📊 Module 5: Evaluation and Optimization

> ***Weeks 13-14*** | *Measuring and improving prompt performance*

Week 13: Testing and Evaluation

✏️ Defining Success Metrics

Theory: Before optimizing prompts, define what "good" means for your use case.

Common Metrics:

Metric	What It Measures	Example
Accuracy	Correctness of outputs	95% of classifications correct
Relevance	Output matches intent	4.5/5 relevance score
Consistency	Same input → similar output	<10% variance
Latency	Response time	<2 seconds
Token Efficiency	Tokens used	<500 tokens per response
Safety	No harmful content	0 policy violations

Metric Selection Framework:

```
For my prompt, the key metrics are:

Primary (Must have):
- [ ] Accuracy: The output must be correct
- [ ] Safety: No harmful/inappropriate content

Secondary (Important):
- [ ] Consistency: Similar inputs give similar outputs
- [ ] Tone: Matches brand voice

Tertiary (Nice to have):
- [ ] Token efficiency: Minimize costs
- [ ] Creativity: Unique responses
```

Example: Customer Service Bot Metrics

```
Metric Definition for CS Bot:

1. Resolution Rate (Primary)
   - Definition: % of queries resolved without escalation
   - Target: >80%
   - Measurement: Track "resolved" vs "escalated" tags

2. Accuracy (Primary)
   - Definition: Information provided is correct
```

```
   - Target: >95%
   - Measurement: Sample review by QA team

3. Tone Compliance (Secondary)
   - Definition: Matches brand voice guidelines
   - Target: >90%
   - Measurement: Rubric scoring on samples

4. Response Time (Tertiary)
   - Definition: Time to generate response
   - Target: <3 seconds
   - Measurement: API latency logging
```

🔬 A/B Testing Prompts

Theory: Compare prompt versions to find what works best.

A/B Test Setup:

```
=== PROMPT VERSION A (Control) ===
Summarize this article in 3 bullet points.

<article>
{article_text}
</article>

=== PROMPT VERSION B (Test) ===
You are an expert editor. Read this article carefully
and create exactly 3 bullet points that capture the
most important information a reader should know.

<article>
{article_text}
</article>

Ensure each bullet is:
- Clear and standalone
- Under 25 words
- Information-rich
```

Testing Framework:

```
A/B Test Plan:

Test Name: "Detailed Instructions vs Minimal"
Hypothesis: "Adding explicit requirements improves quality"

Variants:
- A (Control): Simple instruction
- B (Test): Detailed instruction with examples

Sample Size: 100 test cases each
```

```
Split: Random assignment

Metrics to Compare:
1. Quality score (human evaluation, 1-5)
2. Length consistency (variance)
3. Token usage

Statistical Significance: p < 0.05
Test Duration: Until sample size reached
```

Results Template:

```
A/B Test Results:

| Metric | Prompt A | Prompt B | Difference |
|--------|----------|----------|------------|
| Quality Score | 3.8 | 4.3 | +13.2% ✅ |
| Consistency | 78% | 91% | +16.7% ✅ |
| Token Usage | 120 | 180 | +50% ⚠️ |

Conclusion: Prompt B significantly improves quality and
consistency, with acceptable token cost increase.

Recommendation: Adopt Prompt B for production.
```

🎲 Handling Model Variability

Theory: LLMs can produce different outputs for the same input. Plan for this.

Strategies for Consistency:

Strategy 1: Temperature Control

```
For factual tasks: temperature = 0
For creative tasks: temperature = 0.7-1.0
For balanced: temperature = 0.3-0.5
```

Strategy 2: Structured Output

```
Respond in exactly this format:
{
  "answer": "[your answer]",
  "confidence": "[high/medium/low]",
  "reasoning": "[brief explanation]"
}

This forces consistent structure even with varied content.
```

Strategy 3: Self-Consistency

```
Generate 3 different responses to this question.
Then, select the answer that appears most frequently
```

```
or synthesize the best elements.

Question: [Your question]

Response 1: [Generate]
Response 2: [Generate]
Response 3: [Generate]

Final Answer: [Most consistent/best synthesis]
```

📋 Building Evaluation Datasets

Theory: Create test sets to systematically evaluate prompts.

Dataset Structure:

```
{
  "test_cases": [
    {
      "id": "TC001",
      "category": "classification",
      "input": "I love this product! Best purchase ever!",
      "expected_output": "positive",
      "tags": ["simple", "clear-positive"]
    },
    {
      "id": "TC002",
      "category": "classification",
      "input": "It works, I guess. Nothing special.",
      "expected_output": "neutral",
      "tags": ["ambiguous", "edge-case"]
    }
  ]
}
```

Coverage Categories:

```
Build test cases for:

1. Happy Path (60% of cases)
   - Clear, typical inputs
   - Expected to work well

2. Edge Cases (25% of cases)
   - Ambiguous inputs
   - Very short/long inputs
   - Multiple valid interpretations

3. Adversarial (10% of cases)
   - Attempts to confuse
   - Off-topic inputs
```

```
   - Injection attempts

4. Error Conditions (5% of cases)
   - Invalid input formats
   - Empty inputs
   - Malformed data
```

Evaluation Script Pattern:

```python
def evaluate_prompt(prompt_template, test_cases):
    results = []
    for case in test_cases:
        # Generate response
        response = call_llm(prompt_template.format(input=case['input']))

        # Evaluate
        score = {
            'id': case['id'],
            'correct': response == case['expected_output'],
            'expected': case['expected_output'],
            'actual': response
        }
        results.append(score)

    # Calculate metrics
    accuracy = sum(r['correct'] for r in results) / len(results)
    return accuracy, results
```

Week 14: Iterative Improvement

🔍 Analyzing Failure Modes

Failure Analysis Framework:

```
Analyze failed responses:

FAILURE CATEGORIZATION:
1. Wrong Answer - Incorrect but plausible
2. Off Topic - Didn't address the question
3. Format Error - Right content, wrong structure
4. Hallucination - Made up information
5. Incomplete - Partial answer
6. Refusal - Declined to answer (shouldn't have)
7. Safety - Violated guidelines

For each failure:
- What was the input?
- What was expected?
- What was received?
```

```
- Why did it fail? (Root cause)
- How can prompt be fixed?
```

Example Analysis:

```
Failure Case: TC-047

Input: "What's the capital of Australia?"
Expected: "Canberra"
Actual: "Sydney is the largest city in Australia."

Category: Wrong Answer (indirect)
Root Cause: Model answered a related but different question

Prompt Fix:
Before: "Answer the question: {question}"
After: "Answer the question directly with just the answer.
Question: {question}
Direct Answer:"
```

📝 Prompt Versioning and Documentation

Version Control Template:

```
# Prompt: Customer Email Classifier

## Current Version: 2.3
Last Updated: 2024-03-15
Author: Team

## Changelog

### v2.3 (2024-03-15)
- Added handling for multi-topic emails
- Improved accuracy from 87% to 92%
- Fixed: Refund requests misclassified as complaints

### v2.2 (2024-02-28)
- Added urgency detection
- New category: "Shipping Inquiry"

### v2.1 (2024-02-10)
- Fixed edge case with empty subject lines
- Added examples for unclear emails

### v2.0 (2024-01-15)
- Complete rewrite with few-shot examples
- Accuracy improved from 78% to 87%

### v1.0 (2024-01-01)
- Initial version
```

```
- Basic zero-shot classification

## Current Prompt
[Full prompt text here]

## Performance Metrics
| Version | Accuracy | Consistency | Notes |
|---------|----------|-------------|-------|
| 2.3 | 92% | 95% | Current |
| 2.2 | 89% | 93% | |
| 2.0 | 87% | 90% | Few-shot added |
| 1.0 | 78% | 82% | Initial |
```

⚡ Performance Optimization

Token Reduction Strategies:

```
Original (150 tokens):
"You are a helpful AI assistant. Your job is to help users
with their questions. When a user asks you something, you
should think carefully about the question and provide a
helpful, accurate, and relevant answer. Please be concise
but thorough. The user's question is: {question}"

Optimized (50 tokens):
"Answer concisely and accurately.
Question: {question}
Answer:"

Savings: 66% token reduction
```

Efficiency Patterns:

Technique	Before	After	Savings
Remove filler words	"Please provide"	"Provide"	~5%
Use abbreviations	"for example"	"e.g."	~3%
Shorter instructions	100 words	30 words	~70%
Remove redundancy	Repeated phrases	Single mention	Variable

💰 Cost and Latency Considerations

Cost Calculation:

```
Cost Formula:
Total Cost = (Input Tokens + Output Tokens) × Price per Token

Example (GPT-4):
- Input: 500 tokens × $0.03/1K = $0.015
```

```
- Output: 200 tokens × $0.06/1K = $0.012
- Per Request: $0.027

At 10,000 requests/day:
- Daily Cost: $270
- Monthly Cost: $8,100
```

Optimization Decisions:

```
Cost/Quality Trade-off Analysis:

| Scenario | Model | Cost | Quality | Decision |
|----------|-------|------|---------|----------|
| Simple FAQ | GPT-3.5 | $0.002 | 90% | ✅ Use |
| Complex reasoning | GPT-4 | $0.027 | 98% | ✅ Use |
| Classification | GPT-3.5 | $0.001 | 88% | ✅ Use |
| Code review | GPT-4 | $0.030 | 95% | ✅ Use |

Strategy:
- Use cheaper models for simple tasks
- Reserve expensive models for complex tasks
- Cache common responses
- Batch similar requests
```

💡 Key Takeaways

1. **Define metrics first** - Know what "good" means
2. **A/B test** - Compare versions systematically
3. **Plan for variability** - LLMs aren't deterministic
4. **Document everything** - Version your prompts
5. **Optimize last** - Only after quality is proven

Next: Module 6 - Production and Integration →