single composite score that reflects your application's specific priorities. The weights should match your risk tolerance:

$$\begin{aligned} \text{Answer Quality} &= (\text{Faithfulness} + \text{Answer Relevancy}) / 2 \\ \text{Retriever Quality} &= (\text{Context Precision} + \text{Context Recall}) / 2 \end{aligned}$$

For a weighted composite — adjust weights based on your domain:

$$\begin{aligned} \text{Weighted Score} &= 0.40 \times \text{Faithfulness} + 0.30 \times \text{Answer Relevancy} \\ &+ 0.15 \times \text{Context Precision} + 0.15 \times \text{Context Recall} \end{aligned}$$

| Domain | Faithfulness Weight | Answer Relevancy | Context Precision | Context Recall | Reason |
|--------------------|---------------------|------------------|-------------------|----------------|---|
| Healthcare / Legal | 0.50 | 0.25 | 0.15 | 0.10 | Hallucination is dangerous — highest faithfulness weight |
| Customer Support | 0.40 | 0.35 | 0.15 | 0.10 | Relevancy is critical — users need their question answered |
| Research Assistant | 0.30 | 0.25 | 0.20 | 0.25 | Recall important — missing facts leads to incomplete research |
| General Chatbot | 0.40 | 0.30 | 0.15 | 0.15 | Balanced weights for general-purpose assistant |

9.4 When to Use Which Metric — Quick Decision Guide

| Situation | Recommended Metric(s) | Avoid |
|--------------------------------------|--|--|
| Machine translation evaluation | BLEU + BERTScore | ROUGE (not recall-focused) |
| Summarisation quality | ROUGE-1, ROUGE-L, BERTScore | BLEU (precision-focused) |
| Semantic similarity, no ground truth | Cosine Similarity (sentence embeddings) | BLEU, ROUGE (need reference) |
| RAG hallucination detection | RAGAS Faithfulness, DeepEval Hallucination | BLEU, ROUGE (don't understand context) |
| RAG production monitoring | Full RAGAS or DeepEval suite | BLEU (misses RAG-specific failures) |
| Public chatbot safety | DeepEval Toxicity + Bias + Hallucination | BLEU, ROUGE (no safety coverage) |

| | | |
|---------------------------------------|--|----------------------------------|
| Custom domain quality rules | DeepEval GEval (write your own criteria) | Fixed metrics only |
| Fast, low-cost eval without LLM judge | ROUGE + BERTScore (no API calls needed) | RAGAS, DeepEval (need judge LLM) |

9.5 Quick Revision — Key Definitions

| Term | One-Line Definition |
|-------------------|---|
| Precision | Of predicted positives, fraction that were actually positive — quality of alarms |
| Recall | Of actual positives, fraction that were correctly detected — coverage of true cases |
| F1 Score | Harmonic mean of Precision and Recall — punishes extreme imbalances |
| BLEU | N-gram precision: fraction of generated n-grams that appear in reference text |
| ROUGE | N-gram recall: fraction of reference n-grams that appear in generated text |
| ROUGE-L | LCS-based score: longest ordered sequence of words shared between two texts |
| BERTScore | Token-level semantic similarity using BERT embeddings — understands synonyms |
| Cosine Similarity | Angle-based similarity between two embedding vectors — 1.0 = identical direction |
| Faithfulness | RAGAS/DeepEval: fraction of answer claims supported by retrieved context |
| Answer Relevancy | RAGAS/DeepEval: does the answer actually address the user's question? |
| Context Precision | RAGAS/DeepEval: fraction of retrieved chunks that are relevant to the question |
| Context Recall | RAGAS/DeepEval: fraction of needed information that was actually retrieved |
| RAGAS | RAG evaluation framework with 4 core metrics: faithfulness, relevancy, precision, recall |
| DeepEval | Comprehensive LLM eval framework with safety, quality, and GEval custom metrics |
| GEval | LLM-as-a-judge: write criteria in English, judge LLM scores output 0.0–1.0 |
| LLM-as-a-judge | Using a capable LLM (Gemini, GPT-4) to evaluate another LLM's output quality |
| Hallucination | When an LLM generates factually incorrect content not supported by source documents |
| Smoothing (BLEU) | Adding 1 to numerator/denominator so zero n-gram matches don't collapse score to 0 |
| Soft Alignment | BERTScore's greedy matching: each token paired with its most semantically similar counterpart |