ML 25/26-08: LLM Prompt Testing Framework - Complete Guide

🎯 THE CORE PROBLEM

When you use ChatGPT or any LLM, you give it a **prompt** (instruction), and it generates an **output**. But here's the problem:

The same prompt can give different answers each time!

Example:

- **Prompt:** "What is the capital of France?"
- **Response 1:** "The capital of France is Paris."
- **Response 2:** "Paris is the capital city of France."
- **Response 3:** "France's capital is Paris, located in the north-central part of the country."

All three are **correct** but **worded differently**. Traditional testing (like `assert output == "Paris"`) would **fail** for responses 1-3, even though they're all right!

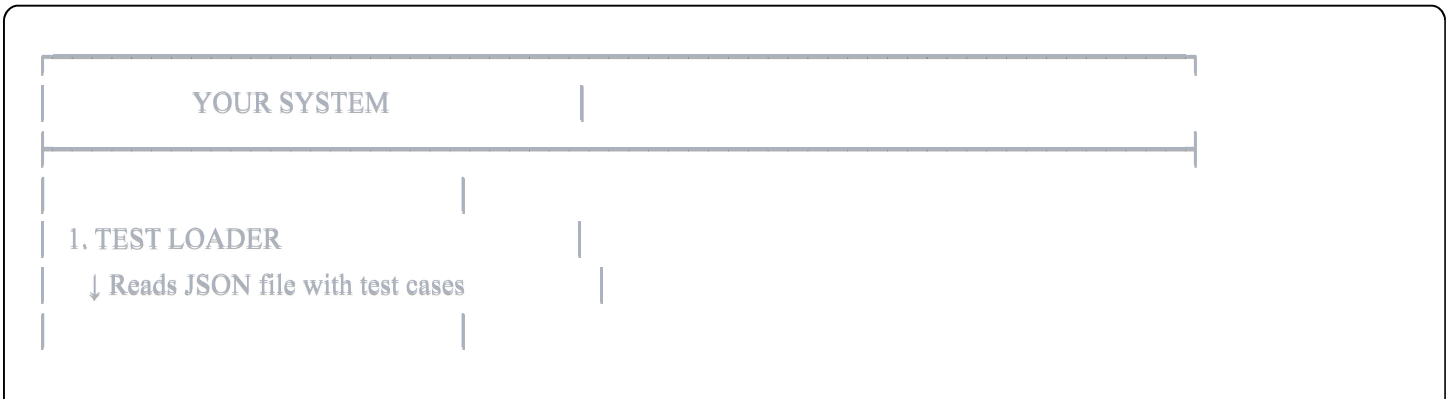
🎯 PROJECT GOAL

Build a testing system that can automatically verify if LLM outputs are correct, even when they're worded differently.

Think of it like a teacher grading exam answers - they don't need the exact same words, they check if the **meaning** is correct.

🔍 WHAT YOU'RE ACTUALLY BUILDING

A Three-Part System:



```
2. LLM EXECUTOR
   ↓ Sends prompts to OpenAI/other LLM
   ↓ Gets responses

3. SEMANTIC VALIDATOR (THE SMART PART)
   ↓ Checks if response is correct
   ↓ Uses AI to understand meaning
   ↓ Reports PASS/FAIL
```

📋 CONCRETE EXAMPLE: Step-by-Step

Test Case in JSON:

```json
{
  "test_id": "capital_france_01",
  "input_prompt": "What is the capital of France?",
  "expected_output": "Paris",
  "evaluation_criteria": "The response must correctly identify Paris as the capital"
}
```

Your System Runs:

Step 1: Load Test

```
✓ Loaded test: capital_france_01
```

Step 2: Execute LLM

```
→ Sent to OpenAI: "What is the capital of France?"
← Got response: "France's capital city is Paris, known for the Eiffel Tower."
```

Step 3: Validate (Two Methods)

Method A: Embedding Similarity

1. Convert expected output to vector: "Paris" → [0.2, 0.8, 0.1, ...]
2. Convert actual output to vector: "France's capital city is Paris..." → [0.19, 0.81, 0.09, ...]
3. Calculate cosine similarity: 0.94 (very similar!)
4. Check threshold: 0.94 ≥ 0.85 ✓ PASS

Method B: LLM-as-Judge

→ Send to another LLM:
"Does this response correctly answer the question?
Question: What is the capital of France?
Expected: Paris
Actual: France's capital city is Paris, known for the Eiffel Tower.
Rate 1-10."

← LLM Judge responds: "9/10 - Correct answer with extra context"
→ 9 ≥ 8 threshold ✓ PASS

Step 4: Report

Test: capital_france_01
Status: ✓ PASS
Embedding Score: 0.94
Judge Score: 9/10

🛠️ IMPLEMENTATION APPROACH

Phase 1: Basic Setup (Week 1-2)

What to do:

1. Create C# console application
2. Set up OpenAI API connection
3. Create simple prompt sender

Code Example:

csharp

```
public class LLMClient
{
    public async Task<string> SendPrompt(string prompt)
    {
        // Send to OpenAI API
        // Get response
        // Return response text
    }
}
```

Deliverable: Console app that can send a prompt and print the response.

Phase 2: Test Case Loading (Week 2-3)

What to do:

1. Create JSON structure for test cases
2. Build test case loader
3. Create 20-30 simple test cases

JSON Structure:

```
json
{
  "tests": [
    {
      "id": "test_001",
      "prompt": "What is 2+2?",
      "expected": "4"
    },
    {
      "id": "test_002",
      "prompt": "Name the first president of the USA",
      "expected": "George Washington"
    }
  ]
}
```

Code Example:

```csharp
public class TestCase
{
    public string Id { get; set; }
    public string Prompt { get; set; }
    public string Expected { get; set; }
}

public class TestLoader
{
    public List<TestCase> LoadTests(string jsonPath)
    {
        // Read JSON
        // Deserialize
        // Return list
    }
}
```

Deliverable: System loads tests from JSON and runs them through LLM.

Phase 3: Embedding-Based Validation (Week 3-5)

What to do:

1. Use OpenAI's embedding API
2. Convert text to vectors
3. Calculate cosine similarity
4. Set threshold for pass/fail

The Math (Don't Panic!):

Cosine Similarity Formula:

```
similarity = (A · B) / (||A|| × ||B||)
```

In simple terms: How similar are two vectors? Result: 0 to 1

- 1.0 = identical
- 0.9+ = very similar

- 0.8+ = similar enough
- < 0.7 = different

Code Example:

```csharp
public class EmbeddingValidator
{
    public async Task<double> GetSimilarity(string text1, string text2)
    {
        // 1. Get embedding for text1
        var embedding1 = await GetEmbedding(text1);

        // 2. Get embedding for text2
        var embedding2 = await GetEmbedding(text2);

        // 3. Calculate cosine similarity
        return CosineSimilarity(embedding1, embedding2);
    }

    private double CosineSimilarity(float[] vec1, float[] vec2)
    {
        double dot = 0, mag1 = 0, mag2 = 0;

        for (int i = 0; i < vec1.Length; i++)
        {
            dot += vec1[i] * vec2[i];
            mag1 += vec1[i] * vec1[i];
            mag2 += vec2[i] * vec2[i];
        }

        return dot / (Math.Sqrt(mag1) * Math.Sqrt(mag2));
    }

    public bool IsPass(double similarity, double threshold = 0.85)
    {
        return similarity >= threshold;
    }
}
```

Deliverable: Tests pass/fail based on semantic similarity.

Phase 4: LLM-as-Judge (Week 5-7)

What to do:

1. Create evaluation prompts
2. Send to second LLM
3. Parse judge's response
4. Combine with embedding scores

Judge Prompt Template:

```csharp
string judgePrompt = $@"
You are evaluating an AI response.

Question: {testCase.Prompt}
Expected Answer: {testCase.Expected}
Actual Answer: {actualResponse}

Criteria: {testCase.Criteria}

Think step by step:
1. Does the actual answer contain the correct information?
2. Is it factually accurate?
3. Does it match the expected meaning?

Then rate from 1-10. Output ONLY the number.
";
```

Code Example:

```csharp
```

```
public class LLMJudge
{
    public async Task<int> EvaluateResponse(
        string prompt,
        string expected,
        string actual)
    {
        var judgePrompt = CreateJudgePrompt(prompt, expected, actual);
        var judgeResponse = await llmClient.SendPrompt(judgePrompt);

        // Parse score from response
        var score = ExtractScore(judgeResponse);
        return score;
    }

    public bool IsPass(int score, int threshold = 8)
    {
        return score >= threshold;
    }
}
```

Deliverable: Dual validation system (embedding + judge).

Phase 5: Reporting & Analysis (Week 7-8)

What to do:

1. Run all 100+ tests
2. Generate summary report
3. Create visualizations (optional)
4. Handle multiple runs for stability

Report Structure:

```
==========================================
TEST EXECUTION REPORT
==========================================
Total Tests: 150
Passed: 142 (94.7%)
Failed: 8 (5.3%)

By Category:
- Factual Questions: 50/50 (100%)
- Math Problems: 45/48 (93.8%)
- Creative Tasks: 47/52 (90.4%)

Average Embedding Score: 0.89
Average Judge Score: 8.7/10

Failed Tests:
1. test_073 - Math problem (judge: 6/10)
2. test_089 - Ambiguous wording (similarity: 0.72)
...
```

Code Example:

```csharp
```

```
public class TestRunner
{
    public async Task<TestReport> RunAllTests(List<TestCase> tests)
    {
        var results = new List<TestResult>();

        foreach (var test in tests)
        {
            // Run test 3 times for stability
            var runs = new List<TestRun>();
            for (int i = 0; i < 3; i++)
            {
                var response = await llmClient.SendPrompt(test.Prompt);
                var embeddingScore = await embeddingValidator.GetSimilarity(
                    test.Expected, response);
                var judgeScore = await llmJudge.EvaluateResponse(
                    test.Prompt, test.Expected, response);

                runs.Add(new TestRun {
                    Response = response,
                    EmbeddingScore = embeddingScore,
                    JudgeScore = judgeScore
                });
            }

            // Aggregate results (e.g., pass if 2/3 pass)
            var passed = runs.Count(r => r.EmbeddingScore >= 0.85) >= 2;

            results.Add(new TestResult {
                TestId = test.Id,
                Passed = passed,
                Runs = runs
            });
        }

        return new TestReport(results);
    }
}
```

💡 REAL-WORLD ANALOGY

Imagine you're building an **automated teacher**:

1. **Test Cases** = Exam questions with answer keys
2. **LLM** = Student taking the exam
3. **Embedding Similarity** = Checking if the answer means the same thing
4. **LLM Judge** = A second teacher reviewing the answer
5. **Report** = Final grade sheet

🎓 WHAT YOU'LL LEARN

Technical Skills:

- ✅ Working with LLM APIs (OpenAI)
- ✅ Understanding embeddings & vector math
- ✅ Building automated testing systems
- ✅ Handling non-deterministic AI outputs
- ✅ C# async/await patterns
- ✅ JSON data handling

Conceptual Understanding:

- ✅ Why exact string matching fails for AI
- ✅ How semantic similarity works
- ✅ Prompt engineering best practices
- ✅ AI evaluation methodologies
- ✅ The challenge of AI reliability

Career-Relevant:

- ✅ LLM testing is a HOT job market skill
- ✅ Every AI company needs this
- ✅ Great portfolio project

- ✅ Foundation for MLOps roles

🚀 SUCCESS CRITERIA

Your project is complete when:

✅ System loads 100+ test cases from JSON
✅ Sends prompts to LLM and receives responses
✅ Validates responses using embedding similarity
✅ Validates responses using LLM-as-judge
✅ Runs each test 3-5 times for stability
✅ Generates comprehensive pass/fail report
✅ Identifies which tests consistently fail
✅ Produces accuracy metrics (% passed)

📊 EXAMPLE TEST CATEGORIES

Create diverse test cases:

1. Factual Questions

```json
{ "prompt": "What is the capital of Germany?", "expected": "Berlin" }
{ "prompt": "Who wrote Romeo and Juliet?", "expected": "William Shakespeare" }
```

2. Math Problems

```json
{ "prompt": "Calculate 15% of 200", "expected": "30" }
{ "prompt": "What is the square root of 144?", "expected": "12" }
```

3. Definition Questions

```json
{ "prompt": "Define photosynthesis", "expected": "Process where plants convert light into energy" }
```

4. Reasoning Tasks

```json
{ "prompt": "If all dogs are animals, and Max is a dog, what is Max?", "expected": "An animal" }
```

🎯 FINAL DELIVERABLE

A complete system that:

1. **Loads** test cases from JSON
2. **Executes** prompts against LLM
3. **Validates** semantic correctness (not exact matching)
4. **Reports** pass/fail with scores
5. **Analyzes** which test categories perform best
6. **Documents** methodology and findings

This proves you understand how to **evaluate AI systems reliably -** a critical skill companies are desperate for!