DEVELOPING WITH AI APIS (CONDENSED)

Module 7 — Harwell Prompt Engineering

LEARNING OBJECTIVES

By the end of this module you will be able to:

- Explain the “delta” between standard REST APIs and LLM APIs: stateful vs. stateless, non-determinism, streaming
- Describe why LLM APIs differ from typical REST: stateful conversations, non-deterministic responses
- Control non-determinism: temperature and top-p; when to use low vs. high
- Handle streaming responses and implications for UX (tokens/sec, buffering)
- Understand cost drivers: token economics (input vs. output), and how to reason about cost when designing features

BRIDGE FROM MODULE 6

What we learned yesterday:

- **MCP** connects AI to live systems
- MCP uses APIs under the hood

Frame explicitly:

- This is the **delta** — what's different from standard REST APIs
- Not a full SDK deep dive
- Focus on understanding differences

Today: Understand how LLM APIs work differently from standard REST.

THE PROBLEM: LLM APIS ARE DIFFERENT

Standard REST expectations:

- Stateless requests
- Deterministic responses
- Simple request/response
- Predictable costs

LLM APIs break these:

- ❌ Stateful conversations vs. stateless
- ❌ Non-deterministic responses
- ❌ Streaming chunks vs. complete response
- ❌ Token-based costs vs. request-based

Why it matters:

- ❌ Expect stateless → conversations won’t work
- ❌ Expect deterministic → confused by variation
- ❌ Don’t handle streaming → poor UX
- ❌ Ignore cost → high bills

STATEFUL VS. STATELESS: CONVERSATIONS

Standard REST:

- **Stateless**: Each request is independent
- Example: GET /users/123 → returns user data
- No memory between requests
- Simple, predictable

LLM APIs:

- **Stateful**: Conversations maintain context
- Example:
  - Request 1: “What is Spring Boot?”
  - Request 2: “How do I use it?” (refers to previous)
- **Conversation ID or message list**: Maintains context
- More complex, but enables natural conversations

HOW STATEFUL WORKS

Option 1: Conversation ID

- First request returns conversation_id
- Subsequent requests include conversation_id
- API maintains conversation state
- ✅ Simpler for client

Option 2: Message list

- You maintain message history
- Send full message list with each request
- You control state
- ✅ More control

When to use:

- **Stateless (one-off)**: Simple Q&A, no context needed
- **Stateful (conversation)**: Multi-turn dialogue, follow-up questions

NON-DETERMINISM: THE PROBLEM

Same prompt → different answers

- “Why is this happening?”
- “How do I control it?”

Why non-deterministic?

- LLMs are probabilistic, not deterministic
- They sample from probability distributions
- Same input can produce different outputs
- This is by design (creativity, variety)

CONTROLLING NON-DETERMINISM

Temperature: Controls randomness

- **Low (0-0.3)**: More deterministic, focused
- **Medium (0.5-0.7)**: Balanced
- **High (0.8-1.0)**: More creative, varied

Top-P (nucleus sampling): Controls diversity

- **Low (0.1-0.3)**: Focused on most likely tokens
- **High (0.9-1.0)**: More diverse options

When you need reproducibility:

- Set temperature=0
- Use same seed
- More deterministic (not perfect)

WHEN TO USE WHAT TEMPERATURE

Low temperature (0-0.3):

- ✅ Code generation
- ✅ Factual answers
- ✅ Tests
- ✅ When you need consistency

Medium temperature (0.5-0.7):

- ✅ General use
- ✅ Balanced responses

High temperature (0.8-1.0):

- ✅ Creative writing
- ✅ Brainstorming
- ✅ Varied responses

STREAMING: CHUNKED RESPONSES

The problem:

- Without streaming: Wait for full response
- ❌ Slow perceived performance
- ❌ User sees nothing until complete
- ❌ Poor UX for long responses

The solution:

- **Streaming**: Responses come in chunks (tokens)
- ✅ User sees progress immediately
- ✅ Better perceived performance
- ✅ Can cancel if needed

HOW STREAMING WORKS

Server-Sent Events (SSE) or WebSocket

- Tokens arrive incrementally
- Client buffers and displays as received
- **Tokens/sec**: Measure of streaming speed
- **Buffering**: Client may buffer before displaying

Implementation considerations:

- Handle partial responses
- Display incrementally
- Handle errors mid-stream
- Buffer for smooth display

COST: TOKEN ECONOMICS

The problem:

- Standard APIs: Cost per request (simple)
- LLM APIs: Cost per token (complex)
- ❌ Easy to underestimate costs
- ❌ Costs scale with usage

Token economics:

- **Input tokens**: What you send (prompt + context)
- **Output tokens**: What you receive (response)
- **Pricing**: Usually per 1K tokens
- **Input vs. output**: Output often more expensive

COST DRIVERS

What affects cost:

- **Length**: Longer prompts/responses = more tokens
- **Frequency**: More requests = higher cost
- **Model**: Different models have different prices
- **Context**: Including context increases input tokens

Cost optimization:

- ✅ Shorter prompts (remove unnecessary context)
- ✅ Cache responses when possible
- ✅ Use cheaper models when appropriate
- ✅ Monitor token usage
- ✅ Set usage limits

ROUGH COST ESTIMATES

Examples:

- 1000 tokens ≈ 750 words
- Example pricing: $0.01 per 1K input, $0.03 per 1K output
- Typical request: 500 input + 500 output = ~$0.02
- Scale: 1000 requests/day = ~$20/day = ~$600/month

Monitor and adjust:

- Track actual token usage
- Set usage limits
- Optimize prompts
- Choose models wisely

SUMMARY

1. **Stateful vs. stateless**: Conversations need state management
2. **Non-determinism**: Control with temperature/top-p
3. **Streaming**: Better UX, handle chunks incrementally
4. **Cost**: Token-based, monitor usage, optimize prompts

BRIDGE TO MODULE 8

What we've learned:

- **How** LLM APIs work differently from standard REST

What's next:

Module 8: The Future of AI & Development — what's coming next.

QUESTIONS?

Module 7 — Developing with AI APIs (Condensed)