API Calls to Large Language Models

Seth J. Chandler, with help from AI

August 21, 2025

Contents

Chapter 1

Basic LLM API Call

This chapter provides a simple Python script demonstrating a basic API call to a Large Language Model (LLM). The script sends a user prompt to the model and retrieves the response, using minimal configuration options to illustrate the core functionality of an LLM API for educational purposes.

```python
import requests

# API endpoint and credentials (hypothetical)
API_URL = "https://api.x.ai/v1/chat/completions"
API_KEY = "your-api-key-here"  # Replace with actual API key

def basic_llm_api_call(user_prompt: str) -> str:
    """
    Make a basic API call to an LLM with a user prompt.

    Args:
        user_prompt: The user's input prompt

    Returns:
        The model's response as a string or an error message
    """
    headers = {
        "Authorization": f"Bearer {API_KEY}",
        "Content-Type": "application/json"
    }

    payload = {
        "model": "grok-3",
        "messages": [
            {"role": "user", "content": user_prompt}
        ]
    }

    try:
        response = requests.post(API_URL, headers=headers, json=payload)
        response.raise_for_status()
        return response.json()["choices"][0]["message"]["content"]
    except requests.exceptions.RequestException as e:
        return f"Error: {str(e)}"

# Example usage
if __name__ == "__main__":
    prompt = "Explain what a transformer model is in one sentence."
    result = basic_llm_api_call(prompt)
    print("Model Response:", result)
```

Listing 1.1: Simple Python script for an LLM API call

Chapter 2

Advanced LLM API Usage

This chapter presents a Python script that demonstrates how to interact with a Large Language Model (LLM) API using advanced configuration options. The code showcases parameters such as temperature, system prompts, output formats, and tool integration, providing insight into the inner workings of LLMs for educational purposes.

```python
import requests
import json
from typing import List, Dict, Any

# API endpoint and credentials (hypothetical)
API_URL = "https://api.x.ai/v1/chat/completions"
API_KEY = "your-api-key-here"  # Replace with actual API key

# Define a tool for function calling (e.g., a calculator tool)
tools = [
    {
        "type": "function",
        "function": {
            "name": "calculate",
            "description": "Perform mathematical calculations",
            "parameters": {
                "type": "object",
                "properties": {
                    "expression": {
                        "type": "string",
                        "description": "Mathematical expression to evaluate (e.g., '2 + 3 * 4')"
                    }
                },
                "required": ["expression"]
            }
        }
    }
]

def make_llm_api_call(
    user_prompt: str,
    system_prompt: str = "You are a helpful assistant with expertise in explaining technical concepts.",
    temperature: float = 0.7,
    max_tokens: int = 512,
    top_p: float = 0.9,
    presence_penalty: float = 0.1,
    frequency_penalty: float = 0.2,
    response_format: Dict[str, str] = {"type": "json_object"},
    tools: List[Dict[str, Any]] = None,
    tool_choice: str = "auto"
) -> Dict[str, Any]:
    """
```

```
    Make an API call to the LLM with advanced configuration options.

    Args:
        user_prompt: The user's input prompt
        system_prompt: System prompt to guide the model's behavior
        temperature: Controls randomness (0.0 to 2.0)
        max_tokens: Maximum number of tokens in the response
        top_p: Nucleus sampling parameter
        presence_penalty: Penalty for new tokens
        frequency_penalty: Penalty for frequent tokens
        response_format: Desired output format (e.g., JSON)
        tools: List of available tools for function calling
        tool_choice: Strategy for tool usage ('auto', 'none', or specific tool)

    Returns:
        API response as a dictionary
    """
    headers = {
        "Authorization": f"Bearer {API_KEY}",
        "Content-Type": "application/json"
    }

    payload = {
        "model": "grok-3",
        "messages": [
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": user_prompt}
        ],
        "temperature": temperature,
        "max_tokens": max_tokens,
        "top_p": top_p,
        "presence_penalty": presence_penalty,
        "frequency_penalty": frequency_penalty,
        "response_format": response_format,
        "tools": tools,
        "tool_choice": tool_choice
    }

    try:
        response = requests.post(API_URL, headers=headers, json=payload)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        return {"error": str(e)}

# Example usage
def demonstrate_llm_api():
    user_prompt = """
    Explain how a transformer model processes input for a college audience.
    Include a simple calculation example: what is 5 + 3 * 4?
    Return the response in JSON format with sections for 'explanation' and 'calculation'.
    """

    # Make the API call with custom parameters
    response = make_llm_api_call(
        user_prompt=user_prompt,
        system_prompt="""
        You are a patient, clear instructor teaching college students.
        Break down complex concepts into simple terms and provide structured JSON output.
        Use the calculator tool for any mathematical operations.
        """,
        temperature=0.8,  # Moderate randomness for clear but varied responses
        max_tokens=1000,  # Allow longer responses for detailed explanations
        top_p=0.95,       # Consider 95% of probability mass for token sampling
        presence_penalty=0.3,  # Encourage new topics
        frequency_penalty=0.4,  # Reduce repetition
        response_format={
            "type": "json_object",
```

```
            "schema": {
                "type": "object",
                "properties": {
                    "explanation": {"type": "string"},
                    "calculation": {"type": "string"}
                },
                "required": ["explanation", "calculation"]
            }
        },
        tools=tools,
        tool_choice="auto"  # Let the model decide when to use the calculator tool
    )

    # Process and display the response
    if "error" in response:
        print(f"Error: {response['error']}")
    else:
        print("API Response:")
        print(json.dumps(response, indent=2))

        # Handle tool call if present
        if "choices" in response and response["choices"][0].get("tool_calls"):
            tool_call = response["choices"][0]["tool_calls"][0]
            if tool_call["function"]["name"] == "calculate":
                expression = json.loads(tool_call["function"]["arguments"])["expression"]
                print(f"Tool called: calculate({expression})")
                # Simulate tool execution (in reality, you'd evaluate the expression)
                result = eval(expression, {"__builtins__": {}})  # Safe eval for demo
                print(f"Tool result: {result}")

if __name__ == "__main__":
    demonstrate_llm_api()
```

Listing 2.1: Python script for advanced LLM API call

Chapter 3

Basic LLM API Call with Web Search

This chapter presents a simple Python script that demonstrates an API call to a Large Language Model (LLM) endpoint with integrated web search capabilities. The script sends a user query to retrieve real-time web information and returns a summarized response, illustrating the core functionality of a web search-enabled LLM API for educational purposes.

The API endpoint used in this example is a hypothetical `https://api.x.ai/v1/web-search/completions`, designed to mimic real-world APIs like Tavily or SerpAPI, which provide web search results as context for LLMs. The script uses minimal parameters: only the user query and a limit on the number of web results (set to 5 for brevity), making it accessible for students. The response is extracted from a JSON structure, assuming the LLM summarizes the web search results. For real-world usage, the API URL and key would need to be replaced with values from a service like Tavily (`https://tavily.com`) or SerpAPI (`https://serpapi.com`). For xAI-specific APIs, students should refer to `https://x.ai/api`, as no public xAI web search API is available as of August 21, 2025.

```python
import requests
import json

# API endpoint and credentials (hypothetical)
API_URL = "https://api.x.ai/v1/web-search/completions"
API_KEY = "your-api-key-here"  # Replace with actual API key

def web_search_llm_api_call(user_query: str) -> str:
    """
    Make a basic API call to an LLM with web search capabilities.

    Args:
        user_query: The user's search query

    Returns:
        The model's summarized response based on web search results or an error message
    """
    headers = {
        "Authorization": f"Bearer {API_KEY}",
        "Content-Type": "application/json"
    }

    payload = {
        "model": "grok-3-web",
        "query": user_query,
        "max_results": 5  # Limit to 5 web search results for context
    }

    try:
        response = requests.post(API_URL, headers=headers, json=payload)
        response.raise_for_status()
        return response.json()["response"]["content"]
    except requests.exceptions.RequestException as e:
        return f"Error: {str(e)}"
```

```
35
36 # Example usage
37 if __name__ == "__main__":
38     query = "What are the latest advancements in transformer models?"
39     result = web_search_llm_api_call(query)
40     print("Model Response:", result)
```

Listing 3.1: Python script for basic LLM API call with web search

Chapter 4

Basic LLM API Call in Multiple Languages

This chapter presents a simple API call to a Large Language Model (LLM) implemented in six programming languages: JavaScript, Node.js, C#, Go, TypeScript, and Wolfram Language. Each example sends a user prompt to a hypothetical LLM endpoint (`https://api.x.ai/v1/chat/completions`) and retrieves the response, using minimal configuration to illustrate the core functionality for educational purposes. The Wolfram Language example leverages the high-level `ChatSubmit` function for a terser implementation. The code blocks are labeled by language for clarity.

4.1 JavaScript

This example uses the `fetch` API in JavaScript to make an HTTP POST request, suitable for browser-based environments.

```javascript
// API endpoint and credentials (hypothetical)
const API_URL = "https://api.x.ai/v1/chat/completions";
const API_KEY = "your-api-key-here"; // Replace with actual API key

async function basicLLMApiCall(userPrompt) {
    // Make a basic API call to an LLM
    const headers = {
        "Authorization": `Bearer ${API_KEY}`,
        "Content-Type": "application/json"
    };

    const payload = {
        model: "grok-3",
        messages: [{ role: "user", content: userPrompt }]
    };

    try {
        const response = await fetch(API_URL, {
            method: "POST",
            headers: headers,
            body: JSON.stringify(payload)
        });
        if (!response.ok) throw new Error(`HTTP error: ${response.status}`);
        const data = await response.json();
        return data.choices[0].message.content;
    } catch (error) {
        return `Error: ${error.message}`;
    }
}

// Example usage
```

```
(async () => {
    const prompt = "Explain what a transformer model is in one sentence.";
    const result = await basicLLMApiCall(prompt);
    console.log("Model Response:", result);
})();
```

Listing 4.1: JavaScript implementation of basic LLM API call

4.2 Node.js

This example uses the `axios` library in Node.js for the HTTP request, suitable for server-side environments.

```
const axios = require('axios');

// API endpoint and credentials (hypothetical)
const API_URL = 'https://api.x.ai/v1/chat/completions';
const API_KEY = 'your-api-key-here'; // Replace with actual API key

async function basicLLMApiCall(userPrompt) {
    // Make a basic API call to an LLM
    const headers = {
        'Authorization': `Bearer ${API_KEY}`,
        'Content-Type': 'application/json'
    };

    const payload = {
        model: 'grok-3',
        messages: [{ role: 'user', content: userPrompt }]
    };

    try {
        const response = await axios.post(API_URL, payload, { headers });
        return response.data.choices[0].message.content;
    } catch (error) {
        return `Error: ${error.message}`;
    }
}

// Example usage
(async () => {
    const prompt = 'Explain what a transformer model is in one sentence.';
    const result = await basicLLMApiCall(prompt);
    console.log('Model Response:', result);
})();
```

Listing 4.2: Node.js implementation of basic LLM API call

4.3 C#

This example uses `HttpClient` in C# to make the API call, suitable for .NET applications.

```
using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using System.Text.Json;

class Program {
    // API endpoint and credentials (hypothetical)
    private static readonly string API_URL = "https://api.x.ai/v1/chat/completions";
    private static readonly string API_KEY = "your-api-key-here"; // Replace with actual API key
```

```
11
12      static async Task<string> BasicLLMApiCall(string userPrompt) {
13          // Make a basic API call to an LLM
14          using (var client = new HttpClient()) {
15              client.DefaultRequestHeaders.Add("Authorization", $"Bearer {API_KEY}");
16
17              var payload = new {
18                  model = "grok-3",
19                  messages = new[] { new { role = "user", content = userPrompt } }
20              };
21
22              var content = new StringContent(JsonSerializer.Serialize(payload), Encoding.UTF8,
                    "application/json");
23
24              try {
25                  var response = await client.PostAsync(API_URL, content);
26                  response.EnsureSuccessStatusCode();
27                  var jsonResponse = await response.Content.ReadAsStringAsync();
28                  var data = JsonSerializer.Deserialize<JsonElement>(jsonResponse);
29                  return
                        data.GetProperty("choices")[0].GetProperty("message").GetProperty("content").GetString();
30              } catch (Exception ex) {
31                  return $"Error: {ex.Message}";
32              }
33          }
34      }
35
36      // Example usage
37      static async Task Main() {
38          string prompt = "Explain what a transformer model is in one sentence.";
39          string result = await BasicLLMApiCall(prompt);
40          Console.WriteLine("Model Response: " + result);
41      }
42  }
```

Listing 4.3: C# implementation of basic LLM API call

4.4 Go

This example uses the `net/http` package in Go to make the API call, suitable for lightweight server applications.

```
1   package main
2
3   import (
4       "bytes"
5       "encoding/json"
6       "fmt"
7       "net/http"
8   )
9
10  // API endpoint and credentials (hypothetical)
11  const apiURL = "https://api.x.ai/v1/chat/completions"
12  const apiKey = "your-api-key-here" // Replace with actual API key
13
14  type Message struct {
15      Role    string `json:"role"`
16      Content string `json:"content"`
17  }
18
19  type Payload struct {
20      Model    string    `json:"model"`
21      Messages []Message `json:"messages"`
22  }
23
```

```go
type Response struct {
    Choices []struct {
        Message struct {
            Content string `json:"content"`
        } `json:"message"`
    } `json:"choices"`
}

func basicLLMApiCall(userPrompt string) (string, error) {
    // Make a basic API call to an LLM
    payload := Payload{
        Model: "grok-3",
        Messages: []Message{{Role: "user", Content: userPrompt}},
    }

    payloadBytes, err := json.Marshal(payload)
    if err != nil {
        return "", err
    }

    req, err := http.NewRequest("POST", apiURL, bytes.NewBuffer(payloadBytes))
    if err != nil {
        return "", err
    }

    req.Header.Set("Authorization", "Bearer "+apiKey)
    req.Header.Set("Content-Type", "application/json")

    client := &http.Client{}
    resp, err := client.Do(req)
    if err != nil {
        return "", err
    }
    defer resp.Body.Close()

    if resp.StatusCode != http.StatusOK {
        return "", fmt.Errorf("HTTP error: %d", resp.StatusCode)
    }

    var response Response
    err = json.NewDecoder(resp.Body).Decode(&response)
    if err != nil {
        return "", err
    }

    return response.Choices[0].Message.Content, nil
}

func main() {
    // Example usage
    prompt := "Explain what a transformer model is in one sentence."
    result, err := basicLLMApiCall(prompt)
    if err != nil {
        fmt.Println("Error:", err)
    } else {
        fmt.Println("Model Response:", result)
    }
}
```

Listing 4.4: Go implementation of basic LLM API call

4.5 TypeScript

This example uses TypeScript with the `fetch` API, adding type definitions for better type safety, suitable for browser-based applications.

```typescript
// API endpoint and credentials (hypothetical)
const API_URL: string = "https://api.x.ai/v1/chat/completions";
const API_KEY: string = "your-api-key-here"; // Replace with actual API key

interface Message {
    role: string;
    content: string;
}

interface Payload {
    model: string;
    messages: Message[];
}

interface ApiResponse {
    choices: { message: { content: string } }[];
}

async function basicLLMApiCall(userPrompt: string): Promise<string> {
    // Make a basic API call to an LLM
    const headers: HeadersInit = {
        "Authorization": `Bearer ${API_KEY}`,
        "Content-Type": "application/json"
    };

    const payload: Payload = {
        model: "grok-3",
        messages: [{ role: "user", content: userPrompt }]
    };

    try {
        const response = await fetch(API_URL, {
            method: "POST",
            headers,
            body: JSON.stringify(payload)
        });
        if (!response.ok) throw new Error(`HTTP error: ${response.status}`);
        const data: ApiResponse = await response.json();
        return data.choices[0].message.content;
    } catch (error: any) {
        return `Error: ${error.message}`;
    }
}

// Example usage
(async () => {
    const prompt: string = "Explain what a transformer model is in one sentence.";
    const result: string = await basicLLMApiCall(prompt);
    console.log("Model Response:", result);
})();
```

Listing 4.5: TypeScript implementation of basic LLM API call

4.6 Wolfram Language

This example uses the high-level `ChatSubmit` function in Wolfram Language, specifying the model and API key via `LLMConfiguration`, to make a concise API call, suitable for Mathematica environments.

```
(* API credentials and configuration (hypothetical) *)
apiKey = "your-api-key-here"; (* Replace with actual API key *)

basicLLMApiCall[userPrompt_] := Module[{response},
    (* Make a basic API call to an LLM *)
    response = ChatSubmit[
        userPrompt,
        LLMConfiguration[
            <|
                "Model" -> "grok-3",
                "Service" -> "Custom",
                "BaseURL" -> "https://api.x.ai/v1/chat/completions",
                "APIKey" -> apiKey
            |>
        ]
    ];

    If[FailureQ[response],
        "Error: " <> ToString[response["Message"]],
        response["Content"]
    ]
]

(* Example usage *)
prompt = "Explain what a transformer model is in one sentence.";
result = basicLLMApiCall[prompt];
Print["Model Response: ", result]
```

Listing 4.6: Wolfram Language implementation of basic LLM API call