Large Language Models for Software Engineers

Session 1: Understanding the Model & The Single Call

Monarch Wadia · monarchwadia.com

SECTION 1

Logistics & Goals

Course Structure and Learning Outcomes

Series Logistics & Structure

- **Course Format** : Three one-hour technical sessions with hands-on coding exercises.
- **Session Progression** :
  - **Session 1 (Today)** : The Single Call & Foundational Concepts.
  - **Session 2** : Multi-Turn Chatbots & State Management.
  - **Session 3** : Tool Calling & Autonomous Agent Loops.
- **Prerequisites** : Backend engineering experience and .NET / local development SDK.
- **AWS Bedrock Access** : Region ca-central-1 (Canada Central) with temporary workshop keys.

Workshop Series Learning Outcomes

- **1. Treat LLMs as Infrastructure** : View models as stateless HTTP microservices.
- **2. Build from First Principles** : Connect the Model, the Harness, and Tools.
- **3. Control Developer Agent Tools** : Diagnose and optimize coding tools (Cursor, Claude Code) through prompt and context mechanics.
- **4. Prevent Production Failures** : Design defenses against hallucinations, format drift, and token limits.

Session 1 Learning Outcomes

- **1. Master the Single Call** : Send structured HTTP requests and parse metadata.
- **2. Manage Memory in Code** : Store conversation state in application memory arrays.
- **3. Apply Single-Call Tasks** : Transform, classify, extract, summarize, evaluate, and expand data.
- **4. Separate Control Planes** : Author modular system prompts with explicit contracts.
- **5. Handle Failure Modes** : Prevent hallucinations, format drift, truncation, and variance.

SECTION 2

The Model & The Single Call

Core Architecture, Primitives, and Production Gotchas

The Engineering Mental Model

- **Standard Infrastructure** : An LLM is a standard backend component. Treat it like a database, cache, or web service.
- **Deterministic Integration** : Your application code controls data flow, authorization, and error handling.
- **Multiple Architecture Patterns** : Agents are only one architecture pattern. LLMs also classify, extract, and convert data.
- **Existing Skills Apply** : Standard software engineering principles ensure reliability, security, and performance.

The Three System Components

- **Standard System Terminology** : You do not need data science training. Use standard software engineering concepts.
- **The Three Core Components** :
  - **The Model** : A remote HTTP service that predicts next tokens.
  - **The Harness** : Your application code that manages state, flow, and logic.
  - **Tools** : Integrations that connect the model to databases and APIs.
- **Clear Separation of Concerns** : Keep business logic in the harness, not in model weights.

The Agent Loop

- **Four-Step Execution Loop** : An agent is software that runs a four-step loop:
  - **1. Send Request** : Send the prompt and history array over HTTP.
  - **2. Receive Output** : Read generated text and tool requests from the response.
  - **3. Execute Tools** : Run local database queries or API calls.
  - **4. Update History** : Append inputs, outputs, and tool results to the history array.
- **The Single Call Foundation** : Start with single API calls before you build loops.

The HTTP Request Payload

- **HTTP POST Interface** : The application sends a JSON payload containing an array of messages.
- **Three Message Roles** :
  - **system** : Defines instructions, security rules, and output schemas.
  - **user** : Contains input from clients or external systems.
  - **assistant** : Contains past responses from the model.
- **Inference Parameters** :
  - **temperature** : Controls randomness (use 0.0 for deterministic tasks).
  - **max_tokens** : Sets the maximum token count for the response.

```
{
  "model": "anthropic.claude-3-5-sonnet",
  "messages": [
    {
      "role": "system",
      "content": "You are a banking assistant."
    },
    {
      "role": "user",
      "content": "What is my TFSA balance?"
    }
  ],
  "temperature": 0.1,
  "max_tokens": 500
}
```

Response Payload Structure

- **Structured JSON Response** : The API returns metadata and a list of content blocks.
- **Content Block** : The response text is stored in the content array with role: 'assistant'.
- **Stop Reasons** :
  - **end_turn** : The model completed the response normally.
  - **max_tokens** : Generation stopped because output reached the token limit.
- **Usage Metrics** : Tracks input and output token counts for cost accounting.

```
{
  "id": "msg_01XYZ...",
  "type": "message",
  "role": "assistant",
  "content": [
    {
      "type": "text",
      "text": "Your TFSA balance is $32,450 CAD."
    }
  ],
  "stop_reason": "end_turn",
  "usage": {
    "input_tokens": 128,
    "output_tokens": 22
  }
}
```

The Conversation History Array

- **State in Application Memory** : Store the conversation history array in your application RAM or database.
- **Array Mutation Flow** :
  - **Turn 1** : Send the user message. Append the assistant response to the array.
  - **Turn 2** : Append the new user message. Send the full array to the API.
- **Stateless Context** : The model interprets new messages only because the request includes past turns.

```
// Turn 1: user sends request
"messages": [
  { "role": "user",
    "content": "What is my TFSA balance?" }
]

// After Turn 1: code appends response
"messages": [
  { "role": "user",
    "content": "What is my TFSA balance?" },
  { "role": "assistant",
    "content": "$32,450.22 CAD." }
]

// Turn 2: resend FULL history array
"messages": [
  { "role": "user",
    "content": "What is my TFSA balance?" },
  { "role": "assistant",
    "content": "$32,450.22 CAD." },
  { "role": "user",
    "content": "And my RRSP balance?" }
]
```

Tool Execution Overview

- **No Direct Database Access** : The remote model cannot read your database directly.
- **Execution Sequence** :
  - **1. Declare Tool Schema** : Define function names and parameter schemas in JSON.
  - **2. Model Emits Request** : The model returns a tool call request with parameters.
  - **3. Application Runs Query** : Your code executes the local database query.
  - **4. Return Result** : Append the tool result to the history array and call the model again.
- **Full Control** : Your application code executes all database queries.

```
{
  "tools": [
    {
      "name": "get_account_balance",
      "description": "Fetch live CAD balance.",
      "input_schema": {
        "type": "object",
        "properties": {
          "account_type": {
            "type": "string",
            "enum": ["TFSA", "RRSP"]
          }
        },
        "required": ["account_type"]
      }
    }
  ]
}
```

Hands-On: Your First API Call

- **Local Environment Verification** : Send an authenticated API request to AWS Bedrock using C#.
- **Step 1** : Extract workshop-1-code.zip to a local folder.
- **Step 2** : Add your AWS credentials to the .env file.
- **Step 3** : Run dotnet run in the terminal to execute the test call.
- **Step 4** : Change the prompt text in Program.cs and test custom inputs.

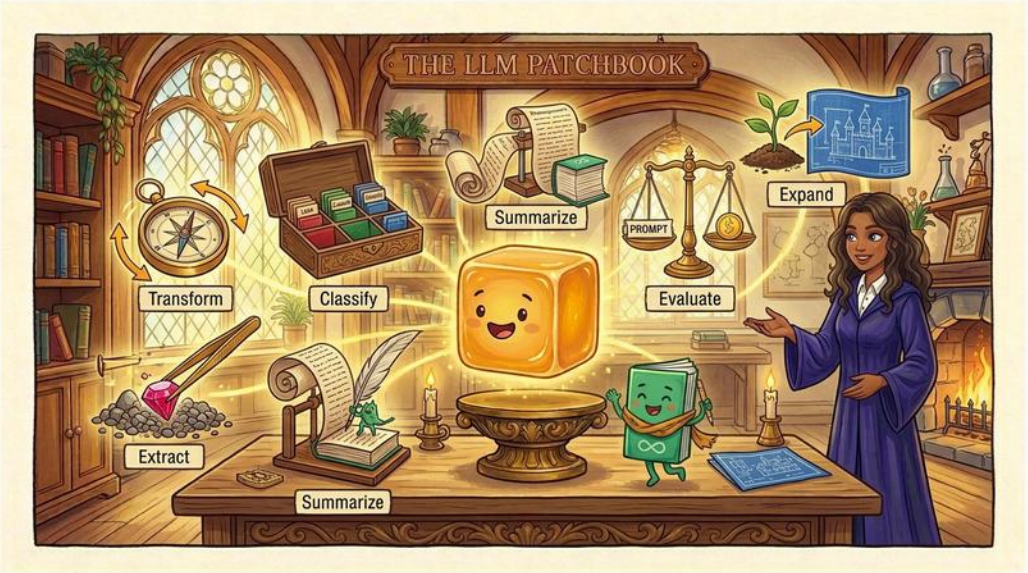
```
// dotnet run
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

var client = new AmazonBedrockRuntimeClient();
var request = new ConverseRequest
{
    ModelId = "anthropic.claude-3-5-sonnet",
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock>
            {
                new ContentBlock { Text = "Ping" }
            }
        }
    },
    InferenceConfig = new InferenceConfiguration
    {
        MaxTokens = 100,
        Temperature = 0.0f
    }
};

var response = await client.ConverseAsync(request);
Console.WriteLine(response.Output.Message.Content[0].Text);
```

What Can a Single Call Do?

- **No Agent Loop Required** : Solve common backend tasks with a single stateless request.
- **Replaces Custom ML Pipelines** : Use prompts instead of collecting training data and hosting custom models.
- **Six Common Tasks** :
  - **Transform** : Convert data representations and schemas.
  - **Classify** : Assign text to predefined categories.
  - **Extract** : Parse structured JSON from unstructured text.
  - **Summarize** : Condense documents while preserving key facts.
  - **Evaluate** : Validate data against compliance rules (LLM-as-a-judge).
  - **Expand** : Generate complete letters or reports from short notes.



Single Call Task: Transform

- **Function** : Converts input data from one format or syntax to another.
- **Common Software Use Cases** :
  - Convert natural language text into SQL or GraphQL queries.
  - Translate technical error messages into client-facing text.
  - Convert legacy data structures to modern API schemas.
- **Primary Benefit** : Requires no custom parsing grammar and handles ambiguous inputs.

```
// Request:
{
  "system": "Translate English to SQL.\n" +
    "Schema: trades(id, ticker, qty, cad_val)",
  "messages": [{
    "role": "user",
    "content": "Find trades > $100k CAD for XIU"
  }]
}

// LLM Output:
SELECT id, ticker, qty, cad_val
FROM trades
WHERE ticker = 'XIU'
  AND cad_val > 100000;
```

Single Call Task: Classify

- **Function** : Maps unstructured text into a fixed set of category labels.
- **Common Software Use Cases** :
  - Route customer support tickets to the correct department.
  - Classify financial transactions for accounting and tax rules.
  - Detect urgent compliance escalations and customer sentiment.
- **Primary Benefit** : Add or modify categories in the prompt without model retraining.

```
// Request:
{
  "system": "Classify message into category:\n" +
    "[TRADE_DISPUTE, TAX_INQUIRY, WIRE_TRANSFER]\n" +
    "Return JSON: {category, confidence}",
  "messages": [{
    "role": "user",
    "content": "Why was my $15k wire delayed?"
  }]
}

// LLM Output:
{
  "category": "WIRE_TRANSFER",
  "confidence": 0.98
}
```

Single Call Task: Extract

- **Function** : Extracts strongly typed fields from unstructured text into JSON.
- **Common Software Use Cases** :
  - Extract transaction amounts, dates, and account numbers from invoices.
  - Parse trade instructions and quantities from incoming emails.
  - Extract structured entity records from regulatory filing documents.
- **Primary Benefit** : Handles spelling errors, abbreviations, and format variations.

```
// Request:
{
  "system": "Extract trade orders to JSON:\n" +
    "[{ticker, action, qty}]",
  "messages": [{
    "role": "user",
    "content": "Buy 500 XIU and sell 200 VCN\n" +
      "for my RRSP."
  }]
}

// LLM Output:
[
  { "ticker": "XIU", "action": "BUY", "qty": 500 },
  { "ticker": "VCN", "action": "SELL", "qty": 200 }
]
```

Single Call Task: Summarize

- **Function** : Condenses long documents while retaining critical numbers and decisions.
- **Common Software Use Cases** :
  - Generate daily executive summaries of market news.
  - Condense advisor call notes into CRM activity records.
  - Create concise digests of financial prospectus filings.
- **Primary Benefit** : Enforces length constraints and preserves specific numerical facts.

```
// Request:
{
  "system": "Summarize policy in 1 bullet.\n" +
    "Keep key policy rates.",
  "messages": [{
    "role": "user",
    "content": "Bank of Canada cut policy rate\n" +
      "by 25 bps to 3.75%, citing core\n" +
      "inflation cooling to 2.3%."
  }]
}

// LLM Output:
- Bank of Canada cut rate by 25 bps
  to 3.75% as inflation cooled to 2.3%.
```

Single Call Task: Evaluate

- **Function** : Scores input text against explicit policy rules and constraints.
- **Common Software Use Cases** :
  - Validate transactions against anti-money laundering (AML) thresholds.
  - Check drafted advisor messages for compliance violations.
  - Run automated test assertions on software outputs (LLM-as-a-judge).
- **Primary Benefit** : Returns a boolean pass/fail status and an explanation.

```
// Request:
{
  "system": "Evaluate against FINTRAC AML rules.\n" +
    "Flag if cash >= $10k CAD.\n" +
    "Return JSON: {flagged: bool, reason: str}",
  "messages": [{
    "role": "user",
    "content": "Client deposited $12,500 cash."
  }]
}

// LLM Output:
{
  "flagged": true,
  "reason": "Deposit of $12,500 CAD exceeds\n" +
    "the $10,000 FINTRAC reporting threshold."
}
```

Single Call Task: Expand

- **Function** : Generates complete, natural text from concise bullet points or data records.
- **Common Software Use Cases** :
  - Draft client portfolio review letters from performance metrics.
  - Generate release notes from git commit logs and pull requests.
  - Draft initial customer service responses for human review.
- **Primary Benefit** : Converts raw metrics into clear human communication.

```
// Request:
{
  "system": "Wealth advisor persona.\n" +
    "Expand notes to client letter.\n" +
    "Tone: professional, concise.",
  "messages": [{
    "role": "user",
    "content": "Dr. Thorne: Shifted $50k\n" +
      "from US Eq to CAD Bonds."
  }]
}

// LLM Output:
Dear Dr. Thorne,

We completed your portfolio
rebalancing today. We shifted
$50,000 from US Equities into
Canadian Bonds to reduce
volatility and lock in yields.

Best regards,
Wealth Advisory Team
```

The Core Rule: LLMs Are Stateless

- **Zero Internal Memory** : The model retains no data between API requests: f(messages) -> message.
- **Architectural Benefits of Statelessness** :
  - **Fault Tolerance** : Retry failed requests without corrupting remote state.
  - **Auditability** : Maintain an exact record of all inputs and outputs.
  - **Dynamic Context** : Add or remove messages on any request.
  - **Branching and Replay** : Fork conversation threads and replay logs for automated tests.

System Prompts: The Control Plane

- **Separation of Planes** : Separate control instructions from untrusted data.
- **Control Plane (system)** : Developers define rules, guardrails, and output schemas.
- **Data Plane (user)** : Contains untrusted input from external users, emails, and webhooks.
- **Instruction Hierarchy** : Frontier models prioritize system rules over user input.

Structuring System Prompts: A Standard Pattern

- **Modular Prompt Design** : Divide system prompts into discrete, testable sections.
- **Four Common Sections** :
  - **1. Persona** : Sets the role, tone, and operational perspective.
  - **2. Runtime Context** : Injects environment variables, dates, and jurisdictions.
  - **3. Guardrails** : Defines explicit negative constraints and forbidden actions.
  - **4. Output Contract** : Specifies the exact JSON schema and required fields.
- **Extensible Architecture** : Adapt the structure to match system complexity.

```
payload = {
    "system": ""
# 1. Persona
Senior Compliance Officer.

# 2. Runtime Context
Jurisdiction: FINTRAC (Canada)
Date: 2026-08-20

# 3. Guardrails
- Flag cash deposits >= $10k CAD.
- Never provide tax advice.

# 4. Output Contract
Return JSON: {flagged: bool, reason: str}
"",
    "messages": [
        {"role": "user", "content": "Deposited $12k cash."}
    ]
}
```

System Prompts vs. Few-Shot Examples

- **Instruction by Rule (System)** : Sets global constraints, policies, and valid enums.
- **Instruction by Example (Few-Shot)** : Injects sample message turns into the history array.
- **Combined Strategy** : Use few-shot examples to clarify edge cases that rules describe poorly.
- **Runtime Configuration** : Guide model behavior at runtime without model fine-tuning.

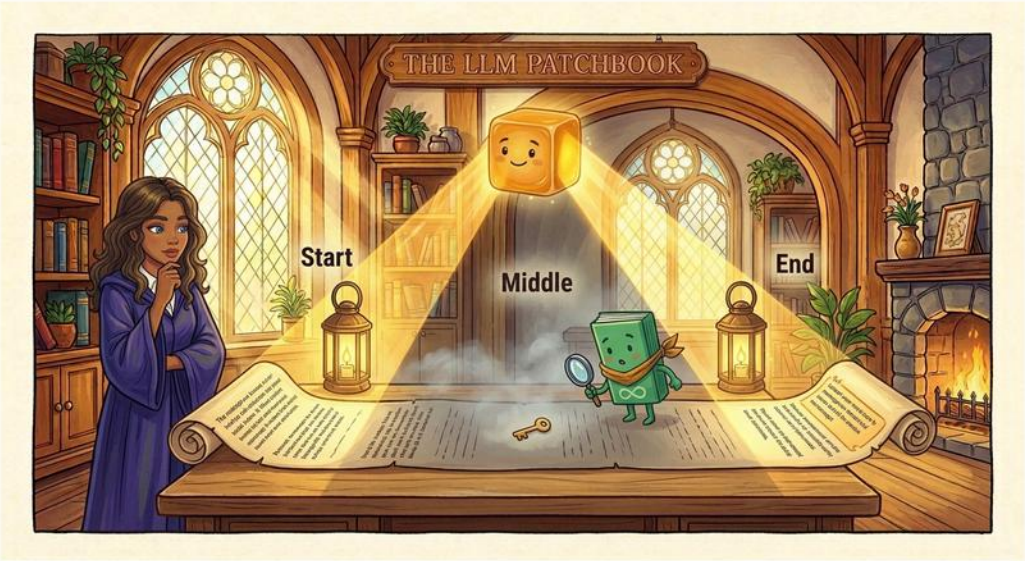
```
payload = {
    "system": ""
Classify tickers into asset class:
[EQUITY, FIXED_INCOME, CRYPTO]
Return JSON: {ticker: str, class: str}
"",
    "messages": [
        # Few-Shot 1: Demonstrates output schema
        {"role": "user", "content": "XIU.TO"},
        {"role": "assistant", "content":
            '{"ticker": "XIU.TO", "class": "EQUITY"}'},
        # Few-Shot 2: Edge-case demonstration
        {"role": "user", "content": "XBB.TO"},
        {"role": "assistant", "content":
            '{"ticker": "XBB.TO", "class": "FIXED_INCOME"}'},
        # Actual User Request
        {"role": "user", "content": "BTC-CAD"}
    ]
}
```

Context Windows: The Working RAM

- **Fixed Token Capacity** : The context window defines the maximum token budget for a single API call.
- **Shared Budget** : Total Capacity = Input Tokens + Output Buffer. Longer inputs reduce available output length.
- **Performance Impact** : Larger input prompts increase processing latency (TTFT) and API cost.

The 'Lost in the Middle' Effect

- **Non-Uniform Attention** : Models process tokens at different positions with varying attention weights.
- **The U-Shaped Attention Curve** : Recall accuracy is highest at the start and end of the prompt.
- **Placement Strategy** : Place critical instructions in the system prompt (top) or task prompt (bottom).



Gotcha 1: Hallucinations

- **Failure Mode** : The model generates plausible but incorrect facts with high confidence.
- **Root Cause** : Models predict likely token sequences. They do not query external facts automatically.
- **Prevention Methods** :
  - Add explicit rule: *'If facts are not in context, state that data is unavailable.'*
  - Inject verified reference documents (RAG) or query tools before generating factual answers.

```
// Request without source documents:
{
  "messages": [{
    "role": "user",
    "content": "What was our Q3 dividend yield?"
  }]
}

// LLM Output:
{
  "dividend_yield": "4.85%",
  "payout_date": "2026-10-15",
  "source": "Q3 Financial Summary"
}
```

Error: Hallucination. The model generated plausible values without reference data in the prompt.

Gotcha 2: Format Drift

- **Failure Mode** : The model encloses JSON in markdown fences (json) or adds conversational text.
- **Impact** : Application JSON parsers fail with parsing exceptions.
- **Prevention Methods** :
  - Add explicit rule: *'Output raw JSON only. Do not use markdown fences.'*
  - Sanitize and strip markdown code fences in application code before parsing.

```
// Request:
{
  "system": "Return raw JSON only: {id, status}",
  "messages": [{
    "role": "user",
    "content": "Order #448-9102"
  }]
}

// LLM Output:
```json
{
  "id": "448-9102",
  "status": "APPROVED"
}
```
```

Error: Format Drift. Output contains markdown code fences (```json). JSON parser failed.

Gotcha 3: Token Limit & Truncation

- **Failure Mode** : The model stops output generation in the middle of a sentence or JSON object.
- **Root Cause** : Generated output reached the configured max_tokens limit.
- **Prevention Methods** :
  - Check stop_reason in the response payload (end_turn vs max_tokens).
  - Set sufficient max_tokens limits for complex schema outputs.

```
// Request with low max_tokens limit:
{
  "max_tokens": 30,
  "messages": [{
    "role": "user",
    "content": "Audit transaction history."
  }]
}

// LLM Output:
{
  "audit_status": "COMPLIANT",
  "flagged_items": [
    { "tx_id": "tx_9918", "note": Cash
```

Error: Truncation. Output reached max_tokens limit before completion. stop_reason = 'max_tokens'.

Gotcha 4: Non-Determinism

- **Failure Mode** : The same prompt produces different outputs on repeated API calls.
- **Root Cause** : Setting temperature to 0.0 minimizes randomness, but GPU cluster concurrency causes slight variance.
- **Prevention Methods** :
  - Enforce strict JSON schemas and enum validation in the system prompt and few-shot examples.
  - Validate and normalize model outputs in application code before execution.

```
// Request sent twice with temperature = 0.0:
{
  "temperature": 0.0,
  "system": "Classify client request.",
  "messages": [{
    "role": "user",
    "content": "Reverse $45 USD wire fee."
  }]
}

// Call 1: {"category": "WIRE_FEE_DISPUTE"}
// Call 2: {"category": "ACCOUNT_MAINTENANCE"}
```

Error: Non-Determinism. Concurrency across GPU nodes produced different category labels.

SECTION 3

Recap & Knowledge Check

Summary, Review Questions, and Next Steps

Key Takeaways & Next Steps

- **Summary of Fundamentals** :
  - **Pure Stateless Functions** : Every call is independent: f(messages) -> message.
  - **Six Single-Call Tasks** : Transform, Classify, Extract, Summarize, Evaluate, and Expand.
  - **Control Plane Architecture** : System prompts enforce rules, tone, and JSON schema.
  - **Defensive Engineering** : Guard against hallucinations, format drift, truncation, and variance.
- **Application for Developer Tools** :
  - Explains context loss in long codebases (*Lost in the Middle*).
  - Explains tool parsing failures caused by markdown leakage (*Format Drift*).
- **Next Session** : Build a multi-turn chatbot with stateful conversation memory.

Check Your Understanding

1. Where is conversation history stored in a multi-turn LLM application?
2. What is the mathematical representation of a stateless LLM API call?
3. Which message role sets instructions and schemas with highest priority?
4. What does the max_tokens stop reason indicate in an API response?
5. Does the remote model execute tool code directly in your database?
6. Which single-call task extracts structured JSON fields from messy emails?
7. What happens to recall accuracy in the middle of a large context window?
8. Why do markdown code fences (json) cause backend JSON parsing errors?

Check Your Understanding: Answer Key (1–4)

1. Where is conversation history stored in a multi-turn LLM application?

➜ In Application RAM (or database). The application appends turns and resends the complete array.

2. What is the mathematical representation of a stateless LLM API call?

➜ f(messages) -> message. The remote model retains zero internal memory between HTTP requests.

3. Which message role sets instructions and schemas with highest priority?

➜ The system role. Frontier models prioritize system instructions and guardrails over user input.

4. What does the max_tokens stop reason indicate in an API response?

➜ Token Truncation. Output generation stopped early because it reached the configured token limit.

Check Your Understanding: Answer Key (5–8)

5. Does the remote model execute tool code directly in your database?
➜ No. The model emits a tool call request; your application code executes the query and returns results.

6. Which single-call task extracts structured JSON fields from messy emails?
➜ Extract. It pulls strongly typed fields into JSON without brittle regular expressions.

7. What happens to recall accuracy in the middle of a large context window?
➜ Recall drops. Attention follows a U-shaped curve (the 'Lost in the Middle' effect).

8. Why do markdown code fences (```json) cause backend JSON parsing errors?
➜ Format Drift. Markdown code delimiters break strict JSON deserializers (System.Text.Json).

Thank You

Questions? Comments? Concerns?

Monarch Wadia · monarchwadia.com