DeepEval Documentation

Overview

DeepEval is an open-source evaluation framework for Large Language Models (LLMs), built by Confident AI. It simplifies building, testing, and iterating on LLM applications by treating evaluations like unit tests (similar to Pytest). It supports over 50 research-backed metrics, multi-modal evaluations, end-to-end and component-level testing, synthetic dataset generation, customizable metrics, and red teaming for security. Evaluations run locally, with optional cloud integration via Confident AI for centralized reporting, regression testing, and monitoring. It's ideal for use cases like RAG, AI agents, and chatbots.

Installation and Setup

- Install via pip: `pip install -U deepeval`.
- Runs locally; requires an LLM API key (e.g., `OPENAI_API_KEY`) for LLM-as-a-Judge metrics.
- Supports providers like OpenAI, Ollama, Azure OpenAI, Anthropic, Gemini, or custom LLMs.
- For cloud features: Sign up at app.confident-ai.com (free), set `CONFIDENT_API_KEY` environment variable.
- Quick setup: Create a virtual environment, install, and configure API keys.

Key Features

- **Unit Testing Style**: Pytest-like interface for asserting LLM outputs.
- **Metrics Library**: 50+ pre-built metrics (e.g., correctness, relevancy, hallucination, safety).
- **Evaluation Modes**:
    - End-to-end (black-box): Tests overall inputs/outputs.
    - Component-level (white-box): Uses tracing for individual modules.
- **Tracing**: Non-intrusive `@observe` decorator for observability in development and production.
- **Dataset Generation**: Synthetic data using evolution techniques.
- **Custom Metrics**: Define criteria for GEval or build new ones.
- **Red Teaming**: DeepTeam scans for 50+ vulnerabilities and 20+ attack methods.
- **Regression Testing**: Compare runs via Confident AI.
- **Multi-Modal Support**: Evaluate text, images, etc.

Available Metrics and Evaluation Methods

- **Core Metrics**:
  - GEval: General-purpose; scores based on custom criteria (e.g., correctness using actual vs. expected output).
  - AnswerRelevancyMetric: Checks output relevance to input.
  - Others: Hallucination, faithfulness, bias, toxicity, etc. (research-backed like G-Eval).
- **Methods**:
  - LLM-as-a-Judge (default; local or via providers).
  - Threshold-based pass/fail (e.g., score > 0.5).
  - Supports no expected output for some metrics.
  - Datasets: Use `EvaluationDataset` and `Golden` for test cases.
- Evaluations can be run via CLI (`deepeval test run`) or in code.

Usage Examples

- **End-to-End Test**:

```
Python

from deepeval import assert_test
from deepeval.test_case import LLMTestCase
from deepeval.metrics import GEval
from deepeval.test_case import LLMTestCaseParams

correctness_metric = GEval(
    name="Correctness",
    criteria="Determine if the actual output is correct based on the expe
    evaluation_params=[LLMTestCaseParams.ACTUAL_OUTPUT, LLMTestCaseParams
)

test_case = LLMTestCase(
    input="What is the capital of Japan?",
    actual_output="Tokyo",
    expected_output="Tokyo"
)

assert_test(test_case, [correctness_metric])
```

Run: `deepeval test run test_file.py`.

- **Component-Level with Tracing**:

```
Python

from deepeval.tracing import observe
from deepeval.metrics import AnswerRelevancyMetric

@observe(metrics=[AnswerRelevancyMetric()])
def rag_component(query: str):
    # Simulate RAG logic
    return "Retrieved answer"

rag_component("Query here")
```

Integrations

- LLM Providers: OpenAI, Ollama, Anthropic, etc.
- Cloud: Confident AI for dashboards, monitoring, and CI/CD.
- Compatible with Python environments; examples on GitHub.

Pricing and Licensing

- Open-source (free to use/install).
- Confident AI cloud: Free tier for basic usage (storage, reports); no paid details specified, but enterprise features implied.

Other Details

- Community: GitHub repo with examples; 10k+ stars.
- Limitations: Relies on external APIs for metrics; tracing requires decorators.
- Updates: Active, with focus on research-backed enhancements.

Arize AI Documentation

Overview

Arize AI is an observability and evaluation platform for LLM applications, including the Phoenix Evals library (open-source). It focuses on building, implementing, and optimizing eval strategies using LLM-as-a-Judge, code-based, or human annotations. It's suited for prompt iteration, agent evaluation, and production monitoring. Key emphasis: Scalable evals without constant human input, with benchmarking against golden datasets.

Installation and Setup

- Phoenix Evals (OSS part): Install via pip (`pip install arize-phoenix` or similar; inferred from related docs).
- Full platform: Sign up at arize.com; integrates with tracing tools.
- Setup: Define eval templates, input data, and run via Python/TypeScript SDKs.
- Requires LLM API keys for judges.

Key Features

- **Eval Types**: LLM Judge (qualitative), Code (objective/low-cost), Human Annotations (expert feedback).
- **Templates**: Pre-built for hallucination, user frustration, agent planning, function calling.
- **Online/Offline Evals**: Real-time on traces or batch on datasets.
- **Aggregation**: Summarize quality across prompts/LLMs.
- **Benchmarking**: Use golden datasets for accuracy metrics (precision, recall, F1).
- **Human Queues**: Manage annotations and golden dataset creation.
- **Composability**: Lightweight building blocks for custom evals.

Available Metrics and Evaluation Methods

- **Metrics**: Relevance, hallucination, latency, positivity/negativity, etc.; aggregations (mean, etc.).
- **Methods**:
    - LLM-as-a-Judge: LLM evaluates outputs with explanations.
    - Code Evals: Programmatic checks (e.g., regex for patterns).
    - Annotations: Human labeling for ground truth.
    - Templates: Custom prompts defining criteria/output formats (e.g., binary labels).
- Supports thousands of instances scalably.

Usage Examples

- **Custom Template**:

```
Python

template = """
You are evaluating the positivity or negativity of responses.
[Question]: {question}
[Response]: {response}
Answer with 'positive' or 'negative'.
"""
# Use in eval function with dataset
```

- **Benchmarking**: Run template on golden dataset with ground truth; compute F1 scores.

Integrations

- SDKs: Python and TypeScript.
- Traces: Compatible with OpenTelemetry or Arize tracing.
- Tools: LLM providers for judges.

Pricing and Licensing

- Phoenix Evals: Open-source (free).
- Full Arize Platform: Freemium model (basic free; enterprise pricing on site, e.g., for advanced monitoring).

Other Details

- Community: Docs at docs.arize.com; research papers hub.
- Limitations: Human evals costly; LLM judges weak on quantitative/expert tasks.
- Updates: Regular, with guides like "Definitive Guide to LLM Evaluation".

Opik Documentation

Overview

Opik, by Comet, is an open-source platform for LLM observability, evaluation, and optimization. It logs traces, evaluates outputs, and monitors apps from prototype to production. Focus: Systematic testing with datasets, metrics, and experiments; supports RAG, agents, etc. Includes UI for analysis and collaborative scoring.

Installation and Setup

- Install SDK: `pip install opik` (Python); npm for TypeScript.
- Setup: Create client (`opik.Opik()`), define datasets, and run evals.
- Local or cloud-hosted; integrates with Comet platform.

Key Features

- **Evaluation Components**: Datasets (inputs/outputs), tasks (e.g., prompt templates), metrics.
- **Pre-Built Metrics**: Hallucination, Answer Relevance, Context Precision/Recall.
- **Methods**: `evaluate_prompt` for prompts; `evaluate` for complex apps.
- **Online Eval**: Rules for auto-scoring production traces.
- **Multi-Value Feedback**: Collaborative scoring with aggregation (mean, std, etc.).
- **UI**: View experiments, compare runs, dashboards.
- **Tracing**: Comprehensive for debugging.

Available Metrics and Evaluation Methods

- **Metrics**: Hallucination, Relevance, Precision/Recall; LLM-as-a-Judge or heuristics.
- **Methods**: Prompt eval, thread eval, complex app eval; aggregate scores for stats.

Usage Examples

- **Prompt Eval**:

```
Python

import opik
from opik.evaluation.metrics import Hallucination

client = opik.Opik()
dataset = client.get_or_create_dataset("MyDataset")
dataset.insert([{"input": "Query", "expected_output": "Answer"}])

evaluate_prompt(dataset=dataset, messages=[{"role": "user", "content": "{
```

- **Score Aggregation**:

```
Python

scores = result.aggregate_evaluation_scores()
```

Integrations

- SDKs: Python/TypeScript.
- Tools: LLM providers; Comet ML for full observability.

Pricing and Licensing

- Open-source (free; GitHub: github.com/comet-ml/opik).
- Comet Platform: Free tier; paid for enterprise scalability.

Other Details

- Community: Docs at comet.com/docs/opik; 17k+ GitHub stars.
- Limitations: Requires datasets for evals; UI-dependent for deep analysis.
- Updates: Active, with new features like multi-value scoring.

Comparison: Pros and Cons

Framework	Pros	Cons
DeepEval	- Extensive 50+ research-backed metrics. - Pytest-like ease for testing. - Strong on customization, red teaming, and synthetic data. - Free cloud integration for regression/monitoring. - OSS with local runs.	- Heavy reliance on API keys for metrics. - Tracing requires code changes. - Potential scalability limits locally.
Arize	- Flexible eval methods (LLM/code/human). - Pre-built templates and benchmarking. - Scalable for production; good for observability. - Supports annotations for expert input. - OSS Phoenix for core evals.	- Human annotations expensive/time-consuming. - LLM judges limited on quantitative tasks. - Less focus on pure unit testing.
Opik	- Collaborative scoring and UI dashboards. - Good for tracing and production monitoring. - Pre-built metrics for common tasks. - Supports complex apps/agents. - OSS with Python/TS SDKs.	- Fewer metrics than DeepEval. - Dataset-heavy setup. - Less emphasis on customization or red teaming.

Recommendation

Based on the research, I recommend **DeepEval** if your focus is on in-depth, customizable LLM evaluations with a testing mindset (e.g., unit tests, many metrics, red teaming). It's the most straightforward for developers iterating quickly, with strong OSS roots and free cloud support—ideal if you've watched the videos and want something video-aligned (DeepEval emphasizes ease like Pytest).

If you need broader observability (tracing, production monitoring) or human-in-the-loop, go with **Arize** for its flexible methods and enterprise polish. **Opik** is a solid middle ground for collaborative teams with UI needs, especially if integrated with Comet ML.

Choose DeepEval to start, as it has the most metrics and research backing, balancing pros without major cons for most eval workflows.

Structured Documentation (PPT-Style Markdown)

Slide 1: Title

LLM Evaluation Frameworks: DeepEval, Arize, Opik
In-Depth Research and Recommendation
Date: January 20, 2026

Slide 2: Agenda

- Framework Overviews
- Detailed Features & Usage
- Pros/Cons Comparison
- Recommendation
- Resources

Slide 3: DeepEval Overview

- OSS framework for LLM evals.
- Key: 50+ metrics, tracing, red teaming.
- Install: `pip install -U deepeval`.
- Usage: Pytest-style tests.

Slide 4: DeepEval Metrics & Examples

- Metrics: GEval, Relevancy, Hallucination.
- Example: Assert test case with criteria.
- Pros: Customizable, free cloud.
- Cons: API dependency.

Slide 5: Arize Overview

- Platform for LLM observability & evals.
- Key: LLM/Code/Human methods, templates.
- Install: Phoenix OSS via pip.
- Usage: Custom eval prompts.

Slide 6: Arize Metrics & Examples

- Metrics: Relevance, Hallucination, Aggregations.
- Example: Tone classification template.
- Pros: Scalable, benchmarking.
- Cons: Costly human annot.

Slide 7: Opik Overview

- OSS for eval & optimization.
- Key: Datasets, metrics, UI.
- Install: `pip install opik`.
- Usage: Evaluate prompts/datasets.

Slide 8: Opik Metrics & Examples

- Metrics: Hallucination, Relevance.
- Example: SDK eval with aggregation.
- Pros: Collaborative, production-ready.
- Cons: Dataset-focused setup.

Slide 9: Pros/Cons Table

(Insert table from above)

Slide 10: Recommendation

- Top Pick: DeepEval (easy, metric-rich).
- Alternatives: Arize for observability; Opik for teams.
- Why: Balances depth, cost, and usability.

Slide 11: Resources

- DeepEval: deepeval.com/docs
- Arize: arize.com/docs
- Opik: comet.com/docs/opik
- GitHubs for code/examples.

Slide 12: Q&A

Thank you! Any questions?