1.3 LIGHTWEIGHT EVALUATION & SAFETY

As LLMs become central to your applications, you need ways to measure their performance and ensure they behave safely. This module covers practical evaluation strategies and responsible AI considerations.

LEARNING OBJECTIVES

By the end of this module, you will:

- Understand why evaluation matters for LLM applications
- Know basic evaluation signals for assessing LLM outputs
- Be aware of safety and responsible use considerations
- Understand how evaluation integrates with the accelerator

WHY EVALUATION MATTERS

THE PROBLEM

LLMs are powerful but unpredictable. Without evaluation, you risk:

Hallucinations: Model generates plausible but false information

```
Q: "What is IBM's revenue in 2025?"
Bad Answer: "IBM's revenue in 2025 was $87.4 billion." [Made up number]
```

Inconsistent Answers: Same question, different responses

```
Monday: "The capital of Australia is Sydney."
Tuesday: "The capital of Australia is Canberra." [Correct]
```

Business Impact: Wrong answers can lead to: - Lost customer trust - Compliance violations - Financial losses - Reputational damage

THE SOLUTION

Systematic evaluation helps you: - Catch problems before they reach users - Compare different models or prompts - Track quality over time - Build confidence in your system

SIMPLE EVALUATION SIGNALS

1. CORRECTNESS (GROUND TRUTH COMPARISON)

What it is: Does the answer match known correct information?

How to measure:

```
def evaluate_correctness(model_answer: str, ground_truth: str) -> int:
    """Returns 1 if correct, 0 if incorrect"""
    # Simple exact match
    if model_answer.strip().lower() == ground_truth.strip().lower():
        return 1

    # Or use semantic similarity
    similarity = compute_similarity(model_answer, ground_truth)
    return 1 if similarity > 0.8 else 0
```

Example:

```
Question: "What year was IBM founded?"
Model Answer: "1911"
Ground Truth: "1911"
Score: 1 (Correct)

---

Question: "What year was IBM founded?"
Model Answer: "1920"
Ground Truth: "1911"
Score: 0 (Incorrect)
```

When to use: When you have known correct answers (test sets, benchmarks)

SAFETY & RESPONSIBLE USE

POTENTIAL RISKY CATEGORIES

LLMs can potentially generate harmful content in these categories:

1. **Personal Information (PII)**
   - Social security numbers, credit cards, passwords
   - Addresses, phone numbers, email addresses
2. **Harmful Content**
   - Hate speech, discrimination, harassment
   - Violence, self-harm, dangerous activities
   - Illegal activities, fraud schemes
3. **Misinformation**
   - Medical advice without disclaimers
   - Financial advice as fact
   - False claims about public figures
4. **Bias and Fairness**
   - Stereotyping based on protected attributes
   - Unfair treatment of groups
   - Lack of representation

HOW THIS TIES INTO THE ACCELERATOR

EVALUATION ENTRY POINTS

1. `tools/eval_small.py`

Purpose: Run a small evaluation dataset through your RAG system

What you'll implement on Day 2-3:

```
# tools/eval_small.py

import pandas as pd
from accelerator.rag.pipeline import RAGPipeline

def evaluate_rag_system(test_file: str, output_file: str):
    """
    Evaluate RAG system on a test set

    Args:
        test_file: CSV with columns [question, ground_truth, category]
        output_file: Where to save results
    """
    # Load test data
    df = pd.read_csv(test_file)

    # Initialize pipeline
    pipeline = RAGPipeline()

    results = []
    for idx, row in df.iterrows():
        # Generate answer
        response = pipeline.answer_question(row['question'])
```

2. `accelerator/assets/notebook/Analyze_Log_and_Feedback.ipynb`

Purpose: Analyze logs and user feedback from the production service

What it does: - Load logs from `service/api.py` - Analyze patterns: - Most common questions - Average

HOW THIS CONNECTS TO LAB 1.3

WHAT YOU'LL BUILD

In Lab 1.3, you'll create a **micro-evaluation framework**:

1. **Test set**: 5-10 diverse prompts
2. **Data collection**: Run prompts through Ollama and watsonx
3. **Rating rubric**: Apply manual ratings (correctness, clarity, style)
4. **Analysis**: Compare backends, identify patterns

EXAMPLE OUTPUT (DATAFRAME)

prompt	backend	answer	correctness	clarity	style_match
"Summarize AI…"	ollama	"AI is…"	4	5	4
"Summarize AI…"	watsonx	"Artificial…"	5	5	5
"Extract emails…"	ollama	"john@…"	5	4	5

SKILLS YOU'LL DEVELOP

- Programmatic evaluation loops
- Rating rubric design
- Comparative analysis

REFERENCE NOTEBOOKS

GOVERNANCE & EVALUATION

`ibm-watsonx-governance-evaluation-studio-getting-started.ipynb`: - Shows watsonx.governance evaluation features - Demonstrates automated evaluation at scale - Connects to compliance tracking

From this, you'll learn: - How to structure evaluation datasets - Metrics that matter for enterprise applications - Integration with governance workflows

Progression:

```
Lab 1.3 (Manual, 10 questions)
    ↓
eval_small.py (Automated, 100 questions)
    ↓
Evaluation Studio (Continuous, production scale)
```

BEST PRACTICES FOR EVALUATION

✅ DO

1. **Start simple:** Don’t over-engineer evaluation initially
2. **Automate what you can**: Manual review doesn’t scale
3. **Track over time**: Evaluation is ongoing, not one-time
4. **Test edge cases**: Don’t just test happy paths
5. **Involve stakeholders**: Domain experts should validate quality
6. **Version everything:** Track prompts, models, and test sets

❌ DON’T

1. **Skip evaluation**: “It looks good” isn’t enough
2. **Rely solely on accuracy**: Context matters (latency, safety, cost)
3. **Forget about drift**: Models and data change over time
4. **Ignore user feedback**: Real usage reveals issues testing doesn’t
5. **Over-optimize for metrics**: Gaming metrics != real quality

EVALUATION MATURITY MODEL

LEVEL 1: AD HOC

- Manual testing with a few examples
- “Looks good to me” approval
- No systematic tracking

LEVEL 2: BASIC (← LAB 1.3 TARGETS THIS)

- Small test set (10-50 examples)
- Manual rubric
- Occasional re-evaluation

LEVEL 3: SYSTEMATIC (← `eval_small.py` TARGETS THIS)

- Curated test set (100-500 examples)
- Automated metrics where possible
- Regular evaluation runs
- Version control for prompts and results

LEVEL 4: CONTINUOUS (← PRODUCTION GOAL)

- Large test set + production monitoring
- Automated evaluation pipeline

KEY TAKEAWAYS

- **Evaluation is essential**: Don't deploy LLMs without systematic quality checks
- **Start simple**: Basic metrics beat no metrics
- **Safety first**: Proactively mitigate risks with guardrails
- **Iterate**: Evaluation frameworks evolve with your application
- **Automate**: Scale evaluation as your system scales

Next: Let's build your first evaluation framework in Lab 1.3!