1. What are Evals (Evaluations)?

Explanation

Evaluations (Evals) are systematic methods to assess the performance, quality, and reliability of Large Language Models (LLMs) and AI systems. They provide quantitative and qualitative measures of how well an AI system performs on specific tasks.

Example: Basic Eval Setup

```
# Example evaluation framework
def evaluate_model(model, test_dataset, metrics):
    results = {}
    for metric in metrics:
        scores = []
        for test_case in test_dataset:
            prediction = model.predict(test_case['input'])
            score = metric.evaluate(prediction, test_case['expected'])
            scores.append(score)
        results[metric.name] = sum(scores) / len(scores)
    return results
```

Sample Image

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Input Data    │───▶│      Model      │───▶│   Evaluation    │
│                 │    │                 │    │   Metrics       │
│ • Test Cases    │    │ • LLM/AI System │    │ • Accuracy      │
│ • Ground Truth  │    │                 │    │ • Precision     │
│ • Edge Cases    │    │                 │    │ • Recall        │
└─────────────────┘    └─────────────────┘    └─────────────────┘
                                ▲
                                │
                       ┌─────────────────┐
                       │   Results &     │
                       │   Insights      │
                       │ • Performance   │
                       │ • Improvements  │
                       │ • Optimization  │
                       └─────────────────┘
```

---

2. Prompt Testing in GSheets (No-Code)

Explanation

Google Sheets provides a user-friendly, no-code environment for testing and iterating on prompts. This approach democratizes prompt engineering by allowing non-technical users to participate in optimization.

Example: GSheets Template Structure

Prompt Version	**Input Text**	**AI Response**	**Quality Score**	**Notes**
Version 1	"Summarize this article..."	[Response]	8/10	Clear but brief
Version 2	"Provide a detailed summary..."	[Response]	9/10	More comprehensive

Sample Image

```
┌─────────────────────────────────────────────────────────────┐
│                  PROMPT TESTING DASHBOARD                    │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐          │
│  │Avg Quality  │  │ Best Prompt│  │ Improvement │          │
│  │   Score     │  │   Version  │  │   Trend     │          │
│  │   8.5/10    │  │   V2.3     │  │   ↗ 15%     │          │
│  └─────────────┘  └─────────────┘  └─────────────┘          │
├─────────────────────────────────────────────────────────────┤
│  Prompt Versions     │ Inputs  │ Outputs │ Scores │ Analysis │
├──────────────────────┼─────────┼─────────┼────────┼──────────┤
│ Version A            │ Text    │ Resp    │ 7/10   │ Too vague  │
│ Version B            │ Text    │ Resp    │ 9/10   │ Clear      │
│ Version C            │ Text    │ Resp    │ 6/10   │ Off-topic  │
└─────────────────────────────────────────────────────────────┘
```

3. Eval for a RAG System

Explanation

RAG (Retrieval-Augmented Generation) evaluation focuses on assessing the quality of retrieved information and generated responses in systems that combine document retrieval with text generation.

Example: RAG Evaluation Code

```python
def evaluate_rag_system(rag_system, test_queries):
    results = {
        'retrieval_precision': [],
        'retrieval_recall': [],
        'generation_faithfulness': [],
        'answer_relevance': []
    }

    for query in test_queries:
        # Retrieve documents
        retrieved_docs = rag_system.retrieve(query)

        # Generate response
        response = rag_system.generate(query, retrieved_docs)

        # Evaluate retrieval
        retrieval_metrics = evaluate_retrieval(retrieved_docs, query)

        # Evaluate generation
        generation_metrics = evaluate_generation(response, retrieved_docs, query)

        # Store results
        for metric in results:
            if metric in retrieval_metrics:
                results[metric].append(retrieval_metrics[metric])
            if metric in generation_metrics:
                results[metric].append(generation_metrics[metric])

    return results
```

Sample Image

```
┌─────────────────┐   ┌─────────────────┐   ┌─────────────────┐
│   User Query    │──▶│   Retrieval     │──▶│   Generation    │
│                 │   │   System        │   │   System        │
└─────────────────┘   └─────────────────┘   └─────────────────┘
         │                     │                     │
         ▼                     ▼                     ▼
┌─────────────────┐   ┌─────────────────┐   ┌─────────────────┐
│Query Processing │   │Document Ranking │   │Response Quality │
│• Intent Analysis│   │• Relevance Score│   │• Accuracy       │
│• Entity Extract │   │• Diversity      │   │• Completeness   │
└─────────────────┘   └─────────────────┘   └─────────────────┘
                               ▲
                               │
                      ┌─────────────────┐
                      │   RAG Metrics   │
                      │ • Retrieval F1  │
                      │ • Faithfulness  │
                      │ • Answer Rel.   │
                      └─────────────────┘
```

4. Prompt Optimization with DSPy - Coding

Explanation

DSPy (Declarative Self-improving Python) is a framework for automatically optimizing prompts and LLM programs through systematic experimentation and optimization.

Example: DSPy Basic Optimization

```python
import dspy

# Define a simple question-answering program
class QAProgram(dspy.Module):
    def __init__(self):
        super().__init__()
        self.generate_answer = dspy.ChainOfThought("question -> answer")

    def forward(self, question):
        return self.generate_answer(question=question)

# Setup evaluation metric
def validate_answer(example, prediction, trace=None):
    # Simple exact match validation
    return prediction.answer.lower() == example.answer.lower()

# Optimize the program
program = QAProgram()
optimizer = dspy.BootstrapFewShot(metric=validate_answer)

# Compile with training data
compiled_program = optimizer.compile(program, trainset=train_data)
```

Sample Image

```
┌─────────────────┐   ┌─────────────────┐   ┌─────────────────┐
│   Base Program  │──▶│    Optimizer    │──▶│Optimized Program│
│                 │   │                 │   │                 │
│ • Basic Prompt  │   │ • DSPy Algo     │   │ • Improved      │
│ • Simple Logic  │   │ • Few-shot      │   │ • Best Practices│
└─────────────────┘   └─────────────────┘   └─────────────────┘
                               ▲
                               │
                      ┌─────────────────┐
                      │  Training Data  │
                      │ • Examples      │
                      │ • Metrics       │
                      │ • Validation    │
                      └─────────────────┘
```

5. Eval Metrics with DSPy - Coding

Explanation

DSPy provides built-in evaluation metrics and tools for comprehensive assessment of LLM performance, including both automated and human-in-the-loop evaluation methods.

Example: DSPy Evaluation Metrics

```python
import dspy
from dspy.evaluate import Evaluate

# Define custom metrics
def exact_match_metric(gold, pred, trace=None):
    """Exact match between prediction and gold standard"""
    return gold.answer.lower() == pred.answer.lower()

def semantic_similarity_metric(gold, pred, trace=None):
    """Semantic similarity using embeddings"""
    gold_embedding = get_embedding(gold.answer)
    pred_embedding = get_embedding(pred.answer)
    return cosine_similarity(gold_embedding, pred_embedding) > 0.8

def comprehensive_metric(gold, pred, trace=None):
    """Combined metric with multiple criteria"""
    exact_score = exact_match_metric(gold, pred)
    semantic_score = semantic_similarity_metric(gold, pred)

    # Weighted combination
    return 0.7 * exact_score + 0.3 * semantic_score

# Setup evaluator
evaluator = Evaluate(
    devset=test_data,
    metric=comprehensive_metric,
    num_threads=4,
    display_progress=True
)

# Run evaluation
evaluation_results = evaluator(program)
print(f"Average Score: {evaluation_results}")
```

Sample Image

```
┌──────────────────────────────────────────────────────────────┐
│                    DSPy Evaluation Dashboard                 │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐          │
│  │Exact Match  │  │Semantic Sim│  │Comprehensive│          │
│  │   0.75      │  │   0.82      │  │    0.78     │          │
│  └─────────────┘  └─────────────┘  └─────────────┘          │
│                                                              │
├──────────────────────────────────────────────────────────────┤
│  Metric Breakdown        │ Score │ Weight │ Contribution     │
├──────────────────────────┼───────┼────────┼─────────────────┤
│ Exact Match             │ 0.75  │ 0.7    │ 0.525           │
│ Semantic Similarity     │ 0.82  │ 0.3    │ 0.246           │
│ BLEU Score             │ 0.68  │ -      │ -               │
│ ROUGE Score            │ 0.71  │ -      │ -               │
└──────────────────────────────────────────────────────────────┘
```

6. Prompt Optimization: 5 Principles of Prompting

Explanation

The 5 Principles of Prompting provide a structured framework for creating effective prompts that maximize LLM performance and reliability.

Example: Applying 5 Principles

Poor Prompt: "Summarize this article."

Optimized Prompt using 5 Principles:

```
Please provide a comprehensive summary of the following article. Follow these guidelines:

CONTEXT: This article discusses climate change impacts on coastal cities, published in a scientific journal.

TASK: Create a summary that includes:
1. Main thesis and key findings
2. Supporting evidence and data
3. Implications for policy and urban planning
4. Limitations of the study

FORMAT: Structure your response with clear headings and bullet points. Keep the summary between 300-400 words.

EXAMPLE: For a similar environmental study, a good summary would highlight the methodology, results, and practical applications.

Please ensure your summary is accurate, objective, and well-structured.
```

Sample Image

```
┌─────────────────────────────────────────────────────────────┐
│             5 Principles of Prompting                       │
├─────────────────────────────────────────────────────────────┤
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐          │
│  │  1. Clarity │  │ 2.Specific │  │  3.Context  │          │
│  │ Clear instr │  │ Precise req │  │ Background │          │
│  └─────────────┘  └─────────────┘  └─────────────┘          │
├─────────────────────────────────────────────────────────────┤
│  ┌─────────────┐  ┌─────────────┐                           │
│  │ 4.Structure│  │ 5.Examples │                           │
│  │ Organized  │  │ Few-shot   │                           │
│  └─────────────┘  └─────────────┘                           │
├─────────────────────────────────────────────────────────────┤
│  Before ──▶ After Optimization                              │
│  "Summarize" ──▶ "Detailed summary with structure..."       │
└─────────────────────────────────────────────────────────────┘
```

Sample Code: Prompt Optimization Function

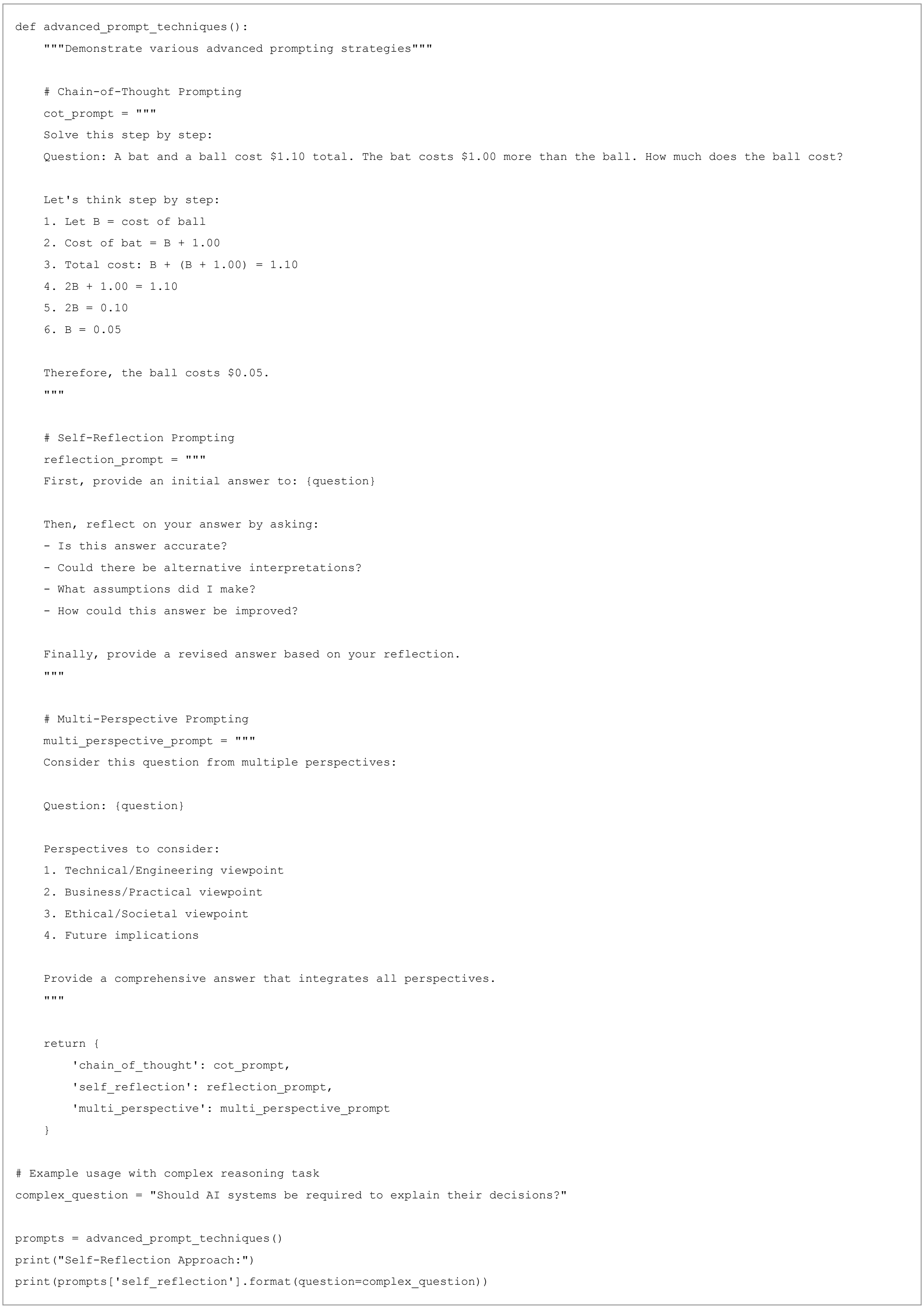
```python
def optimize_prompt_with_principles(task_description, context=None):
    """
    Apply 5 principles of prompting to optimize a basic task description
    """
    optimized_prompt = f"""
Please perform the following task with high quality and precision:

CONTEXT: {context or "General purpose task"}

TASK: {task_description}

REQUIREMENTS:
• Be specific and detailed in your response
• Provide step-by-step reasoning if applicable
• Include relevant examples or evidence
• Structure your answer clearly with sections if needed
• Ensure accuracy and completeness

FORMAT: Use clear headings, bullet points, and maintain professional tone.

INSTRUCTION: Take your time to provide a thoughtful, well-reasoned response.
"""
    return optimized_prompt

# Example usage
basic_task = "Explain machine learning"
context = "Explaining to high school students"
optimized = optimize_prompt_with_principles(basic_task, context)
print(optimized)
```

7. Prompt Optimization: Advanced

Explanation

Advanced prompt optimization techniques go beyond basic principles to include complex strategies like chain-of-thought, self-reflection, multi-step reasoning, and adaptive prompting based on model behavior.

Example: Advanced Prompting Techniques



```python
def advanced_prompt_techniques():
    """Demonstrate various advanced prompting strategies"""

    # Chain-of-Thought Prompting
    cot_prompt = """
    Solve this step by step:
    Question: A bat and a ball cost $1.10 total. The bat costs $1.00 more than the ball. How much does the ball cost?

    Let's think step by step:
    1. Let B = cost of ball
    2. Cost of bat = B + 1.00
    3. Total cost: B + (B + 1.00) = 1.10
    4. 2B + 1.00 = 1.10
    5. 2B = 0.10
    6. B = 0.05

    Therefore, the ball costs $0.05.
    """

    # Self-Reflection Prompting
    reflection_prompt = """
    First, provide an initial answer to: {question}

    Then, reflect on your answer by asking:
    - Is this answer accurate?
    - Could there be alternative interpretations?
    - What assumptions did I make?
    - How could this answer be improved?

    Finally, provide a revised answer based on your reflection.
    """

    # Multi-Perspective Prompting
    multi_perspective_prompt = """
    Consider this question from multiple perspectives:

    Question: {question}

    Perspectives to consider:
    1. Technical/Engineering viewpoint
    2. Business/Practical viewpoint
    3. Ethical/Societal viewpoint
    4. Future implications

    Provide a comprehensive answer that integrates all perspectives.
    """

    return {
        'chain_of_thought': cot_prompt,
        'self_reflection': reflection_prompt,
        'multi_perspective': multi_perspective_prompt
    }

# Example usage with complex reasoning task
complex_question = "Should AI systems be required to explain their decisions?"

prompts = advanced_prompt_techniques()
print("Self-Reflection Approach:")
print(prompts['self_reflection'].format(question=complex_question))
```

Sample Image

```
┌──────────────────────────────────────────────────────────────┐
│              Advanced Prompting Techniques                   │
├──────────────────────────────────────────────────────────────┤
│  ┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐│
│  │Chain-of-Thought │  │Self-Reflection  │  │Multi-Perspective│  │
│  │Step-by-step     │  │Critique own     │  │Multiple views   │  │
│  │reasoning        │  │responses        │  │integration      │  │
│  └─────────────────┘  └─────────────────┘  └─────────────────┘│
├──────────────────────────────────────────────────────────────┤
│  ┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐│
│  │Progressive      │  │Adaptive         │  │Contextual       │  │
│  │Refinement       │  │Strategies       │  │Optimization     │  │
│  │Iterative improve│  │Model-specific   │  │Domain expertise │   │
│  └─────────────────┘  └─────────────────┘  └─────────────────┘│
├──────────────────────────────────────────────────────────────┤
│  Performance Comparison                                      │
│  ┌─────────────┬─────────────┬─────────────┬─────────────┐    │
│  │ Technique   │ Basic Promp │ Adv. Tech  │ Improvement │    │
│  ├─────────────┼─────────────┼─────────────┼─────────────┤    │
│  │ CoT         │ 65%         │ 82%         │ +17%        │    │
│  │ Reflection  │ 70%         │ 85%         │ +15%        │    │
│  │ Multi-view  │ 68%         │ 88%         │ +20%        │    │
│  └─────────────┴─────────────┴─────────────┴─────────────┘    │
└──────────────────────────────────────────────────────────────┘
```

8. Sammo - Introduction

Explanation

Sammo is an advanced framework for prompt optimization that focuses on systematic prompt engineering through data-driven approaches and automated optimization strategies.

Example: Sammo Basic Setup

```python
import sammo

# Initialize Sammo session
session = sammo.Session()

# Define optimization objective
objective = sammo.Objective(
    name="response_quality",
    metric=sammo.metrics.ExactMatch(),
    target=0.85  # Target performance level
)

# Create prompt template with variables
prompt_template = sammo.PromptTemplate("""
{instruction_style}
{context}
Please answer: {question}
{output_format}
""")

# Define search space
search_space = {
    "instruction_style": [
        "Answer the following question:",
        "Please provide a detailed answer to:",
        "Solve this problem step by step:"
    ],
    "context": [
        "",
        "Given your expertise in this domain:",
        "As a knowledgeable assistant:"
    ],
    "output_format": [
        "",
        "Format your answer clearly:",
        "Structure your response with bullet points:"
    ]
}

# Setup optimizer
optimizer = sammo.Optimizer(
    template=prompt_template,
    search_space=search_space,
    objective=objective,
    strategy="grid_search"  # or "random_search", "bayesian"
)

# Run optimization
optimized_prompt = optimizer.optimize(training_data)
```

Sample Image

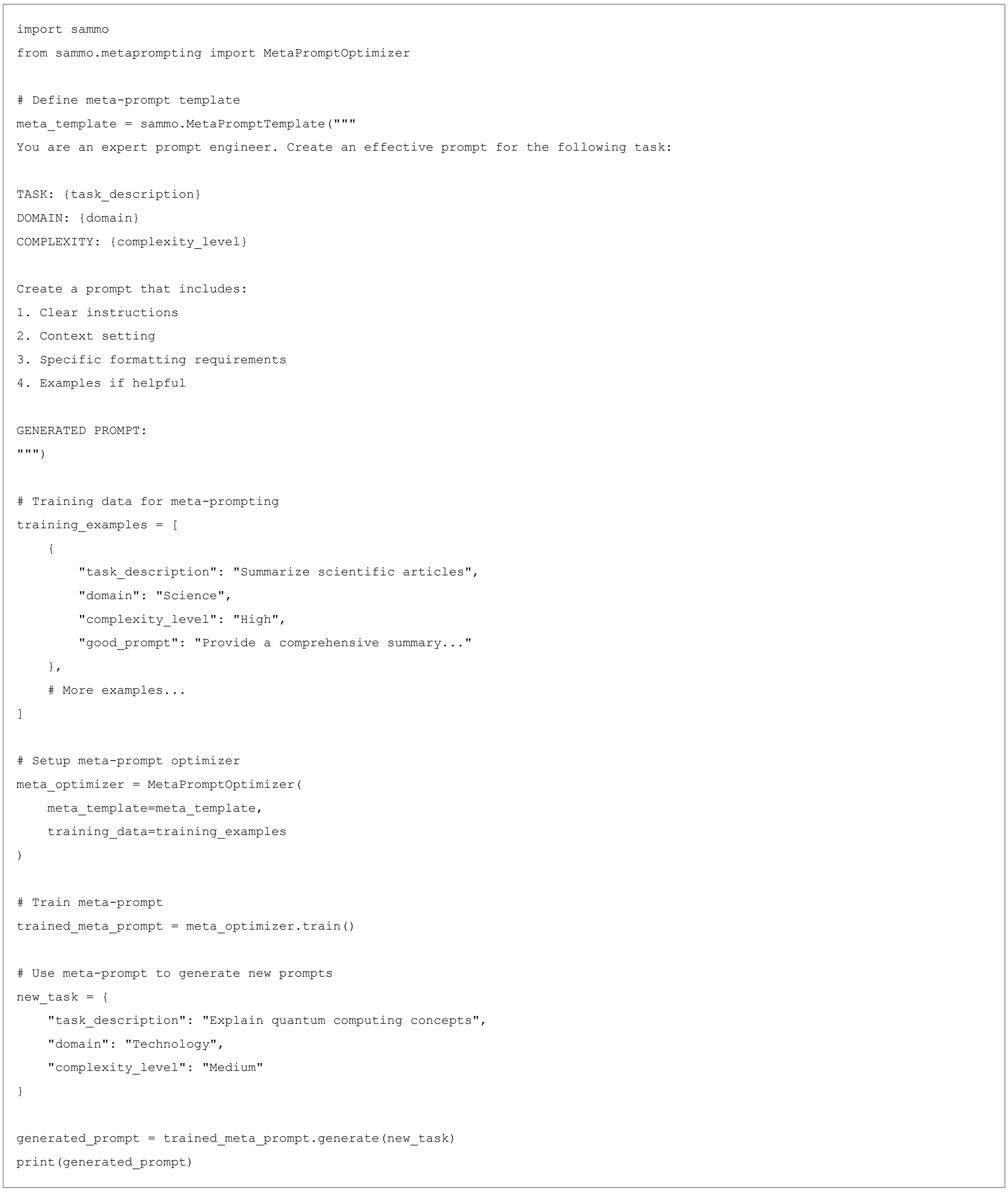
```
┌─────────────────┐   ┌─────────────────┐   ┌─────────────────┐
│   Define Goal   │──►│   Create Search │──►│   Optimize       │
│                 │   │     Space       │   │                 │
│ • Quality Metric│   │ • Prompt Vars   │   │ • Run Algorithm │
│ • Target Score  │   │ • Variations    │   │ • Find Best     │
└─────────────────┘   └─────────────────┘   └─────────────────┘
                               ▲
                               │
                      ┌─────────────────┐
                      │   Best Prompt   │
                      │ • Template      │
                      │ • Performance   │
                      │ • Metrics       │
                      └─────────────────┘
```

9. Sammo - Metaprompting

Explanation

Metaprompting in Sammo involves creating prompts that can generate or optimize other prompts automatically. It's a higher-level approach where the system learns to create effective prompts for various tasks.

Example: Sammo Metaprompting



```python
import sammo
from sammo.metaprompting import MetaPromptOptimizer

# Define meta-prompt template
meta_template = sammo.MetaPromptTemplate("""
You are an expert prompt engineer. Create an effective prompt for the following task:

TASK: {task_description}
DOMAIN: {domain}
COMPLEXITY: {complexity_level}

Create a prompt that includes:
1. Clear instructions
2. Context setting
3. Specific formatting requirements
4. Examples if helpful

GENERATED PROMPT:
""")

# Training data for meta-prompting
training_examples = [
    {
        "task_description": "Summarize scientific articles",
        "domain": "Science",
        "complexity_level": "High",
        "good_prompt": "Provide a comprehensive summary..."
    },
    # More examples...
]

# Setup meta-prompt optimizer
meta_optimizer = MetaPromptOptimizer(
    meta_template=meta_template,
    training_data=training_examples
)

# Train meta-prompt
trained_meta_prompt = meta_optimizer.train()

# Use meta-prompt to generate new prompts
new_task = {
    "task_description": "Explain quantum computing concepts",
    "domain": "Technology",
    "complexity_level": "Medium"
}

generated_prompt = trained_meta_prompt.generate(new_task)
print(generated_prompt)
```

Sample Image

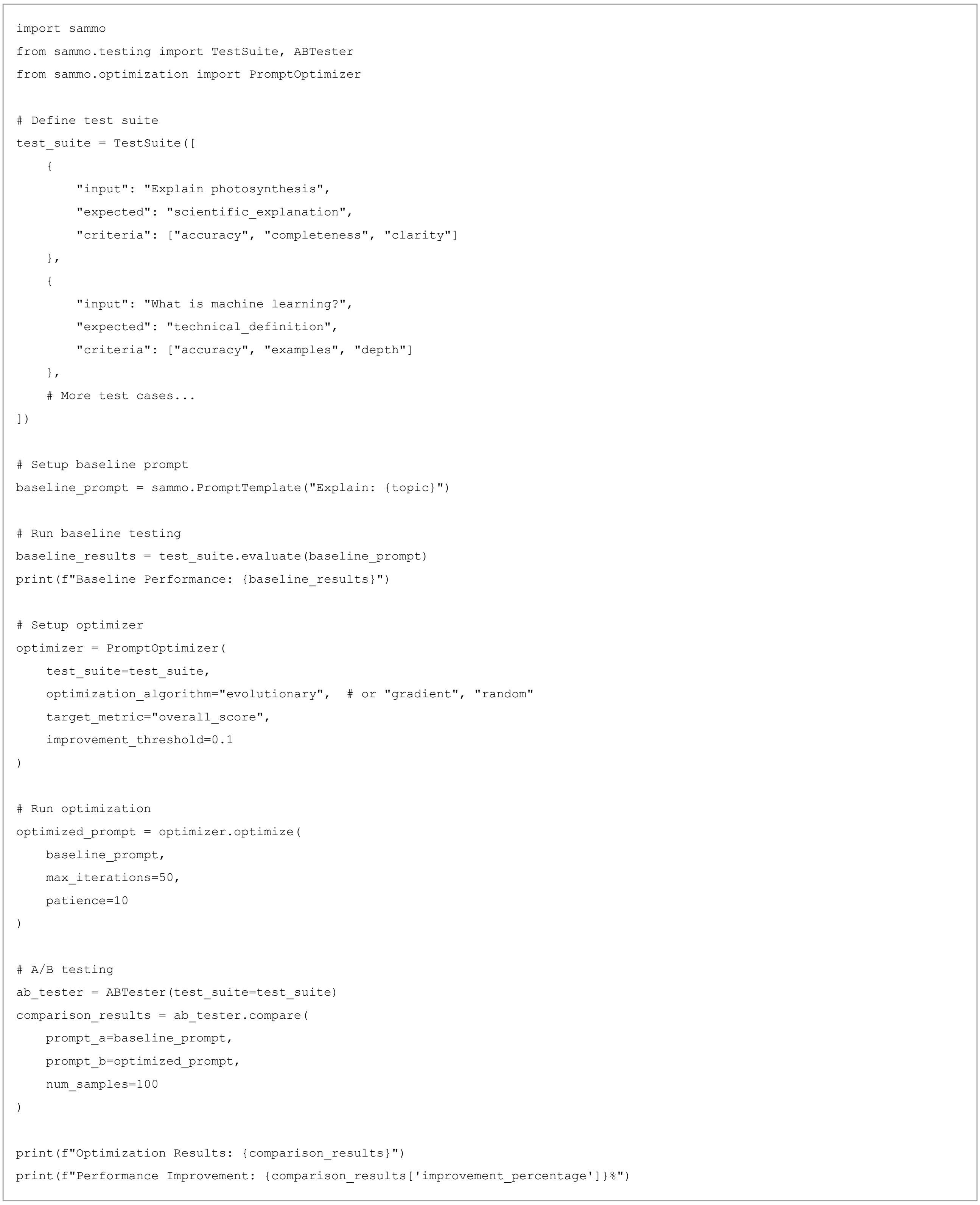
```
┌─────────────────┐   ┌─────────────────┐   ┌─────────────────┐
│   Meta-Template │──►│   Train Meta    │──►│Generate Prompts │
│                 │   │   Prompt        │   │                 │
│ • Template for  │   │ • Examples      │   │ • Task-specific │
│   Prompts       │   │ • Patterns      │   │ • Optimized     │
└─────────────────┘   └─────────────────┘   └─────────────────┘
                              ▲
                              │
                      ┌─────────────────┐
                      │   Quality       │
                      │   Assessment    │
                      │ • Performance   │
                      │ • Refinement    │
                      │ • Iteration     │
                      └─────────────────┘
```

10. Sammo - Testing and Optimization

Explanation

Sammo's testing and optimization capabilities provide comprehensive tools for evaluating prompt performance and systematically improving them through data-driven approaches.

Example: Sammo Testing and Optimization



```python
import sammo
from sammo.testing import TestSuite, ABTester
from sammo.optimization import PromptOptimizer

# Define test suite
test_suite = TestSuite([
    {
        "input": "Explain photosynthesis",
        "expected": "scientific_explanation",
        "criteria": ["accuracy", "completeness", "clarity"]
    },
    {
        "input": "What is machine learning?",
        "expected": "technical_definition",
        "criteria": ["accuracy", "examples", "depth"]
    },
    # More test cases...
])

# Setup baseline prompt
baseline_prompt = sammo.PromptTemplate("Explain: {topic}")

# Run baseline testing
baseline_results = test_suite.evaluate(baseline_prompt)
print(f"Baseline Performance: {baseline_results}")

# Setup optimizer
optimizer = PromptOptimizer(
    test_suite=test_suite,
    optimization_algorithm="evolutionary",  # or "gradient", "random"
    target_metric="overall_score",
    improvement_threshold=0.1
)

# Run optimization
optimized_prompt = optimizer.optimize(
    baseline_prompt,
    max_iterations=50,
    patience=10
)

# A/B testing
ab_tester = ABTester(test_suite=test_suite)
comparison_results = ab_tester.compare(
    prompt_a=baseline_prompt,
    prompt_b=optimized_prompt,
    num_samples=100
)

print(f"Optimization Results: {comparison_results}")
print(f"Performance Improvement: {comparison_results['improvement_percentage']}%")
```

Sample Image

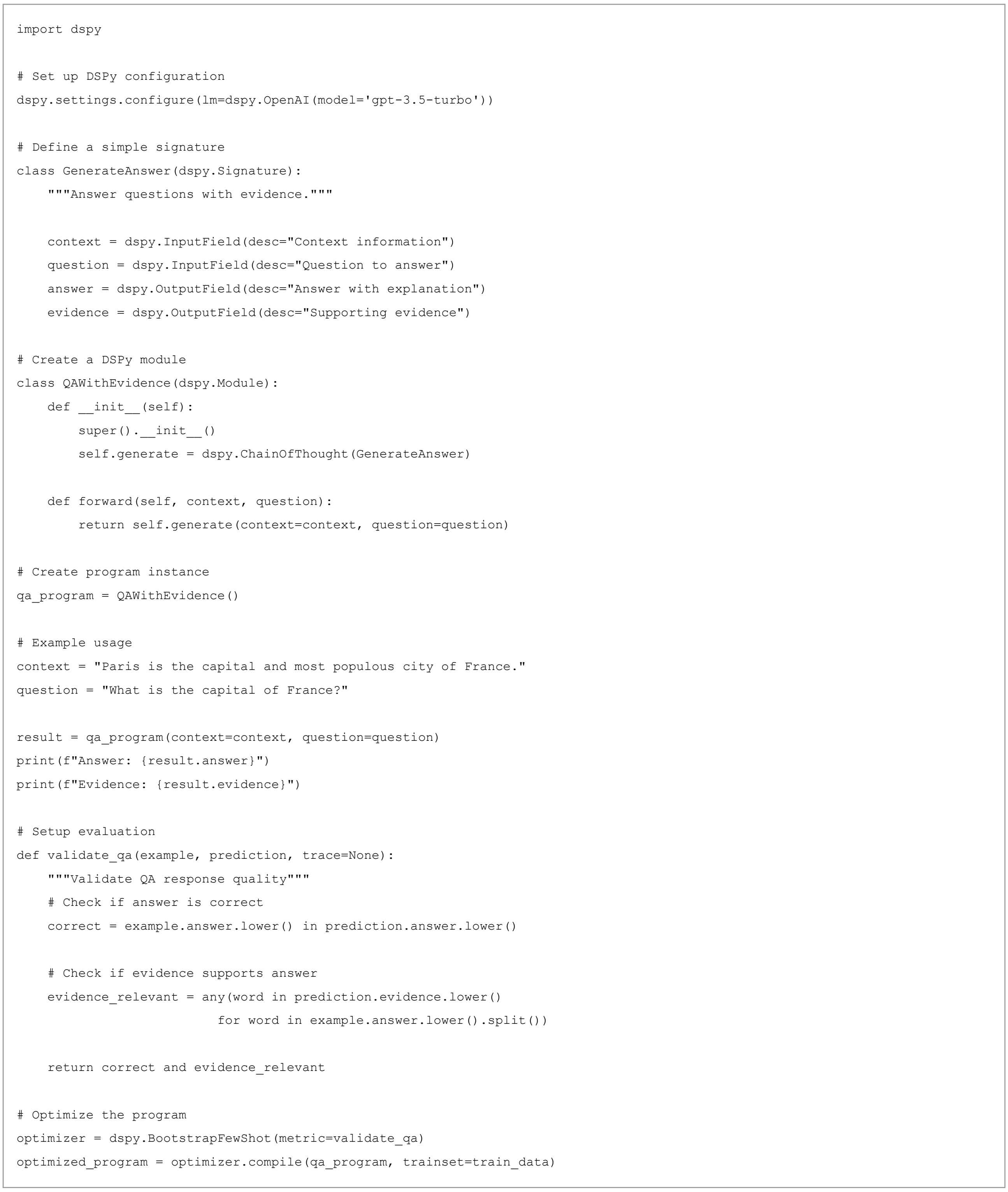
```
┌─────────────────┐   ┌─────────────────┐   ┌─────────────────┐
│   Test Suite    │──▶│  Optimization   │──▶│   A/B Testing   │
│                 │   │   Algorithm     │   │                 │
│ • Test Cases    │   │ • Evolutionary  │   │ • Baseline vs   │
│ • Criteria      │   │ • Gradient      │   │   Optimized     │
│ • Benchmarks    │   │ • Random        │   │ • Statistical   │
└─────────────────┘   └─────────────────┘   └─────────────────┘
                               ▲
                               │
                      ┌─────────────────┐
                      │   Performance   │
                      │   Dashboard     │
                      │ • Metrics       │
                      │ • Improvements  │
                      │ • Trends        │
                      └─────────────────┘
```

11. DSPy Primer

Explanation

DSPy Primer provides a comprehensive introduction to the DSPy framework, covering its core concepts, components, and practical applications for building and optimizing LLM programs.

Example: DSPy Primer Program



```python
import dspy

# Set up DSPy configuration
dspy.settings.configure(lm=dspy.OpenAI(model='gpt-3.5-turbo'))

# Define a simple signature
class GenerateAnswer(dspy.Signature):
    """Answer questions with evidence."""

    context = dspy.InputField(desc="Context information")
    question = dspy.InputField(desc="Question to answer")
    answer = dspy.OutputField(desc="Answer with explanation")
    evidence = dspy.OutputField(desc="Supporting evidence")

# Create a DSPy module
class QAWithEvidence(dspy.Module):
    def __init__(self):
        super().__init__()
        self.generate = dspy.ChainOfThought(GenerateAnswer)

    def forward(self, context, question):
        return self.generate(context=context, question=question)

# Create program instance
qa_program = QAWithEvidence()

# Example usage
context = "Paris is the capital and most populous city of France."
question = "What is the capital of France?"

result = qa_program(context=context, question=question)
print(f"Answer: {result.answer}")
print(f"Evidence: {result.evidence}")

# Setup evaluation
def validate_qa(example, prediction, trace=None):
    """Validate QA response quality"""
    # Check if answer is correct
    correct = example.answer.lower() in prediction.answer.lower()

    # Check if evidence supports answer
    evidence_relevant = any(word in prediction.evidence.lower()
                          for word in example.answer.lower().split())

    return correct and evidence_relevant

# Optimize the program
optimizer = dspy.BootstrapFewShot(metric=validate_qa)
optimized_program = optimizer.compile(qa_program, trainset=train_data)
```

Sample Image

```
┌─────────────────────────────────────────────────────────────┐
│                     DSPy Architecture                       │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐          │
│  │   Modules   │  │ Signatures  │  │ Optimizers  │          │
│  │ • ChainOfThought │ • Input/Output │ • BootstrapFewShot │     │
│  │ • Predict    │ • Descriptions│ • MIPRO          │          │
│  │ • Retrieve   │ • Types      │ • BayesianOpt    │          │
│  └─────────────┘  └─────────────┘  └─────────────┘          │
├─────────────────────────────────────────────────────────────┤
│  ┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐│
│  │   Teleprompters │  │   Assertions    │  │   Suggestions   │  │
│  │ • LabeledFewShot│  │ • Quality       │  │ • Improvements  │  │
│  │ • BootstrapFewSh│  │ • Constraints   │  │ • Alternatives  │  │
│  └─────────────────┘  └─────────────────┘  └─────────────────┘│
├─────────────────────────────────────────────────────────────┤
│  Program Flow: Input ──▶ Module ──▶ Optimizer ──▶ Output      │
└─────────────────────────────────────────────────────────────┘
```

12. LLM & Image Model Performance: Advanced Evaluation Strategies

Explanation

Advanced evaluation strategies for LLMs and image models involve sophisticated techniques to assess multimodal performance, including cross-modal understanding, generation quality, and alignment between different modalities.

Example: Advanced Multimodal Evaluation

```python
import torch
from evaluation_metrics import (
    CLIPSimilarity, ImageQuality, TextQuality,
    CrossModalAlignment, GenerationDiversity
)

class AdvancedMultimodalEvaluator:
    def __init__(self):
        self.clip_similarity = CLIPSimilarity()
        self.image_quality = ImageQuality()
        self.text_quality = TextQuality()
        self.cross_modal_alignment = CrossModalAlignment()
        self.generation_diversity = GenerationDiversity()

    def evaluate_image_captioning(self, model, dataset):
        """Evaluate image-to-text generation performance"""
        results = {
            'clip_similarity': [],
            'text_quality': [],
            'relevance': [],
            'diversity': []
        }

        for item in dataset:
            # Generate caption for image
            generated_caption = model.generate_caption(item['image'])

            # Calculate CLIP similarity
            clip_score = self.clip_similarity(
                item['image'],
                generated_caption
            )
            results['clip_similarity'].append(clip_score)

            # Evaluate text quality
            text_score = self.text_quality(generated_caption)
            results['text_quality'].append(text_score)

            # Check relevance to ground truth
            relevance_score = self.evaluate_relevance(
                generated_caption,
                item['ground_truth_caption']
            )
            results['relevance'].append(relevance_score)

            # Measure diversity
            diversity_score = self.generation_diversity(generated_caption)
            results['diversity'].append(diversity_score)

        # Calculate averages
        final_results = {}
        for metric, scores in results.items():
            final_results[metric] = sum(scores) / len(scores)

        return final_results

    def evaluate_text_to_image(self, model, prompts):
        """Evaluate text-to-image generation performance"""
        results = {
            'image_quality': [],
            'prompt_alignment': [],
            'aesthetics': [],
            'diversity': []
        }

        for prompt in prompts:
            # Generate image from text
            generated_image = model.generate_image(prompt)

            # Evaluate image quality
            quality_score = self.image_quality(generated_image)
            results['image_quality'].append(quality_score)

            # Measure alignment with prompt
            alignment_score = self.cross_modal_alignment(
                prompt,
                generated_image
            )
            results['prompt_alignment'].append(alignment_score)

            # Assess aesthetic quality
            aesthetic_score = self.evaluate_aesthetics(generated_image)
            results['aesthetics'].append(aesthetic_score)

            # Measure generation diversity
            diversity_score = self.generation_diversity(generated_image)
            results['diversity'].append(diversity_score)

        # Calculate averages
        final_results = {}
        for metric, scores in results.items():
            final_results[metric] = sum(scores) / len(scores)

        return final_results

    def comprehensive_evaluation(self, model, test_data):
        """Run comprehensive multimodal evaluation"""
        print("Running Image Captioning Evaluation...")
        caption_results = self.evaluate_image_captioning(model, test_data['image_captioning'])

        print("Running Text-to-Image Evaluation...")
        image_results = self.evaluate_text_to_image(model, test_data['text_to_image'])

        # Combine results
        comprehensive_results = {
            'image_captioning': caption_results,
            'text_to_image': image_results,
            'overall_score': self.calculate_overall_score(caption_results, image_results)
        }

        return comprehensive_results

    def calculate_overall_score(self, caption_results, image_results):
        """Calculate weighted overall score"""
        # Define weights for different components
        weights = {
            'caption_clip': 0.2,
            'caption_quality': 0.2,
            'image_quality': 0.2,
            'image_alignment': 0.2,
            'diversity': 0.1,
            'aesthetics': 0.1
        }

        overall_score = (
            weights['caption_clip'] * caption_results['clip_similarity'] +
            weights['caption_quality'] * caption_results['text_quality'] +
            weights['image_quality'] * image_results['image_quality'] +
            weights['image_alignment'] * image_results['prompt_alignment'] +
            weights['diversity'] * (caption_results['diversity'] + image_results['diversity']) / 2 +
            weights['aesthetics'] * image_results['aesthetics']
        )

        return overall_score

# Example usage
evaluator = AdvancedMultimodalEvaluator()

# Sample test data
test_data = {
    'image_captioning': [
        {
            'image': 'path/to/image1.jpg',
            'ground_truth_caption': 'A beautiful sunset over the ocean'
        },
        # More test cases...
    ],
    'text_to_image': [
        'A serene lake surrounded by mountains at dawn',
        'A futuristic city with flying cars',
        # More prompts...
    ]
}

# Run comprehensive evaluation
results = evaluator.comprehensive_evaluation(model, test_data)

print("Evaluation Results:")
for category, scores in results.items():
    if isinstance(scores, dict):
        print(f"\n{category.upper()}:")
        for metric, score in scores.items():
            print(f"  {metric}: {score:.3f}")
    else:
        print(f"\nOVERALL SCORE: {scores:.3f}")
```

Sample Image

```
┌─────────────────────────────────────────────────────────────┐
│         Advanced Multimodal Evaluation Framework          │
├─────────────────────────────────────────────────────────────┤
│  ┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐│
│  │Image Captioning │  │Text-to-Image    │  │Cross-Modal      │  │
│  │• CLIP Similarity│  │• Quality        │  │• Alignment      │  │
│  │• Text Quality   │  │• Aesthetics     │  │• Consistency    │  │
│  │• Relevance      │  │• Diversity      │  │• Integration    │  │
│  └─────────────────┘  └─────────────────┘  └─────────────────┘│
├─────────────────────────────────────────────────────────────┤
│  ┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐│
│  │   Metrics       │  │   Weights       │  │   Overall       │  │
│  │ • Accuracy      │  │ • 20% CLIP      │  │   Score         │  │
│  │ • Diversity     │  │ • 20% Quality   │  │ • Weighted Avg  │  │
│  │ • Alignment     │  │ • 20% Alignment │  │ • Comprehensive │  │
│  └─────────────────┘  └─────────────────┘  └─────────────────┘│
├─────────────────────────────────────────────────────────────┤
│  Performance Radar Chart                                    │
│  ┌─────────────────────────────────────────────────────────┐  │
│  │                    •                                    │  │
│  │                 ◊     ◊                                 │  │
│  │              ◊           ◊                              │  │
│  │            ◊               ◊                            │  │
│  │          ◊                   ◊                          │  │
│  │        ◊                       ◊                        │  │
│  │      ◊                           ◊                      │  │
│  │    ◊                               ◊                    │  │
│  │  ◊                                   ◊                  │  │
│  │◊                                       ◊                │  │
│  │  ◊                                   ◊                  │  │
│  │    ◊                               ◊                    │  │
│  │      ◊                           ◊                      │  │
│  │        ◊                       ◊                        │  │
│  │          ◊                   ◊                          │  │
│  │            ◊               ◊                            │  │
│  │              ◊           ◊                              │  │
│  │                 ◊     ◊                                 │  │
│  │                    •                                    │  │
│  └─────────────────────────────────────────────────────────┘  │
│     CLIP   Quality   Alignment   Diversity   Aesthetics      │
└─────────────────────────────────────────────────────────────┘
```