Practical Guide: Hands-On Local LLM Evaluation with Arize Phoenix

Overview

Based on my practical experience, I am going to walk through setting up a complete local LLM evaluation pipeline using Arize Phoenix, Ollama and other tools. By the end, you'll have a reproducible evaluation workflow that runs entirely on your machine with no external APIs required.

Tools Used

- **Ollama** - Local LLM runtime (running Phi model)
- **Arize Phoenix** - Evaluation and observability platform
- **OpenTelemetry** - Instrumentation and data collection
- **Cursor IDE** - AI-powered development environment
- **Cline** - AI assistant for terminal operations
- **Python 3.8+** - Runtime environment

Prerequisites & Setup

1. Install Core Dependencies

Install Ollama:

1. Visit https://ollama.ai and download the installer for your operating system
2. Run the installer and follow the setup wizard
3. Open terminal/command prompt and verify installation: ollama --version

Download Phi:

1. In terminal, run: ollama pull phi
2. Verify with: ollama list (should show phi in the list)

Install Python Dependencies:

1. Create and activate a new Python virtual environment
2. Install required packages: pip install arize-phoenix opentelemetry-api opentelemetry-sdk requests

Install Arize Phoenix latest/updated version

Ensure Phoenix version alignment with OpenTelemetry:

- Phoenix v4.0+ requires opentelemetry-api >= 1.20.0
- Check compatibility by running: pip show arize-phoenix opentelemetry-api in terminal

2.Development Environment

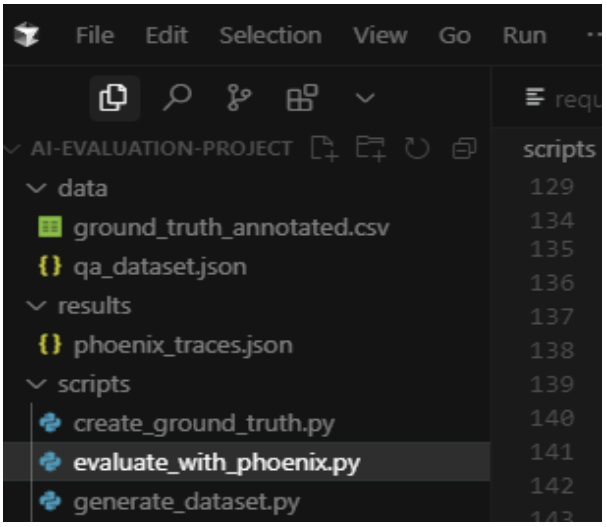
Install Cursor IDE:

1. Install and launch Cursor
2. Install the Cline extension for AI-assisted terminal operations

Project Structure

Step 1: Create main project folder called AI-evaluation-project

Step 2: Inside that folder, create a **scripts** subfolder
Step 3: Your final structure should look like:
AI-evaluation project -> Scripts-> Ground Truth, Dataset and Evaluation (3 files)



Implementation: Step-by-Step

1. Dataset Generation Script (scripts/dataset.py)

Purpose: Creates Q&A pairs using your local Phi model for evaluation testing.

Key Components:

- Predefined Questions: Set of 5 evaluation questions covering different topics (geography, ML concepts, science)
- Ollama Integration: Connects to local Phi model via HTTP API calls
- Answer Generation: Sends each question to Phi and collects responses
- Rate Limiting: Includes delays to prevent overwhelming the local model
- Error Handling: Manages connection issues and timeout scenarios

Output: Complete dataset with questions, generated answers, and timestamps for tracking.

2. Ground Truth Script (scripts/ground_truth.py)

Purpose: Provides reference answers for evaluation comparison against Phi's responses.

Key Components:

- Reference Answers: Manually crafted correct answers for each evaluation question
- Answer Retrieval: Functions to fetch ground truth for specific questions
- Display Methods: Formatted output showing all reference answers
- Quality Standards: Well-researched, concise answers that serve as evaluation benchmarks

Sample Questions & References:

- "What is the capital of France?" → "Paris"
- "Explain machine learning" → Comprehensive but concise explanation

3. Evaluation Script (scripts/evaluation.py)

Purpose: Core evaluation engine that compares Phi's responses to ground truth and sends results to Phoenix as it needs inputs, outputs and reference.

Key Components:

Evaluation Methods:

- Exact Match: Direct comparison between generated and reference answers
- Semantic Similarity: Word overlap analysis
- Custom Scoring: Business logic for different question types

Phoenix Integration:

- Session management and UI launch
- OpenTelemetry trace configuration
- Project setup and data streaming

Data Processing:

- Combines dataset generation with ground truth comparison
- Creates structured evaluation records with scores and explanations
- Generates Phoenix-compatible traces with metadata

Arize Phoenix UI Integration:

- Automatic session startup
- Real-time trace streaming
- Performance metrics collection

Running the Complete Pipeline – Step by Step process

1. Execute via Cline / Command Prompt

1. Open your AI-evaluation project folder on Cursor IDE
2. Use Cline to execute the following commands

- Run: python scripts/dataset.py in the integrated terminal
- Run the ground_truth.py script to show reference answers

Run Complete Evaluation:

- Finally run and execute the evaluation.py script to start the Phoenix evaluation

2. Expected Output

The evaluation script will:

1. Start Phoenix session and display UI URL local host
2. Generate Q&A pairs using local Phi model
3. Compare against ground truth answers
4. Send traces and evaluations to Arize Phoenix
5. Display summary statistics

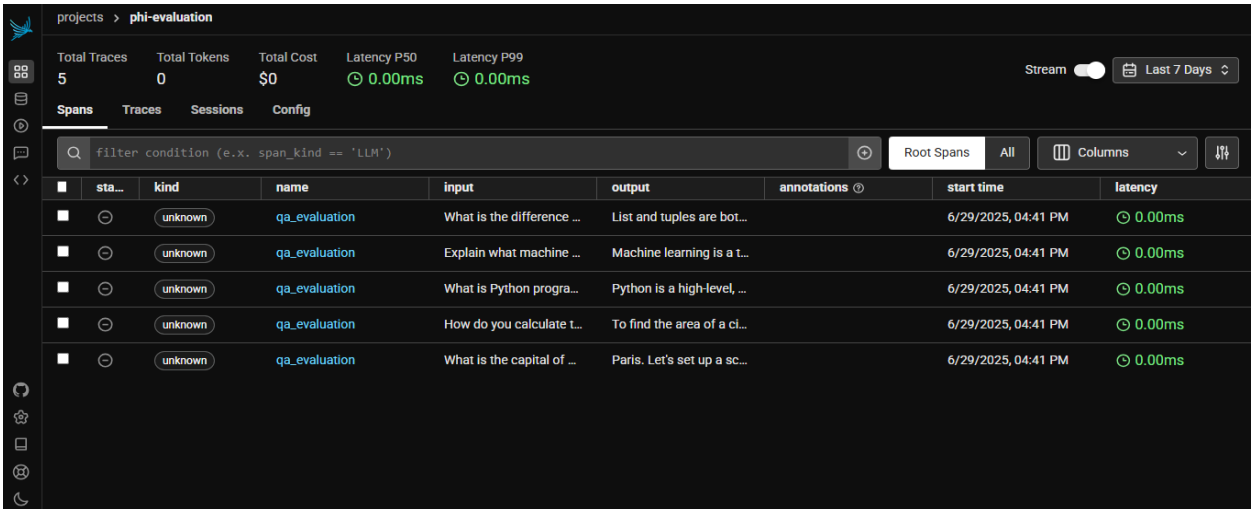
```
PS C:\Users\ramya\ai-evaluation-project> python scripts\phoenix_evaluation.py
🚀 Starting Phoenix Evaluation...
🔥 Starting Phoenix session...
❗ The launch_app 'port' parameter is deprecated and will be removed in a future release. Use the 'PHOENIX_PORT' environment v
C:\Users\ramya\anaconda3\Lib\contextlib.py:144: SAWarning: Skipped unsupported reflection of expression-based index ix_cumulat
tal
```

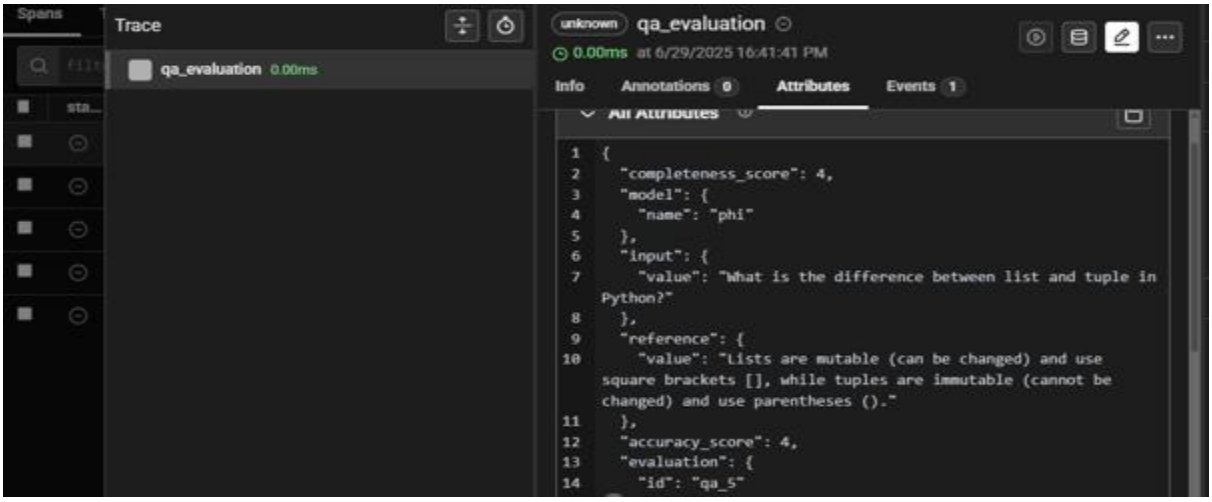
```
Command Prompt - ollama ru | Windows PowerShell | Windows PowerShell | Windows PowerShell

  next(self.gen)
C:\Users\ramya\anaconda3\Lib\contextlib.py:144: SAWarning: Skipped unsupported reflection of expression-based index ix_latency
  next(self.gen)
🌍 To view the Phoenix app in your browser, visit http://localhost:6006/
📖 For more information on how to use Phoenix, check out https://arize.com/docs/phoenix
✅ Phoenix UI available at: http://localhost:6006
OpenTelemetry Tracing Details
|  Phoenix Project: phi-evaluation
|  Span Processor: SimpleSpanProcessor
|  Collector Endpoint: http://localhost:6006/v1/traces
|  Transport: HTTP + protobuf
|  Transport Headers: {}
|
|  Using a default SpanProcessor. `add_span_processor` will overwrite this default.
|
|  WARNING: It is strongly advised to use a BatchSpanProcessor in production environments.
|
|  `register` has set this TracerProvider as the global OpenTelemetry default.
|  To disable this behavior, call `register` with `set_global_tracer_provider=False`.

✅ Loaded 5 evaluation records
📊 Creating evaluation traces...
✅ Traces created and sent to Phoenix

🎉 Evaluation traces sent to Phoenix!
📱 Open http://localhost:6006 to view your evaluation
🔍 Look for the 'phi-evaluation' project
📊 You should see 5 traces with your Q&A evaluations

📋 SUMMARY:
   Questions evaluated: 5
   Average relevance: 5.0/5
   Average accuracy: 4.0/5
```

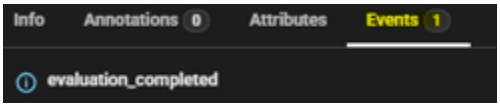
3. Phoenix UI Exploration

Navigate to http://localhost:6006 to explore:

- **Projects**: View your evaluation project
- **Traces**: Inspect individual Q&A evaluations
- **Metrics**: Overall performance statistics
- **Latency**: Model response time analysis
- **Datasets**: Browse questions and answers

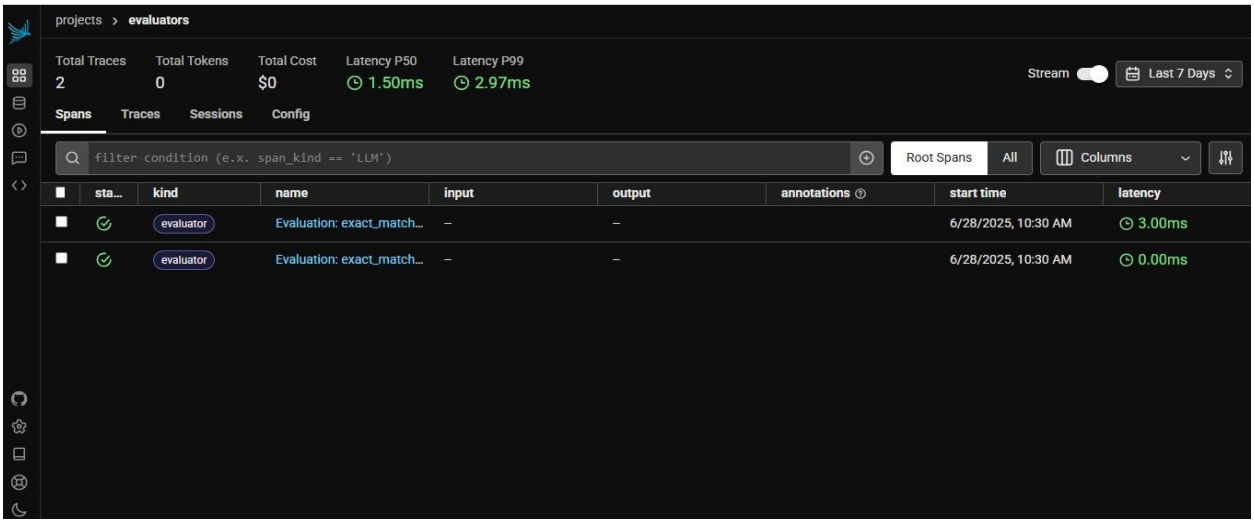


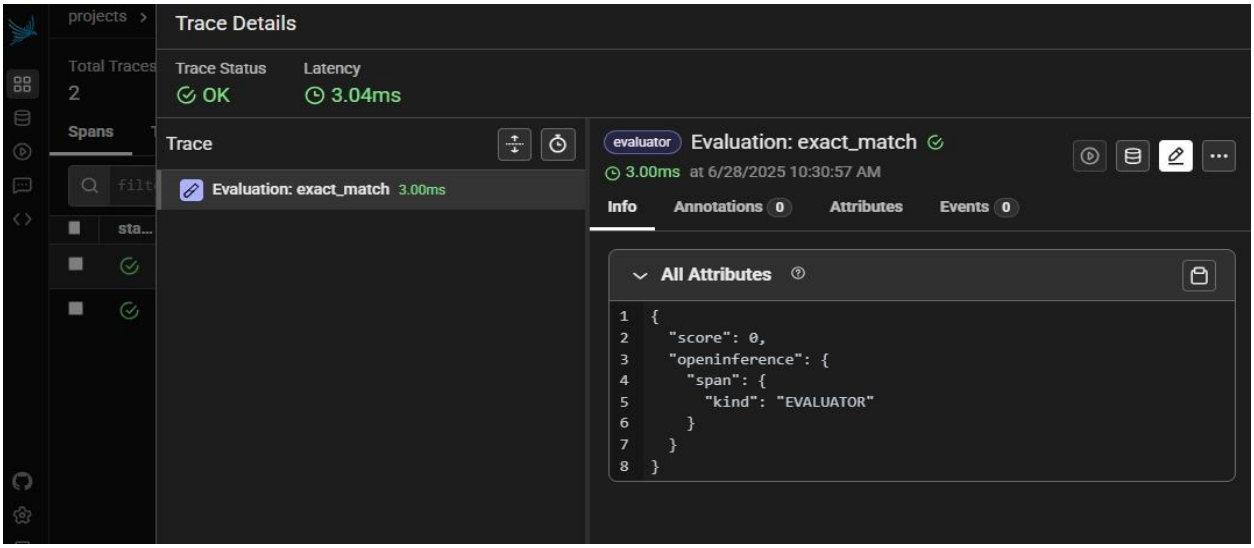




An **alternative workflow** I used:

1. It's similar to all the above process steps and could be a simpler set up where Phoenix directly called Phi LLM and got real traces giving an OK status on Phoenix
2. Create a CSV file that has inputs(questions) and expected outputs (ground truth answers) together in one file.
3. Launch Arize Phoenix manually, open a default project, create a dataset and do mapping of input and output.
4. Configure the Evaluation Script following Phoenix's predefined evaluation template, to call your local LLM model phi to generate the LLM's answers and also define the exact match eval criteria in the python script/code.
5. Run and execute this evaluation on Cline, open Phoenix UI
6. Arize Phoenix processes the dataset and displays task runs, scores, and trace-level insights as you can see below.





Evaluation Modes

Batch evaluation involves processing a static dataset consisting of predefined inputs and reference outputs. By comparing the model's responses against the ground truth answers, you can calculate metrics. This evaluation is typically run in bulk once, giving you a snapshot of the model's performance across the dataset. It's an ideal approach for validating performance early in development or testing new model versions. But it doesn't capture real-time dynamics and might miss important feedback from continuously evolving inputs.

Live evaluation, on the other hand, allows for real-time interaction with the model. Here, the LLM makes predictions based on dynamic inputs, and these are immediately compared to reference answers, with results streamed continuously to **Phoenix**. This process mimics real-world agent behavior, making it invaluable for monitoring agents in production environments. It ensures that the model's performance is continuously measured, and any deviations or issues are caught early. Live evaluation, however, demands robust instrumentation and monitoring, and can be more challenging to control.

I hope this write-up is helpful for anyone trying to run an AI evaluation locally for the first time. Whether you're exploring LLMs out of curiosity or preparing for real-world deployment, this guide aims to make the process more approachable and practical. Special thanks to **Arize Phoenix** for being such a **powerful and intuitive tool** in this journey, it has truly streamlined the evaluation and observability process, and I'm excited to keep exploring its potential in AI model performance and agent monitoring!