CS 203 - Computer Science Techn

Week 09: Getting Started with LLMs and Prompting

Enhanced Edition: Context Windows, APIs & Hands-On Practice

Computer Science Department

2025 Academic Year

Lecture: 1.5 hours | Lab: 2 hours

Enhanced Learning Objectives

What You'll Master Today

Core Concepts

- Understanding LLM fundamentals and architecture
- Mastering tokenization and text generation processes
- Deep-dive into context windows across model families
- Exploring LLM capabilities and current limitations
- Learning advanced prompting techniques

Practical Skills

- Chat Completion API fundamentals and best practices
- Hands-on LiteLLM programming with real examples
- Parameter tuning and optimization techniques
- Error handling and troubleshooting
- Building robust LLM applications

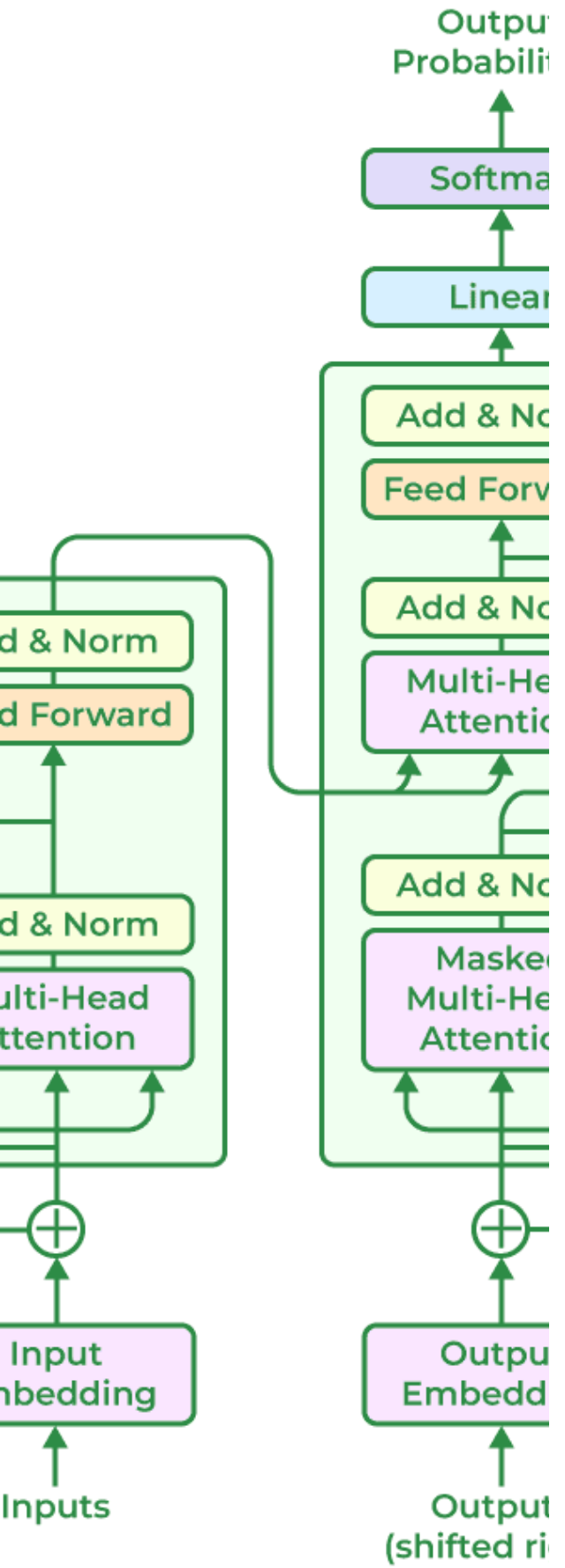
Session Structure: Lecture (1.5h) → Hands-on Lab (2h) → Q&A (30min)

What are Large Language Models?

- AI systems trained on vast amounts of text data
- Neural networks with billions to trillions of parameters
- Capable of understanding and generating human-like text
- Built primarily on Transformer architecture

- **Scale:** Trained on petabytes of text from the internet
- **Emergent Abilities:** Capabilities that emerge at scale
- **Few-shot Learning:** Can learn new tasks from examples
- **Generalization:** Apply knowledge across domains

> 💡 **Key Insight:** LLMs learn patterns in language by predicting the next word in a sequence, developing sophisticated understanding of grammar, facts, and reasoning.



Major LLM Families - 2025 Overview

Understanding the Leading Language Models

GPT Family (OpenAI)

- GPT-3.5: Fast, cost-effective, 4K-16K context
- GPT-4: Advanced reasoning, 8K-128K context
- **Strengths:** Code generation, reasoning, versatility
- **Best for:** Development, analysis, creative tasks

Claude Family (Anthropic)

- Claude 3 Haiku: Fast, efficient, 200K context
- Claude 3 Sonnet: Balanced performance, 200K context
- Claude 3 Opus: Top-tier capabilities, 200K context
- **Strengths:** Safety, long documents, nuanced reasoning

Mistral Family

- Mistral 7B: Efficient, open-source, 32K context
- Mistral Large: Enterprise-grade, 32K context
- **Strengths:** Multilingual, code, open-source availability
- **Best for:** European markets, privacy-focused applications

LLaMA Family (Meta)

- LLaMA 2: Research-focused, 4K context
- Code Llama: Specialized for programming, 16K context
- **Strengths:** Open research, customization, code tasks
- **Best for:** Research, fine-tuning, code generation

🎯 **Coming Next:** Deep dive into context windows for each family

LLM Architecture & Key Differences

Understanding the Technical Foundation

Transformer Foundation

Self-attention mechanisms enable parallel processing

Scale Matters

Model performance scales with parameters, data, and compute

Emergent Abilities

New capabilities appear at certain scale thresholds

Key Architectural Differences

Model Size

- **Small:** 7B-13B parameters (efficient, fast)
- **Medium:** 30B-70B parameters (balanced)
- **Large:** 175B+ parameters (highest capability)

Training Approaches

- **Base models:** Raw text prediction
- **Instruct models:** Fine-tuned for following instructions
- **RLHF models:** Human feedback optimization

Specialized Variants

- **Code-specialized:** Trained on programming languages
- **Multilingual:** Optimized for multiple languages
- **Long-context:** Enhanced for processing long documents
- **Reasoning-focused:** Improved logical and mathematical capabilities

> 💡 **Technical Insight:** The same Transformer architecture underlies all major LLMs, but differences in scale, training data, and fine-tuning create distinct capabilities and use cases.

Real-World LLM Applications

How LLMs Are Transforming Industries

💻 Software Development

- Code generation and completion
- Bug detection and fixing
- Documentation writing
- Code review and optimization

🏭 Business & Enterprise

- Customer service chatbots
- Content creation and marketing
- Data analysis and reporting
- Email automation and summarization

📋 Content Creation

- Writing assistance and editing
- Translation and localization
- Social media content generation
- Technical documentation

🗺️ Research & Education

- Literature review and synthesis
- Personalized tutoring systems
- Research paper assistance
- Educational content creation

🔍 Data Analysis

- Text mining and sentiment analysis
- Report generation from data
- Pattern recognition in documents
- Automated insights extraction

🌐 Creative Industries

- Screenplay and story writing
- Game narrative development
- Marketing copy creation
- Creative brainstorming assistance

🎯 **Industry Impact:** LLMs are estimated to affect 80% of jobs positively, augmenting human capabilities rather than replacing workers.

How LLMs Generate Text - The Complete Process

From Input to Output: Understanding Each Step

1 Input Processing

- User provides prompt or query
- Text is received by the API endpoint
- Context and conversation history are gathered

2 Tokenization

- Text is broken into tokens (subword units)
- Special tokens added (start, end, system)
- Token IDs mapped for model processing

3 Model Processing

- Tokens pass through transformer layers
- Self-attention mechanisms identify relationships
- Neural network computes probability distributions

4 Next-Token Prediction

- Model predicts most likely next token
- Sampling parameters (temperature, top-p) applied
- Token selected based on probability and randomness

5 Output Generation

- Process repeats until stop condition met
- Tokens are decoded back to human text

Tokenization Fundamentals

Breaking Text Into Processing Units

What Are Tokens?

- Smallest units of text that LLMs can process.
- Not always words: Can be parts of words, whole words, or punctuation.
- Subword tokenization: Most modern LLMs use BPE (Byte-Pair Encoding).
- Language agnostic: Works across different languages and scripts.

Tokenization Examples

"Hello, world!"

```
["Hello", ",", " world", "!"]
```

→ Token count: 4 tokens

"Understanding tokenization"

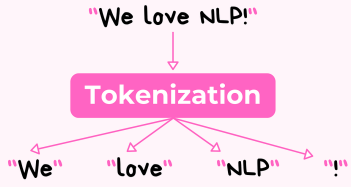
```
["Under", "standing", " token", "ization"]
```

→ Token count: 4 tokens

"GPT-4 is amazing!"

```
["GPT", "-", "4", " is", " amazing", "!"]
```

→ Token count: 6 tokens



Key Implications

- **Context Limits:** Token count determines what fits in a context window.
- **Cost Factor:** Many APIs charge per token processed.
- **Performance Impact:** More tokens can lead to longer processing times.
- **Language Efficiency:** Some languages are more "token-efficient" than others.

🛠️ **Pro Tip:** Use tokenizer tools (like tiktoken for OpenAI models) to count tokens before API calls to optimize cost and performance.

Tokenization in Practice - Real Examples

See How Different Texts Are Tokenized

Example 1: Simple Text

Input:

```
"The quick brown fox"
```

GPT-3.5 Tokens:

`"The"` `" quick"` `" brown"` `" fox"`

4 tokens

Example 2: Technical Terms

Input:

```
"API authentication token"
```

GPT-3.5 Tokens:

`"API"` `" authentication"` `" token"`

3 tokens

Example 3: Numbers & Special Chars

Input:

```
"Price: $29.99 (USD)"
```

GPT-3.5 Tokens:

`"Price"` `":"` `" $"` `"29"` `"."` `"99"` `" ("` `"USD"` `")"`

9 tokens

Example 4: Code Snippet

Input:

```
"def hello(): print('Hi!')"
```

GPT-3.5 Tokens:

`"def"` `" hello"` `"():"` `" print"` `"('"` `"Hi"` `"!')"` `"\n"`

8 tokens

Language Comparison

English: "Hello world" → 2 tokens

Spanish: "Hola mundo" → 3 tokens

Chinese: "你好世界" → 4 tokens

Code: `print("hello")` → 4 tokens

🔍 Observations

- Spaces are often part of tokens (e.g., `" quick"`).
- Punctuation usually gets separate tokens.
- Familiar words are more efficiently tokenized.
- Code and technical terms may use more tokens.

Next-Token Prediction: The Core Mechanism

How LLMs Decide What Comes Next

Input Context: **"The capital of France is"**

1 Model Processing

- All input tokens processed simultaneously.
- Context is built via self-attention layers.
- Calculates a probability distribution for the entire vocabulary.

→

2 Probability Distribution

→

3 Token Selection

Temp = 0: Always picks "Paris" (deterministic).
Temp = 0.7: Weighted random choice (balanced).
Temp = 1.2: More random, creative output.

→

4 Process Repeats

The output becomes new input:

```
"The capital of France is Paris"
```

...and the model predicts the next token (e.g. a period).

Temperature (0.0 - 2.0)

Lower: More focused, predictable.
Higher: More creative, random.

Top-p Sampling (0.1 - 1.0)

Considers tokens with cumulative probability up to 'p'. Filters out highly unlikely tokens.

Max Tokens

Sets the maximum length of the generated response. Prevents infinite generation loops.

🧠 **Core Insight:** Every word you see from an LLM was chosen from thousands of possibilities, with the model weighing context, training, and randomness parameters to make each selection.

Context Windows: The Foundation of LLM Memory

Understanding How Much LLMs Can Remember

What is a Context Window?

- **Definition:** The maximum amount of text (in tokens) an LLM can process at once.
- **Includes:** Your prompt + conversation history + generated response.
- **Limitation:** Fixed size determined by the model's architecture.
- **Behavior:** When exceeded, oldest tokens are "forgotten" (sliding window).

Why Context Windows Matter

📈 **Task Capability**

💰 **Cost Implications**

🗺️ **Use Case Selection**

Visual Metaphor

Think of it like a whiteboard:

Small Context

(4K)

- ✓ Fits a short essay or brief conversation
- ✓ Good for quick tasks and simple queries

Medium Context

(32K)

- ✓ Fits a research paper or long document
- ✓ Ideal for code projects & doc analysis

Large Context

(200K)

- ✓ Fits a small book or extensive dialogue
- ✓ Perfect for complex, long-form analysis

💡 **Critical Insight:** Context window size is often the primary factor in choosing an LLM. It's not just about intelligence, but about memory capacity and its suitability for a specific task.

GPT Family Context Windows

OpenAI's Approach to Context Management

GPT-3.5-Turbo (Standard)

Context Window: 4,096 tokens (~3,000 words)

Best For: Quick queries, simple tasks, cost-effective solutions

Use Cases: Chatbots, simple content generation, basic Q&A

Cost: Most economical option

GPT-3.5-Turbo-16k

Context Window: 16,384 tokens (~12,000 words)

Best For: Medium documents, code analysis, longer conversations

Use Cases: Document summarization, code review, extended dialogue

Cost: 2x standard GPT-3.5 pricing

GPT-4 (Standard)

Context Window: 8,192 tokens (~6,000 words)

Best For: Complex reasoning, high-quality output, creative tasks

Use Cases: Analysis, creative writing, complex problem-solving

Cost: Premium pricing for superior capability

GPT-4-32k

Context Window: 32,768 tokens (~24,000 words)

Best For: Large documents, extensive code bases, comprehensive analysis

Use Cases: Book summarization, large codebase analysis

Cost: Highest tier pricing

GPT-4-Turbo-128k

Context Window: 128,000 tokens (~96,000 words)

Best For: Massive documents, entire codebases, comprehensive research

Use Cases: Academic papers, full application analysis

Cost: Premium with volume discounts

Practical Examples

- **4K Context:** Email responses, short articles, basic conversations.
- **16K Context:** Research papers, code files, detailed analysis.
- **32K Context:** Technical manuals, large datasets, extensive documentation.
- **128K Context:** Entire books, complete applications, comprehensive reviews.

🎯 Strategic Insights

Strategy: Start with smaller context models for prototyping, then scale up based on actual needs. GPT-3.5-16k often provides the best price/performance ratio for most applications.

Claude Family Context Windows

Anthropic's Long-Context Champions

Claude 3 Haiku

Context Window: 200,000 tokens (~150,000 words)

Positioning: Fast, efficient, cost-effective

Strengths: Speed, affordability with massive context

Best For: Document processing, customer service, content moderation

Performance: Optimized for throughput and cost efficiency

Claude 3 Sonnet

Context Window: 200,000 tokens (~150,000 words)

Positioning: Balanced performance and intelligence

Strengths: Versatility, reasoning, creative tasks

Best For: Analysis, writing, research, general productivity

Performance: Sweet spot for most complex tasks

Claude 3 Opus

Context Window: 200,000 tokens (~150,000 words)

Positioning: Highest capability, premium performance

Strengths: Advanced reasoning, nuanced understanding, creativity

Best For: Complex analysis, research, high-stakes applications

Performance: Top-tier intelligence with massive context

Consistent Long Context | Constitutional AI | Document Excellence | Nuanced Reasoning | Multilingual

What 200K Tokens Means

- ~400-500 pages of standard text
- Entire novels like "The Great Gatsby"
- Complete academic papers with references
- Large codebases with multiple files
- Extended conversations with full history retention

Claude Excels At:

- Legal document analysis (contracts, case law)
- Academic research (literature reviews, thesis work)
- Content strategy (analyzing competitor content)
- Code review (entire application analysis)
- Long-form creative projects (novels, screenplays)

Mistral Family Context Windows

European Open-Source Excellence

Mistral AI

Mistral 7B

Context Window: 32,768 tokens (~24,000 words)

Parameters: 7 billion (efficient architecture)

License: Apache 2.0 (fully open-source)

Best For: Research, fine-tuning, cost-sensitive applications

Deployment: Can run on consumer hardware

Languages: Excellent multilingual support

Mistral Large

Context Window: 32,768 tokens (~24,000 words)

Parameters: Proprietary large model

License: Commercial API access

Best For: Enterprise applications, production systems

Deployment: Cloud-based API service

Performance: Competitive with GPT-4 class models

🇪🇺 European Privacy & Compliance

- GDPR-compliant by design
- European data residency options
- Transparent training data policies
- Strong privacy protections

🔓 Open-Source Advantages

- Full model weights available (7B)
- Custom fine-tuning capabilities
- No vendor lock-in
- Community-driven improvements

🌍 Multilingual Excellence

- Native support for European languages
- Strong performance in French, German, Spanish, Italian
- Cultural context understanding
- Localized business applications

📄 Document Processing

- Legal contracts and compliance documents
- Financial reports and analysis
- Technical manuals and specifications
- Academic papers and research

💻 Code & Development

- Medium-sized codebases analysis
- API documentation processing
- Code review and optimization
- Technical documentation generation

🎯 **Mistral Strategy:** Perfect for organizations prioritizing data sovereignty, open-source flexibility, and European regulatory compliance while maintaining competitive performance.

LLaMA Family Context Windows

Meta's Research-Driven Open Models

LLaMA 2 (7B, 13B, 70B)

- **Context Window:** 4,096 tokens (~3,000 words)
- **License:** Custom license (commercial use allowed)
- **Strengths:** Strong foundational capabilities, efficient training
- **Best For:** Research, fine-tuning experiments, educational use
- **Community:** Large open-source ecosystem
- **Deployment:** Requires local hosting or cloud deployment

Code Llama (7B, 13B, 34B)

- **Context Window:** 16,384 tokens (~12,000 words)
- **Specialization:** Code generation, completion, and debugging
- **Languages:** Python, C++, Java, PHP, TypeScript, C#, Bash
- **Best For:** Software development, code analysis, programming education
- **Training:** Specialized on 500B tokens of code
- **Performance:** Competitive with GitHub Copilot

🔧 Research Innovations

- **RoPE (Rotary Position Embedding):** Enables context extension
- **YaRN:** Yet another RoPE extensioN technique
- **LongLLaMA:** Community extensions to 256K+ tokens
- **Code-specific optimizations:** Tailored for programming contexts

🔍 Fine-tuning Capabilities

- **LoRA (Low-Rank Adaptation):** Efficient parameter updates
- **QLoRA:** Quantized fine-tuning for consumer hardware
- **Instruction tuning:** Alpaca, Vicuna, and other derivatives
- **Domain specialization:** Medical, legal, scientific variants

4K Context (LLaMA 2) Applications

- Research experiments & model comparison
- Fine-tuning base models for specific tasks
- Educational projects and learning
- Efficient inference for simple tasks

16K Context (Code Llama) Applications

- Complete function and class analysis
- Multi-file code understanding & generation
- Documentation and docstring creation
- Code refactoring and optimization

🔬 Research & Community Focus

LLaMA models serve as a foundation for hundreds of research projects, fine-tuned variants, and community experiments, encouraging efficient prompt design and specialized fine-tuning approaches.

LLM Context Windows: Complete Comparison

Choosing the Right Model for Your Use Case

MODEL FAMILY	MODEL NAME	CONTEXT WINDOW	~WORD COUNT	COST TIER	BEST USE CASES
GPT (OpenAI)	GPT-3.5-Turbo	4K tokens	~3,000 words	💰 Low	Quick queries, chatbots
	GPT-3.5-16K	16K tokens	~12,000 words	💰💰 Medium	Document analysis
	GPT-4	8K tokens	~6,000 words	💰💰💰 High	Complex reasoning
	GPT-4-32K	32K tokens	~24,000 words	💰💰💰💰 Very High	Large documents
	GPT-4-Turbo-128K	128K tokens	~96,000 words	💰💰💰 High+	Massive analysis
Claude (Anthropic)	Claude 3 Haiku	200K tokens	~150,000 words	💰💰 Medium	Fast long-context
	Claude 3 Sonnet	200K tokens	~150,000 words	💰💰💰 High	Balanced performance
	Claude 3 Opus	200K tokens	~150,000 words	💰💰💰💰 Very High	Premium analysis
Mistral	Mistral 7B	32K tokens	~24,000 words	💰 Low (Open)	Research, fine-tuning
	Mistral Large	32K tokens	~24,000 words	💰💰💰 High	Enterprise apps
LLaMA (Meta)	LLaMA 2	4K tokens	~3,000 words	Free (Self-host)	Research, education
	Code Llama	16K tokens	~12,000 words	Free (Self-host)	Code development

Key Insights

- 🏆 Context Champions: Claude models lead with consistent 200K tokens across all variants.
- 💰 Cost Efficiency: Budget: GPT-3.5, Mistral 7B. Premium: Claude Opus, GPT-4-32K. Value: Claude Haiku, GPT-4-Turbo.
- 🎯 Use Cases: Short (≤4K): GPT-3.5, LLaMA 2. Medium (16-32K): GPT-4, Code Llama, Mistral. Long (100K+): Claude family, GPT-4-Turbo.

🧪 Quick Selection Guide

1. **Budget priority?** → GPT-3.5 or Mistral 7B
2. **Long documents?** → Claude Haiku/Sonnet
3. **Highest quality?** → Claude Opus or GPT-4-32K
4. **Code focus?** → Code Llama or Claude
5. **Research/Learning?** → LLaMA 2 (free)

Chat Completion API Fundamentals

Understanding the Foundation of LLM Interactions

REQUEST → LLM SERVICE → RESPONSE

What Are Chat Completion APIs?

- HTTP-based interfaces for interacting with LLMs.
- Most LLMs follow OpenAI's API format as a de-facto standard.
- Use RESTful principles with JSON for requests and responses.
- Enable programmatic access to powerful language model capabilities.

Authentication & Headers

Requests must be authenticated using an API key sent in the headers.

```
Authorization: Bearer YOUR_API_KEY
Content-Type: application/json
```

🔑 Essential Concepts

- **Stateless:**Each API request is independent and self-contained.
- **Token Counting:**Costs are based on input + output tokens.
- **Rate Limits:**APIs have usage restrictions to ensure fair use.

Typical Request Components

```
{
  "model": "gpt-3.5-turbo",
  "messages": [
    {"role": "user", "content": "Hello!"}
  ],
  "temperature": 0.7,
  "max_tokens": 100
}
```

Typical Response Structure

```
{
  "id": "chatcmpl-abc123",
  "choices": [{
    "message": {
      "role": "assistant",
      "content": "Hello! How can I help?"
    }
  }],
  "usage": {
    "prompt_tokens": 10,
    "completion_tokens": 6,
    "total_tokens": 16
  }
}
```

Message Format & Conversation Roles

Building Effective Conversations with LLMs

🤖 System Messages

Purpose: Set context, behavior, and instructions for the AI.
Position: Typically first in the conversation.
Content: Instructions, persona, guidelines, constraints.

> *"You are a helpful programming tutor. Explain concepts clearly and provide code examples."*

👤 User Messages

Purpose: User input, questions, and requests.
Position: Throughout the conversation.
Content: Queries, prompts, follow-up questions.

> *"How do I create a Python function that calculates the factorial?"*

🤖 Assistant Messages

Purpose: AI responses and generated content.
Position: Following user messages.
Content: Answers, explanations, generated text.

> *"Here's a Python function to calculate factorial: def factorial(n): ..."*

```
[
  {
    "role": "system",
    "content": "You are a helpful coding assistant specializing in Python
  },
  {
    "role": "user",
    "content": "I need help with loops in Python."
  },
  {
    "role": "assistant",
    "content": "I'd be happy to help! Python has two main types of loops:
  },
  {
    "role": "user",
    "content": "Can you show me a for loop example?"
  }
]
```

📝 System Message Tips

- Be specific about desired behavior and output format.
- Include relevant context and constraints.
- Set the tone and expertise level.
- Keep it concise but comprehensive for clarity.

💬 Conversation Management

- Include relevant conversation history for context.
- Manage context window limits effectively.
- Use assistant messages to maintain conversational flow.
- Implement summarization for long chats.

✨ Expert Tips

- System messages are more influential than user instructions.
- Include examples in system messages for better results.
- Test different system message variations for optimal results.

API Parameters: Fine-Tuning LLM Responses

Understanding How Parameters Shape Output

Temperature (0.0 - 2.0) 🌡️

Controls randomness and creativity in the output.

0.0 0.7 1.5 2.0

Deterministic (0.0): Always produces the same output. Best for factual queries.

Balanced (0.7): Good mix of consistency and creativity. Recommended for most apps.

Creative (1.5+): High randomness, can sacrifice accuracy. Best for brainstorming.

Top-p Sampling (0.1 - 1.0) 🎯

Limits token selection to a cumulative probability mass.

0.1 0.9 1.0

Focused (0.1): Only considers the most likely tokens for the next choice.

Standard (0.9): Includes a wide range of probable tokens, allowing for diversity.

Max Diversity (1.0): All tokens are considered, regardless of probability.

Max Tokens 📈

Sets the maximum length for the generated response.

Prevents overly long or incomplete responses and helps control API costs by limiting token usage.

Cost Control: Fewer tokens mean lower costs. Essential for budget management.

Response Integrity: Ensure the limit accounts for both prompt and desired output length.

Parameter Examples

Query: "Explain artificial intelligence"

Temperature 0.0 Output:

"Artificial intelligence (AI) refers to the simulation of human intelligence in machines..."

Temperature 0.7 Output:

"AI is a fascinating field that involves creating smart machines capable of..."

Temperature 1.5 Output:

"Whoa, AI! It's like teaching computers to think and dream, creating digital minds..."

Frequency Penalty (-2.0 to 2.0)

Reduces token repetition based on frequency. Positive values discourage repetition.

Presence Penalty (-2.0 to 2.0)

Reduces repetition regardless of frequency. Encourages discussing new topics.

Stop Sequences

Custom strings that halt generation, e.g., `["\n", "---"]`. Useful for structured outputs.

🎨 Parameter Strategy

- **Start conservative:** Temp 0.7, top_p 0.9.
- **A/B test:** Try different values for your use case.
- **Task-specific tuning:** Factual (low temp) vs Creative (high temp).
- **Cost control:** Use max_tokens and stop sequences to manage expenses.

API Error Handling & Best Practices

Building Robust LLM Applications

⚠ 401 Unauthorized

Cause: Invalid or missing API key

Solution: Check API key format and permissions

Prevent: Secure key storage, regular key rotation

⚠ 429 Too Many Requests

Cause: Rate limit exceeded

Solution: Implement exponential backoff

Prevent: Track request rates, use queuing

⚠ 400 Bad Request

Cause: Invalid parameters or format

Solution: Validate inputs before sending

Prevent: Input validation, parameter checking

⚠ 500 Internal Server Error

Cause: Provider-side issues

Solution: Retry with exponential backoff

Prevent: Implement robust retry logic

Understanding Limits

- **RPM:** Requests per minute (e.g., 60 RPM)
- **TPM:** Tokens per minute (e.g., 90,000 TPM)
- **Daily quotas:** Total usage per day
- **Concurrent requests:** Simultaneous request limits

Mitigation Strategies

- **Request queuing:** Queue requests to stay under limits
- **Token estimation:** Count tokens before API calls
- **Batch processing:** Combine multiple requests when possible
- **Graceful degradation:** Fallback options for limit hits

Best Practices Code Example

```
import time
import random

def call_llm_with_retry(prompt, max_retries=3):
    for attempt in range(max_retries):
        try:
            response = client.chat.completions.create(
                model="gpt-3.5-turbo",
                messages=[{"role": "user", "content":
prompt}],
                timeout=30
            )
            return response
        except RateLimitError:
            if attempt < max_retries - 1:
                wait_time = (2 ** attempt) + random.uniform(0,
1)
                time.sleep(wait_time)
            else:
                raise
        except APIError as e:
            print(f"API Error: {e}")
            return None
```

Production Readiness Checklist

- ✓ **API Key Security:** Environment variables, secure storage
- ✓ **Error Handling:** Comprehensive try-catch blocks
- ✓ **Retry Logic:** Exponential backoff with jitter
- ✓ **Rate Limiting:** Request queuing and throttling
- ✓ **Monitoring:** Log errors and track usage
- ✓ **Fallback Options:** Alternative responses for failures
- ✓ **Timeout Handling:** Set appropriate timeout values
- ✓ **Input Validation:** Sanitize and validate all inputs

LiteLLM Installation & Setup

Getting Started with Universal LLM Integration

Universal Interface: Single API for 100+ LLM providers

OpenAI Compatible: Drop-in replacement for OpenAI SDK

Cost Tracking: Built-in usage and cost monitoring

Error Handling: Automatic retries and fallbacks

Step 1 — Step 2 — Step 3 — Step 4

Step 1: Installation

```
# Install LiteLLM using pip
$ pip install litellm

# Or with additional proxy dependencies
$ pip install 'litellm[proxy]'

# Verify installation
$ python -c "import litellm; print(litellm.__
version__)"
```

Step 2: API Key Configuration

Option A: Environment Variables (Recommended)

```
# For OpenAI
export OPENAI_API_KEY="your-openai-key"

# For Anthropic
export ANTHROPIC_API_KEY="your-anthropic-key"
```

Option B: Python Configuration

```
import litellm
import os

# Set in code (not for production)
litellm.api_key = "your-openai-key"

# Better: Read from environment
litellm.api_key = os.getenv("OPENAI_API_KEY")
```

Step 3: Basic Import & Setup

```
# Import LiteLLM
from litellm import completion

# Basic completion call
response = completion(
    model="gpt-3.5-turbo",
    messages=[{
        "role": "user",
        "content": "Hello!"
    }]
)

print(response.choices[0].message.content)
```

Step 4: Quick Verification

```
# 🚀 Quick test script
from litellm import completion

try:
    response = completion(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content":
"Say hello!"}],
        max_tokens=10
    )
    print("✅ Setup successful!")
    print(f"Response: {response.choices[0].me
ssage.content}")
except Exception as e:
    print(f"❌ Setup error: {e}")
```

LiteLLM Basic Completion Example

Your First LLM API Call with LiteLLM

Complete Working Example

```python
# basic_completion.py
from litellm import completion
import json

def basic_llm_call():
    try:
        # Make a simple completion request
        response = completion(
            model="gpt-3.5-turbo",
            messages=[{
                "role": "system",
                "content": "You are a helpful assistant."
            }, {
                "role": "user",
                "content": "Explain machine learning in one sentence."
            }],
            temperature=0.7,
            max_tokens=100
        )
        return response
    except Exception as e:
        print(f"Error: {e}")
        return None
```

Expected Output

```
AI Response:
Machine learning is a subset of artificial
intelligence where computers learn to make
predictions or decisions by finding patterns in
data without being explicitly programmed for each
specific task.

Token Usage:
Prompt tokens: 32
Completion tokens: 28
Total tokens: 60
```

Error Handling Demonstration

```python
# Robust error handling
def safe_llm_call(model="gpt-3.5-turbo"):
    try:
        response = completion(
            model=model,
            messages=[...],
            timeout=30
        )
        return response.choices[0].message.content
    except litellm.RateLimitError:
        return "Rate limit exceeded."
    except litellm.AuthenticationError:
        return "Authentication failed."
    except Exception as e:
        return f"An error occurred: {str(e)}"
```

Key Insights & Tips

- **Response Structure**: Access content via `response.choices[0].message.content`
- **Token Counting**: Monitor usage with `response.usage` for cost tracking.
- **Error Handling**: Always wrap API calls in try-catch blocks for robustness.
- **Timeout**: Set reasonable timeouts to prevent hanging requests.

Text Summarization with LiteLLM

Complete Example with Real Input and Output

Sample Input Text

Sample Article: "Recent advances in artificial intelligence have transformed multiple industries. Machine learning algorithms can now process vast amounts of data to identify patterns and make predictions with unprecedented accuracy. In healthcare, AI systems assist doctors in diagnosing diseases by analyzing medical images and patient data. The financial sector uses AI for fraud detection, algorithmic trading, and risk assessment. Meanwhile, autonomous vehicles rely on computer vision and neural networks to navigate complex traffic scenarios. Despite these achievements, challenges remain in AI ethics, data privacy, and ensuring algorithmic fairness across diverse populations. As AI continues to evolve, researchers emphasize the importance of responsible development and deployment of these powerful technologies."

Actual Output Results

Brief Summary:
AI has revolutionized multiple industries through advanced machine learning algorithms that analyze data and make accurate predictions. While AI shows great promise in healthcare, finance, and autonomous vehicles, challenges remain in ethics, privacy, and algorithmic fairness.

Medium Summary:
Artificial intelligence has significantly transformed various industries by using machine learning algorithms to process large datasets and make highly accurate predictions. AI applications now span healthcare for medical diagnosis, finance for fraud detection and trading, and transportation through autonomous vehicles. However, the field faces ongoing challenges related to AI ethics, data privacy protection, and ensuring fair algorithmic outcomes across different population groups.

Token Usage: 89 prompt tokens, 67 completion tokens, 156 total tokens

Complete Code Implementation

```
# summarization_example.py
from litellm import completion

def summarize_text(text, length="brief"):
    # Define length instructions
    length_instructions = {
        "brief": "in 1-2 sentences",
        "medium": "in 3-4 sentences",
        "detailed": "in 5-6 sentences with key points"
    }

    instruction = length_instructions.get(length, "in 2-3 sentenc
es")

    try:
        response = completion(
            model="gpt-3.5-turbo",
            messages=[
                {
                    "role": "system",
                    "content": f"You are an expert summarizer. Pr
ovide clear, concise summaries that capture the main points. Summ
arize the given text {instruction}."
                },
                {
                    "role": "user",
                    "content": f"Please summarize this text:\n\n
{text}"
                }
            ],
            temperature=0.3,  # Lower temperature for consistent
summaries
            max_tokens=150
        )

        return response.choices[0].message.content.strip()

    except Exception as e:
        return f"Summarization failed: {str(e)}"

# Sample article text (shortened for example)
article = """Recent advances in artificial intelligence have tran
sformed multiple industries..."""

# Test different summary lengths
brief_summary = summarize_text(article, "brief")
medium_summary = summarize_text(article, "medium")

print("Brief Summary:")
print(brief_summary)
print("\nMedium Summary:")
print(medium_summary)
```

Pro Tips

- **Temperature 0.3:** Lower temp produces more focused and consistent summaries.
- **System Prompts:** Clearly define the AI's role, style, and length requirements for better control.
- **Length Control:** Use 'max_tokens' as a hard limit to manage output length and cost.
- **Batch Processing:** For multiple documents, loop through them to call the function efficiently.

Multi-Language Translation with LiteLLM

Professional Translation Examples with Quality Control

```python
# translation_example.py

from litellm import completion

def translate_text(text, target_language, source_language="auto"):
    # Create context-aware translation prompt
    prompt = f"""Translate from {source_language} to {target_languag
e}.
Maintain original tone and meaning.
Text: {text}"""

    response = completion(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a professional tr
anslator."},
            {"role": "user", "content": prompt}
        ],
        temperature=0.2, # Low temperature for consistency
        max_tokens=200
    )
    return response.choices[0].message.content.strip()
```

Real Translation Results

CASUAL TRANSLATION:
Original (English): Hello, how are you today? I hope you're having a wonderful day!
Spanish: ¡Hola, ¿cómo estás hoy? ¡Espero que tengas un día maravilloso!
French: Bonjour, comment allez-vous aujourd'hui ? J'espère que vous passez une merveilleuse journée !

BUSINESS TRANSLATION:
Original (English): Please confirm your attendance at tomorrow's meeting by 3 PM.
Spanish: Por favor, confirme su asistencia a la reunión de mañana antes de las 3 PM.
French: Veuillez confirmer votre présence à la réunion de demain avant 15h.

TECHNICAL TRANSLATION:
Original (English): The API endpoint returns a JSON response with user data and authentication tokens.
Spanish: El endpoint de la API devuelve una respuesta JSON con datos de usuario y tokens de autenticación.
French: Le point de terminaison API renvoie une réponse JSON avec des données utilisateur et des jetons d'authentification.

```python
# Advanced Translation Features

# Enhanced translation with quality control
def professional_translate(text, target_lang):
    base_prompt = f"""Translate to {target_lang}. Requirements:
    1. Maintain original meaning and tone
    2. Use natural, native-speaker phrasing
    3. Consider cultural context

    Text: {text}"""

    response = completion(
        model="gpt-4", # Use a better model for quality
        messages=[
            {"role": "system", "content": "You are a certified profe
ssional translator."},
            {"role": "user", "content": base_prompt}
        ],
        temperature=0.1
    )
    return response.choices[0].message.content
```

🌍 Translation Excellence

- **Low temperature:** Use 0.1-0.3 for consistent, high-fidelity translations.
- **Context matters:** Include domain (business, technical, casual) in prompts for better accuracy.
- **Quality models:** Use advanced models like GPT-4 or Claude for superior translation quality.
- **Cultural awareness:** Prompt the model to consider cultural nuances, not just literal translation.

Text Rewriting & Style Transformation

Adapt Content Tone and Style with LiteLLM

Flexible Rewriting System

```python
# text_rewriting_example.py
from litellm import completion

def rewrite_text(original_text, style):
    style_prompts = {
        "professional": "formal, business-appropriat
e ... ",
        "casual": "friendly, conversational ton
e ... ",
        "academic": "scholarly tone, precise termino
logy ... ",
        "marketing": "persuasive, engaging languag
e ... ",
        "technical": "clear, precise technical writi
ng ... ",
        "creative": "imaginative, expressive languag
e ... "
    }

    style_instruction = style_prompts.get(style, "cl
ear and appropriate")

    try:
        response = completion(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "system", "content": f"You
are an expert ... Rewrite using {style_instructio
n}"},
                {"role": "user", "content": f"Rewrit
e: {original_text}"}
            ],
            temperature=0.7, max_tokens=200
        )
        return response.choices[0].message.content.s
trip()
    except Exception as e:
        return f"Rewriting failed: {str(e)}"
```

Actual Rewriting Results

ORIGINAL TEXT

Our new software helps companies manage their data better. It's fast and easy to use. Many customers like it.

PROFESSIONAL STYLE

Our innovative software solution enables organizations to optimize their data management processes with enhanced efficiency. The platform delivers superior performance while maintaining exceptional user accessibility, earning widespread approval from our clientele.

CASUAL STYLE

We've got this awesome new software that makes handling your company's data a breeze! It's super quick and really user-friendly - our customers absolutely love it!

ACADEMIC STYLE

The newly developed software application facilitates enhanced data management capabilities for corporate entities. The system demonstrates high-performance characteristics and intuitive usability, resulting in positive user satisfaction metrics.

MARKETING STYLE

Revolutionize your data management with our game-changing software! Experience lightning-fast performance and effortless usability that your team will love. Join thousands of satisfied customers!

Advanced Rewriting Features

```python
def advanced_rewrite(text, requirements):
    prompt = f"""Rewrite the text according to these
requirements:
    {requirements}
    Original text: {text}"""

    response = completion(
        model="gpt-4",
        messages=[
            {"role": "system", "content": "You are a
professional content editor ... "},
            {"role": "user", "content": prompt}
        ],
        temperature=0.5
    )
    return response.choices[0].message.content
```

Practical Applications

✍️ Email Adaptation: Transform formal emails into casual replies or vice versa to match recipient tone.

📈 Marketing Copy: Convert dry technical features into persuasive, benefit-driven promotional content.

📄 Content Simplification: Adapt complex academic or technical papers for broader, non-expert audiences.

📱 Social Media Posts: Condense long-form articles or reports into engaging, platform-specific updates.

> ***Customized Rewrite:*** *Simplify your data management with our fast, user-friendly software. Boost your small business's efficiency today. Try it now!*

📘 User Documentation: Rewrite technical developer docs into clear, user-friendly guides for end-users.

Parameter Tuning: Real Examples & Effects

See How Parameters Shape LLM Responses

Experiment 1: Temperature Effects

```python
# temperature_effects.py
from litellm import completion

def test_temperature(prompt, temperatures=[0.0, 0.5, 1.0,
    results = {}

    for temp in temperatures:
        response = completion(
            model="gpt-3.5-turbo",
            messages=[{"role": "user", "content": prompt}]
            temperature=temp,
            max_tokens=50
        )
        results[temp] = response.choices[0].message.conten

    return results

# Test with creative writing prompt
test_prompt = "Write the opening line of a short story abo
results = test_temperature(test_prompt)
```

Results: From Deterministic to Creative

Deterministic (temp=0.0)

"Sarah had always walked past the old wooden door at the end of the hallway, but today something made her stop and stare."

Balanced (temp=0.5)

"The brass handle of the peculiar door seemed to glow faintly in the dim corridor light, beckoning Emma closer."

Creative (temp=1.0)

"Nobody remembered when the shimmering door appeared in Mrs. Chen's basement, or why it hummed softly at midnight."

Highly Creative (temp=1.5)

"Seventeen purple butterflies danced around the singing doorframe that definitely hadn't existed yesterday morning."

Experiment 2: Top-p Sampling Effects

```python
# top_p_effects.py
def test_top_p(prompt, top_p_values=[0.1, 0.9, 1.0]):
    results = {}
    for top_p in top_p_values:
        response = completion(
            model="gpt-3.5-turbo",
            messages=[{"role": "user", "content": prompt}],
            temperature=0.8, # Fixed temperature
            top_p=top_p, max_tokens=40
        )
        results[top_p] = response.choices[0].message.content
    return results
# Test with explanation prompt
explain_prompt = "Explain quantum computing in simple terms."
top_p_results = test_top_p(explain_prompt)
```

Focused (top_p=0.1)

"Quantum computing uses quantum bits that can exist in multiple states simultaneously, allowing for incredibly fast calculations."

Balanced (top_p=0.9)

"Think of quantum computing like having a magical computer that can explore many different solutions at once, making it super

🎯 Parameter Combination Strategy

- 📚 **Factual content:** Low temp (0.1-0.3) + top_p (0.8-0.9)
- ✍️ **Creative writing:** Med-high temp (0.7-1.2) + top_p (0.9-1.0)
- 📄 **Technical docs:** Very low temp (0.0-0.2) + top_p (0.7-0.9)
- 💡 **Brainstorming:** High temp (1.0-1.5) + top_p (1.0)

powerful."

Full Diversity (top_p=1.0)

"Quantum computing harnesses weird physics phenomena where particles dance between possibilities, creating computational superpowers!"

⚙️ **Consistent output:** temp=0.0 (deterministic mode)

LLM Capabilities & Current Limitation

Understanding What LLMs Can and Cannot Do

What LLMs Excel At

🏆 Text Generation & Writing

- Creative writing, technical documentation, email composition, content creation
- Style adaptation and rewriting, multiple language support

📊 Data Analysis & Summarization

- Document summarization and analysis, information extraction from text
- Pattern recognition in content, research assistance and synthesis

💻 Code Generation & Programming

- Code writing and completion, bug detection and debugging
- Code explanation and documentation, multiple programming language support

🧠 Reasoning & Problem Solving

- Step-by-step problem breakdown, logical reasoning and analysis
- Decision support and recommendations

🌍 Translation & Language Tasks

- High-quality translation between languages, grammar correction
- Language learning assistance, cultural context understanding

Current Limitations

⚠️ Knowledge & Information

- Training data cutoff (knowledge gaps), cannot access real-time information
- May generate outdated facts, no internet browsing capability

🌀 Hallucinations & Accuracy

- May generate convincing but false information, inconsistent factual accuracy
- Cannot verify information independently, overconfident in incorrect responses

🔪 Context Window Constraints

- Limited memory of long conversations, cannot process extremely long documents
- Forgets earlier context when limit exceeded, token counting affects performance

🔢 Mathematical & Computational

- Complex arithmetic calculation errors, inconsistent with multi-step math
- Cannot execute code or verify results, struggles with precise numerical tasks

👁️ Sensory & Multimodal

- Cannot see images or process visual data, no audio processing capabilities
- Limited to text-based interactions, cannot access files or external systems

Context Window Impact on Tasks

Small Context (4K tokens)

✅ **Good for:** Quick Q&A, short summaries, simple code

❌ **Limited:** Long document analysis, extended conversations

Medium Context (32K tokens)

✅ **Good for:** Document analysis, code review, research papers

❌ **Limited:** Book-length analysis, multi-document tasks

Large Context (200K+)

✅ **Good for:** Comprehensive analysis, entire codebases

❌ **Limited:** Still cannot access external data or real-time info

🛡️ Mitigation Strategies

Fact-checking: Verify important information independently.

Prompt engineering: Use clear, specific instructions.

Context management: Break large tasks into smaller chunks.

Human oversight: Always review critical outputs.

Zero-Shot Prompting: Direct Task Instructions

Asking the LLM to perform a task without providing any examples

Key Principles

- ✔ Clear, specific instructions
- ✔ Define the desired output format
- ✔ Provide necessary context
- ✔ Be explicit about requirements

Best Practices

- ✔ Be specific about desired output format
- ✔ Include relevant context and constraints
- ✔ Use clear, unambiguous language
- ✔ Test and iterate on prompts

Practical Examples

Example 1: Text Classification

```
Classify the following email as spam or not spam: [email content]
```

Example 2: Code Generation

```
Write a Python function that calculates the factorial of a number using recursion.
```

Example 3: Summarization

```
Summarize this article in 3 bullet points focusing on key findings: [article]
```

Few-Shot Prompting: Learning from Examples

Providing the LLM with a few examples of the desired input-output pattern to guide its response

Definition

Providing the LLM with a few examples of the desired input-output pattern to guide its response.

Why It Works

- Helps the model understand the specific task format
- Shows desired output style and structure
- Reduces ambiguity in task interpretation
- Enables in-context learning

Few-Shot Example: Sentiment Analysis

```
Classify the sentiment of these reviews:

Review: 'This product is amazing!' → Positive
Review: 'Terrible quality, waste of money.' → Negative
Review: 'It's okay, nothing special.' → Neutral

Review: 'Love the design and functionality!' → ?
```

Best Practices

- Use diverse, representative examples
- Keep examples concise but complete
- Show edge cases when relevant
- Maintain consistent formatting across examples

Pro Tip: "2-5 examples are usually sufficient for most tasks"

Chain-of-Thought Reasoning

Step-by-Step Problem Solving

Why Effective

- ✓ Improves reasoning accuracy
- ✓ Makes process transparent
- ✓ Catches logical errors
- ✓ Better problem decomposition

Best For

- ✓ Math problems
- ✓ Logic puzzles
- ✓ Multi-step analysis

Without CoT

```
"25 * 14?" → "350" (no reasoning shown)
```

With CoT

```
"25 * 14? Think step by step."
Step 1: 25 * (10 + 4)

Step 2: (25*10) + (25*4)
Step 3: 250 + 100 = 350
```

Advanced Prompting Techniques

Role Prompting

Assign specific roles to focus responses and frame the AI's perspective.

```
"You are a senior DevOps engineer. Review
this CI/CD pipeline configuration for
potential bottlenecks..."
```

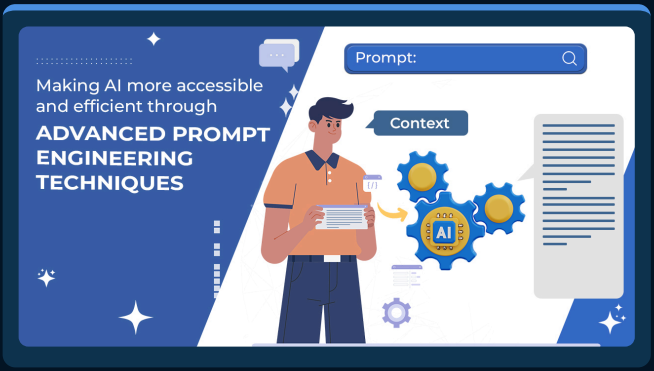
System Messages

Set high-level instructions for the model's behavior that persist across conversational turns.

- Persist across multiple turns
- Set expertise level & persona
- Define output structure or format

Best Practices

- **Be Specific:** Use detailed context and instructions to avoid ambiguity.
- **Use Examples:** Few-shot prompting guides the model's output style.
- **Structure:** Use clear sections, headings, or delimiters like XML tags.
- **Iterate:** Refine prompts based on outputs to improve performance.
- **Chain Prompts:** Break down complex tasks into a sequence of simpler prompts.



Hands-On Lab Session Overview

2 Hours of Practical LLM Programming Experience

🚀 Part 1: Environment Setup (20 minutes)

0:00-0:10: Python environment verification
0:10-0:15: LiteLLM installation and API key setup
0:15-0:20: First successful API call test

Goal: *Everyone ready to code with LLMs*

💻 Part 2: Basic API Mastery (30 minutes)

0:20-0:35: Simple completion exercises with different models
0:35-0:45: Error handling and debugging practice
0:45-0:50: Parameter exploration (temperature, max_tokens)

Goal: *Confident with basic LLM API calls*

🎨 Part 3: Prompt Engineering Workshop (40 minutes)

0:50-1:05: Zero-shot prompting challenges
1:05-1:20: Few-shot learning implementations
1:20-1:30: Chain-of-thought & Role prompting

Goal: *Master advanced prompting techniques*

📈 Part 4: Advanced Applications (30 minutes)

1:30-1:55: Build advanced application workflows

Goal: *Build production-ready LLM applications*

What You'll Build Today

📄 Text Processor

- Summarization tool
- Multi-language translator
- Style rewriter
- Sentiment classifier

🤖 Intelligent Assistant

- Context-aware chatbot
- Code explanation tool
- Research assistant
- Content creation workflow

Learning Outcomes

- ✓ Install and configure LiteLLM
- ✓ Write robust error handling
- ✓ Master key generation parameters
- ✓ Implement advanced prompting

📋 Pre-Lab Checklist

- ✓ Python 3.8+ installed
- ✓ Text editor or IDE ready
- ✓ Internet connection
- ✓ Enthusiasm for learning!

Bonus: Students who complete all exercises early will tackle advanced challenges like multi-model comparison and custom prompt optimization.

Detailed Lab Structure & Student Expectations

Your Roadmap to LLM Programming Success

Part 1: Env. Setup (20 min)

Foundational

Objectives:

- Verify Python & env
- Install LiteLLM package
- Configure API keys securely
- Execute 1st successful API call

Key Activities:

- Step-by-step installation
- API key config & security
- "Hello World" API test
- Resolve common issues

Deliverable: Screenshot of API response

Part 2: API Mastery (30 min)

Core Skills

Objectives:

- Execute requests with diff. models
- Implement error handling
- Explore key parameters
- Compare provider outputs

Key Activities:

- Model comparison exercises
- Parameter experiments
- Error handling simulation
- Token usage & cost tracking

Deliverable: Python script with error handling

Part 3: Prompting (40 min)

Advanced Techniques

Objectives:

- Master zero-shot prompting
- Implement few-shot learning
- Practice chain-of-thought
- Design effective system messages

Key Activities:

- Prompt optimization challenges
- Task-specific prompting
- A/B test prompt effectiveness
- Creative applications

Deliverable: Portfolio of optimized prompts

Part 4: Applications (30 min)

Production Ready

Objectives:

- Build robust applications
- Manage context windows
- Apply cost optimization
- Create user-friendly interfaces

Key Activities:

- Mini-project development
- Performance optimization
- User experience design
- Deployment preparation

Deliverable: Complete application with docs

🏆 Success Metrics

Participation: Active engagement in exercises

Completion: Finish all four lab parts

Quality: Working, well-structured code

Creativity: Innovative use of techniques

🤝 Collaboration Policy

Encouraged: Help peers with tech issues

Share Insights: Discuss prompt strategies

Peer Learning: Form groups for challenges

Individual Work: Submit your own deliverables

🔄 Continuous Support

Instructor: Assistance throughout session

TA Support: For troubleshooting & guidance

Resources: Curated docs and tutorials

Office Hours: Post-lab extended help

🎇 Tips for Success

- Start early on setup issues
- Experiment freely with models
- Document your findings
- Ask lots of questions
- Have fun exploring LLMs!

Python Environment Verification

Step 1: Confirm Your Development Environment

✅ Verification Checklist

- [x] Check Python version: `python --version`
- [ ] Get system details: `python -c "import sys; print(sys.version)"`
- [ ] Verify executable path: `which python` (Unix) / `where python` (Windows)

💡 Expected Output Examples

```
$ python --version

Python 3.10.4
```

```
$ where python

C:\Users\YourUser\AppData\Local\Programs\Python\Python310\python.exe
```

🔧 Troubleshooting

- 'python' not found? Check your system's PATH variable.
- Wrong version? Ensure correct Python installation is first in PATH.
- Using an environment manager (like conda or venv)? Make sure it's activated.

⏱️ Est. Time: 5-10 minutes

LiteLLM Installation Step-by-Step

Step 2: Installing the Universal LLM Interface

1. Installation Commands

Primary Method (Recommended):

```
pip install litellm
```
COPY

Alternative for multiple Python versions:

```
pip3 install litellm
```
COPY

Using a Virtual Environment (Best Practice):

```
python -m venv llm_env
source llm_env/bin/activate
pip install litellm
```
COPY

2. Verification

✔

```
import litellm; print("LiteLLM installed successfully")
```
COPY

```
Expected Output: LiteLLM installed successfully
```

✔

```
litellm --version
```
COPY

Troubleshooting

- ▲ **Permission Denied:** If you see a permission error, try installing as a user. `pip install litellm --user`
- ▲ **Command Not Found:** If `pip` or `pip3` is not found, ensure Python and Pip are in your system's PATH. You may need to install or upgrade them first.
- ▲ **Version Conflicts:** Dependency issues? A clean virtual environment is the best solution to avoid conflicts with other packages.

⏱ 10-15 minutes | Lab Step 2/17

API Key Setup & Configuration

Step 3: Secure API Authentication Setup

🔑 Obtaining OpenAI API Key

- → Sign up or log in at `platform.openai.com`.
- → Navigate to your dashboard profile menu: "View API keys".
- → Click "+ Create new secret key".
- → Copy the key immediately. You won't be able to see it again for security reasons.

⚙️ Configuration

Set the key as an environment variable to keep it secure.

```
# Windows (Command Prompt)
set OPENAI_API_KEY=your_key_here

# macOS / Linux (Bash/Zsh)
export OPENAI_API_KEY=your_key_here
```

Or use a `.env` file with Python for project-specific keys:

```
# 1. In your project, create a .env file:
OPENAI_API_KEY="your_key_here"

# 2. In your Python script:
# pip install python-dotenv
import os
from dotenv import load_dotenv

load_dotenv()
api_key = os.getenv('OPENAI_API_KEY')
```

⏱️ 10-15 minutes | Lab Step 3/17 | 🔒 Security Critical

First Successful Connection Test

Step 4: Verify Everything Works Together

Python: Test Script

```python
import litellm

response = litellm.completion(
  model="gpt-3.5-turbo",
  messages=[{"role": "user", "content": "Hello! Can you confirm the connection is working?"}]
)
print(response.choices[0].message.content)
```

✓ **Connection Status: SUCCESS**

💬 *"Yes, the connection is working perfectly! I am ready to assist you."*

Pre-flight Checklist

- ✅ Python environment working
- ✅ LiteLLM installed correctly
- ✅ API key configured
- ✅ Internet connection active
- ✅ No firewall blocking

🎉 **Congratulations! You're Ready for LLM Development** 🎉

Simple Text Completion - Template

Part 2: Basic API Operations - Interactive Practice

```python
import litellm

# TODO: Fill in your prompt here
user_prompt = "                                "

response = litellm.completion(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "user", "content": user_prompt}
    ],
    max_tokens=       ,  # TODO: Choose token limit
    temperature=      # TODO: Set creativity level
)

print(response.choices[0].message.content)
```

📝 Suggested Prompts to Try:

- ✍️ Write a short poem about artificial intelligence
- ✍️ Explain quantum computing in simple terms
- ✍️ Create a recipe for chocolate chip cookies
- ✍️ Draft a professional email apologizing for a delay

Parameter Guidance

`max_tokens` : 50-200 for short responses.

`temperature` : 0.3-0.7 for balanced creativity.

Simple Text Completion - Practice

Working Examples with Expected Outputs

Example 1: Creative Writing

```
response = litellm.completion(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": "Write a haiku about programming"}],
    max_tokens=60,
    temperature=0.7
)
```

Expected Output:

> *"Code flows like rivers / Logic builds tomorrow's dreams / Debug brings us peace"*

Example 2: Technical Explanation

```
response = litellm.completion(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": "Explain APIs in one sentence"}],
    max_tokens=50,
    temperature=0.3
)
```

Parameter Analysis

- `max_tokens` controls the maximum length of the generated response. A smaller value results in a shorter text.
- `temperature` influences randomness. Higher values (e.g. `0.8`) lead to more creative, less predictable text. Lower values (e.g. `0.2`) result in more focused, consistent output.
- `Model Selection` (e.g. `gpt-3.5-turbo` vs. `gpt-4`) significantly impacts the nuance, accuracy, and quality of the response.

🎯 Try These Exercises

- Write a tweet announcing a new AI product.
- Generate a product description for a new smartwatch.
- Summarize the concept of machine learning in two sentences.

Summarization Task - Step by Step

Advanced Text Processing with LLMs

Sample Article

Recent advancements in artificial intelligence, particularly in the realm of large language models (LLMs), have revolutionized how we interact with information. Models like GPT-4 and Claude 3 can now understand and generate human-like text with unprecedented accuracy, enabling applications from sophisticated chatbots to automated content creation.

These models are trained on vast datasets, allowing them to grasp complex nuances, contexts, and even creative styles. The underlying architecture, often based on the Transformer model, utilizes attention mechanisms to weigh the importance of different words in the input text, leading to more coherent and relevant outputs.

As these technologies continue to evolve, they are being integrated into various industries, including software development, healthcare, and finance, to automate tasks, analyze data, and drive innovation. However, this rapid progress also brings challenges, such as ethical considerations, the potential for misuse, and the need for robust evaluation metrics to ensure reliability and prevent the spread of misinformation.

Summarization Code

```
long_text = """***[Sample article text from the left pan
el]***"""

response = litellm.completion(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a profess
ional summarizer. Create concise, accurate summaries."},
        {"role": "user", "content": f"Summarize this art
icle in 2-3 sentences:\n\n{long_text}"}
    ],
    max_tokens=150,
    temperature=0.2
)
```

Best Practices

- ✓ Use system messages for role definition.
- ✓ Specify desired summary length clearly.
- ✓ Lower temperature for factual accuracy.

Quality Indicators

- ✓ **Concise:** Captures main points briefly.
- ✓ **Accurate:** Faithfully represents source facts.
- ✓ **Coherent:** Reads smoothly and logically.

Translation Task - Implementation

Multilingual Text Processing

```python
def translate_text(text, target_language):
    response = litellm.completion(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": f"You are a professional translator. Translate text to {target_language} while preserving meaning and tone."},
            {"role": "user", "content": f"Translate this text: {text}"}
        ],
        max_tokens=200,
        temperature=0.1  # Low for accuracy
    )
    return response.choices[0].message.content
```

Language Pair Examples

🇬🇧 → 🇪🇸 "Hello, how are you today?" → "Hola, ¿cómo estás hoy?"

🇬🇧 → 🇫🇷 "Artificial Intelligence is revolutionary" → "L'Intelligence Artificielle est révolutionnaire"

🇬🇧 → 🇩🇪 "Programming is creative problem solving" → ?

Cultural Considerations

- Context-aware translations
- Formal vs. informal registers
- Cultural nuances and idioms

Practice Exercises

- English → Japanese
- Spanish → English
- French → German
- German → Chinese

Text Rewriting - Style Variations

Adapting Content for Different Audiences

- **Original Text**

"Our company's new software update includes several improvements to user experience and performance optimization."

- **Formal**

"The latest software revision incorporates significant enhancements to the user experience, alongside substantial performance optimizations."

- **Casual**

"Hey everyone! We've just rolled out a new update that makes the app way smoother and easier to use. Check it out!"

- **Technical**

"Version 3.1.0 reduces API latency via payload compression and refactors the UI rendering engine, resulting in a 15% decrease in load times."

- **Marketing**

"Unlock a seamless experience! Our new update supercharges your workflow with lightning-fast performance and an intuitive redesign."

- **Python Code Template**

```python
import litellm

def rewrite_for_audience(text, target_style):
    # The system message guides the model's tone
    response = litellm.completion(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system",
             "content":
             f"Rewrite text in {target_style}
              style while preserving key information."},
            {"role": "user",
             "content": f"Rewrite: {text}"}
        ],
        max_tokens=150,
        temperature=0.4
    )
    return response.choices[0].message.content
```

- **Practice Challenges**

```
"The meeting is scheduled for 3 PM tomorrow."
```

Error Handling Basics

Part 2 Wrap-up: Robust LLM Applications

⚠️ Common Error Types

API Errors: Invalid API key, rate limits, quota exceeded

Connection Errors: Network issues, timeout errors

Input Errors: Invalid model names, malformed requests

Response Errors: Empty responses, JSON parsing failures

🔧 Debugging Techniques

- ☑ Enable verbose logging ('litellm.set_verbose=True')
- ☑ Check provider API status pages
- ☑ Validate input parameters before the call
- ☑ Isolate and test with minimal examples

```
try:
    response = litellm.completion(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=150
    )
    return response.choices[0].message.content
except litellm.exceptions.RateLimitError as e:
    print(f"Rate limit exceeded: {e}")
    return "Please try again later"
except litellm.exceptions.AuthenticationError as
e:
    print(f"Authentication failed: {e}")
    return "Check your API key"
except Exception as e:
    print(f"Unexpected error: {e}")
    return "Something went wrong"
```

⏱️ 15-20 minutes | Lab Step 10/17 | 🔧 Debugging Skills

Role Prompting Workshop

Part 3: Advanced Prompt Engineering - System Messages

🧐 Professional Roles

Teacher | Consultant | Analyst | Code Reviewer

🎨 Creative Roles

Storyteller | Poet | Comedian | Game Master

🔬 Technical Roles

Software Engineer | Data Scientist | Technical Writer

System Message Implementation

```
def create_role_assistant(role, expertise):
    system_message = f"""You are a professional {role} with
expertise in {expertise}.

Respond in character with appropriate:
- Tone and vocabulary
- Professional insights
- Relevant examples
- Practical advice"""

    response = litellm.completion(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": system_message},
            {"role": "user", "content": user_query}
        ]
    )
    return response.choices[0].message.content
```

Workshop Challenges

- 🎯 Create a coding mentor assistant
- 🎯 Design a creative writing coach

Consistency Tip

Maintain character throughout the entire conversation for a more immersive experience.

Chain-of-Thought Implementation

Teaching LLMs to Think Step-by-Step

Problem Example

A restaurant sold 42 pizzas on Monday, 38 on Tuesday, and 51 on Wednesday. If each pizza costs $18, what was the total revenue for these three days?

Reasoning Steps Visualization

Step 1: 42 + 38 + 51 = 131 total pizzas

Step 2: 131 × $18 = $2,358 total revenue

Chain-of-Thought Prompt

```
def solve_with_reasoning(problem):
    prompt = f"""Solve this step-by-step. Show
your reasoning:

    Problem: {problem}

    Step 1: Identify what we know
    Step 2: Determine what we need to find
    Step 3: Plan the solution approach
    Step 4: Execute calculations
    Step 5: Verify the answer

    Let's work through this:"""

    response = litellm.completion(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content":
prompt}],
        temperature=0.2
    )
    return response.choices[0].message.content
```

Practice Problems

- ✓ Logic puzzles
- ✓ Multi-step math problems
- ✓ Causal analysis tasks

Benefits

- ✓ Improved accuracy on complex tasks
- ✓ Transparent and auditable reasoning
- ✓ Better general problem-solving ability

Temperature & Parameter Tuning Lab

Fine-Tuning LLM Output Behavior

Temperature 0.1
Highly focused, consistent, deterministic output.

Temperature 0.5
Balanced creativity and coherence. Good for most tasks.

Temperature 0.9
Highly creative, unpredictable, and varied results.

INTERACTIVE EXPERIMENT CODE

```
def compare_temperatures(prompt):
    temperatures = [0.1, 0.5, 0.9]
    results = {}

    for temp in temperatures:
        response = litellm.completion(
            model="gpt-3.5-turbo",
            messages=[{"role": "user", "content": prompt}],
            temperature=temp,
            max_tokens=100
        )
        results[temp] = response.choices[0].message.content

    return results
```

EXPERIMENT PROMPT

```
"Write a short story
opening about a
robot discovering
music"
```

Advanced Techniques Combination

Part 3 Mastery: Integrating Multiple Prompt Strategies

```python
def advanced_completion(task, context, examples=None):
    # System message (Role) + Few-shot (Examples) +
Chain-of-thought
    system_msg = "You are an expert technical writer.
Think step-by-step."

    user_prompt = f"""Context: {context}

    {'Examples: ' + examples if examples else ''}

    Task: {task}

    Please:
    1. Analyze the requirements
    2. Consider the context
    3. Apply best practices
    4. Provide a detailed response

    Let me work through this step by step:"""

    return litellm.completion(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": system_msg},
            {"role": "user", "content": user_prompt}
        ],
        temperature=0.3,
        max_tokens=400
    )
```

Technique Stack

- ✓ System Messages (Role Definition)
- ✓ Few-Shot Learning (Examples)
- ✓ Chain-of-Thought (Reasoning)
- ✓ Parameter Tuning (Optimization)

Mastery Challenge

Create a technical documentation assistant using all four techniques combined in the master prompt template.

Best Practices

Combine techniques for complex tasks requiring nuance and structure. Be mindful of over-prompting, which can confuse the model or exceed token limits.

25-30 minutes | Lab Step 14/17

Advanced Mastery

Build Your Own AI Assistant

Part 4: Creative Applications - Capstone Project

```python
class PersonalAIAssistant:
    def __init__(self, name, specialty, personality):
        self.name = name
        self.specialty = specialty
        self.personality = personality

    def get_system_message(self):
        return f"""You are {self.name}, a helpful AI assistant.
        Specialties: {self.specialty}
        Personality: {self.personality}
        Always respond in character with expertise and helpfulness."""

    def chat(self, user_message):
        response = litellm.completion(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "system", "content": self.get_system_message()},
                {"role": "user", "content": user_message}
            ],
            temperature=0.6
        )
        return response.choices[0].message.content
```

🎓 Study Buddy

Academic help, homework assistance, concept explanation

💻 Code Mentor

Programming guidance, debugging help, best practices

🎨 Creative Partner

Writing assistance, brainstorming, idea generation

📅 Personal Organizer

Task management, scheduling, productivity tips

🚀 Enhancement Ideas

Add memory, conversation history, or specialized knowledge bases.

🏆 Student Challenge: Create & Demo!

Production Best Practices

Deploying LLM Applications at Scale

Security & Authentication

- **API Key Management:** Use environment variables or dedicated secret management services (e.g., AWS Secrets Manager, HashiCorp Vault).
- **Access Control:** Implement robust rate limiting, user authentication, and strict input validation to prevent abuse and injection attacks.
- **Data Privacy:** Never include PII or sensitive data in prompts. Ensure compliance with regulations like GDPR and CCPA.

Error Handling & Reliability

```python
import time
import random

def robust_completion(messages, max_retries=3):
    for attempt in range(max_retries):
        try:
            response = litellm.completion(
                model="gpt-3.5-turbo",
                messages=messages,
                timeout=30
            )
            return response.choices[0].message.content
        except Exception as e:
            if attempt == max_retries - 1:
                raise e
            time.sleep(random.uniform(1, 3))  # Exponential backoff
    return None
```

30-35 minutes | Lab Step 16/17 | Enterprise Ready

Lab Wrap-up & Next Steps

Congratulations on Your LLM Development Journey! 🎉

What You've Accomplished

- ✅ **Environment Setup:** Python, LiteLLM, API configuration
- ✅ **Basic Operations:** Text completion, summarization, translation, rewriting
- ✅ **Advanced Techniques:** Role prompting, chain-of-thought, parameter tuning
- ✅ **Production Skills:** Error handling, best practices, robust applications
- ✅ **Creative Project:** Built your own AI assistant

Skills Mastered

- Prompt engineering fundamentals
- API integration and management
- Error handling and debugging
- Production-ready development practices

Next Steps & Resources

- 📁 **Homework Assignment:** Extend your AI assistant with new features
- 📚 **Additional Learning:** Advanced prompt patterns, fine-tuning, RAG systems
- 🎨 **Project Ideas:** Chatbots, content generators, automation tools
- 🔗 **Community:** Join LLM dev communities, contribute to open source

Preview Next Week: Advanced LLM Applications: RAG, Function Calling & Multi-Agent Systems