Lecture 18: Cost, Performance & Token Optimization

🎯 Learning Objectives

By the end of this lecture, you should be able to:

- Understand the cost structure of LLM API calls.
- Identify performance bottlenecks in agentic workflows.
- Apply techniques to reduce latency and token usage.
- Profile and optimize agents for real-world deployment.

🧩 Key Concepts

Understanding LLM Cost Drivers

- LLM usage cost is typically based on **tokens processed**:
  - **Input tokens**: User prompt, memory, tool results.
  - **Output tokens**: LLM's generated content.
- Frequent re-prompting and long context windows increase cost rapidly.

Performance Metrics

- **Latency**: Total response time (LLM + tool calls).
- **Token usage**: Measured per interaction and session.
- **Throughput**: Number of completed tasks per time unit.
- **Reliability**: Number of successful vs. failed executions.

Optimization Techniques

- **Prompt Compression**: Shorten prompt instructions and memory.
- **Response Trimming**: Use stop sequences to reduce verbosity.
- **Tool Efficiency**: Avoid redundant tool calls.
- **Chunking Strategies**: Optimize document splitting and context loading.
- **Caching**: Reuse embedding or tool call results.

🛠️ Required Tools/Libraries

- Python
- OpenAI API (or compatible)
- LangChain (for token counting and caching)
- Optional: LLM token profiler or usage dashboard

```
pip install langchain openai tiktoken
```

Hands-on Exercise: Optimize Token Usage

Goal: Profile an agent and reduce its token usage by 30%.

Step 1: Measure current token usage

```
from langchain.callbacks import get_openai_callback

with get_openai_callback() as cb:
    result = agent.run("Summarize this 2-page article.")
    print(cb)

# Output includes total tokens, cost, and prompt/output breakdown
```

Step 2: Compress prompt + reduce memory

```
- Remove extra formatting, unnecessary instructions.
- Use condensed memory buffer (e.g., ConversationSummaryMemory).
- Replace full documents with extracted summaries.
```

Step 3: Re-run and compare

```
with get_openai_callback() as cb:
    optimized_result = agent.run("Summarize the article concisely.")
    print(cb)

# Compare token usage and cost before vs. after optimization
```

Bonus:

- Implement semantic caching: store LLM responses by embedding similarity.
- Add cost display to your agent's logs or dashboard.
- Explore GPT-3.5 vs. GPT-4 cost/performance trade-offs for different tasks.
