

# **Week 3: LLM APIs & Multimodal AI**

**CS 203: Software Tools and Techniques for AI**

Prof. Nipun Batra  
IIT Gandhinagar

# **Today's Agenda (90 minutes)**

## **1. Introduction to LLM APIs (10 min)**

- What are LLM APIs?
- Major providers and models

## **2. Gemini API Setup (15 min)**

- API keys and authentication
- Python SDK setup
- Basic text generation

## **3. Text Understanding (20 min)**

- Classification, NER, QA
- Few-shot learning

# What are LLM APIs?

## Large Language Model APIs

APIs that provide access to powerful AI models:

- Generate and understand text
- Analyze images, audio, video
- Extract structured information
- Perform complex reasoning

## Why Use LLM APIs?

- No need to train models yourself
- State-of-the-art performance

# Major LLM Providers

Provider	Models	Strengths
OpenAI	GPT-4, GPT-3.5	Text, code, vision
Google	Gemini Pro, Ultra	Multimodal, long context
Anthropic	Claude 3	Long context, safety
Meta	Llama 2, 3	Open source
Mistral	Mixtral, Mistral	Efficient, multilingual

## Today's Focus: Gemini API

- Powerful multimodal capabilities
- Generous free tier

# Gemini API Setup

## Get Your API Key

1. Visit [Google AI Studio](#)
2. Create or select a project
3. Generate API key
4. Set environment variable:

```
export GEMINI_API_KEY='your-api-key-here'
```

## Install SDK

```
pip install google-genai pillow requests
```

# Initialize Gemini Client

## Basic Setup

```
import os
from google import genai

# Check for API key
if 'GEMINI_API_KEY' not in os.environ:
    raise ValueError("Set GEMINI_API_KEY environment variable")

# Initialize client
client = genai.Client(api_key=os.environ['GEMINI_API_KEY'])

# Available models
MODEL = "models/gemini-3-pro-preview"
IMAGE_MODEL = "models/gemini-3-pro-image-preview"

print("Gemini client initialized!")
```

# Your First API Call

## Simple Text Generation

```
# Create a simple prompt
response = client.models.generate_content(
    model=MODEL,
    contents="Explain what a Large Language Model is in one sentence."
)

print(response.text)
```

### Output:

A Large Language Model (LLM) is an AI system trained on massive amounts of text data to understand and generate human-like language.

That's it! You've just used an LLM API.

# Understanding the Response

## Response Structure

```
response = client.models.generate_content(  
    model=MODEL,  
    contents="What is 2 + 2?"  
)  
  
# Access different parts  
print(response.text)                      # "2 + 2 equals 4"  
print(response.usage_metadata)              # Token usage  
print(response.candidates[0].finish_reason) # Why it stopped
```

## Key Attributes

- `text` : The generated text
- `usage_metadata` : Input/output tokens

# Part 2: Text Understanding

## Common NLP Tasks

1. **Sentiment Analysis:** Positive/Negative/Neutral
2. **Named Entity Recognition:** Extract people, places, orgs
3. **Classification:** Categorize text
4. **Summarization:** Condense long text
5. **Question Answering:** Answer questions from context
6. **Translation:** Multilingual translation

**Key advantage:** No training required! Just describe the task.

# Sentiment Analysis

## Basic Example

```
text = "This product exceeded my expectations! Absolutely love it."  
  
response = client.models.generate_content(  
    model=MODEL,  
    contents=f""")  
  
Analyze the sentiment of this text.  
Respond with only: Positive, Negative, or Neutral.  
  
Text: {text}  
"""  
)  
  
print(response.text) # "Positive"
```

Pro tip: Clear, specific instructions work best.

# Few-Shot Learning

## Teach by Example

```
prompt = """  
Classify movie reviews as Positive or Negative.
```

Examples:

Review: "Amazing film! Best I've seen this year."

Sentiment: Positive

Review: "Terrible waste of time and money."

Sentiment: Negative

Now classify:

Review: "The acting was mediocre and plot predictable."

Sentiment:

"""

```
response = client.models.generate_content(model=MODEL, contents=prompt)
```

# Named Entity Recognition

## Extract Entities from Text

```
text = "Apple CEO Tim Cook announced new products in Cupertino on Monday."  
  
prompt = f"""\nExtract all named entities from this text and categorize them.  
Return as JSON with categories: Person, Organization, Location, Date.  
  
Text: {text}\n"""  
  
response = client.models.generate_content(model=MODEL, contents=prompt)  
print(response.text)
```

## Output:

```
{  
    "text": "Apple CEO Tim Cook announced new products in Cupertino on Monday.",  
    "entities": [{"entity": "Apple", "category": "Organization"}, {"entity": "Tim Cook", "category": "Person"}, {"entity": "Cupertino", "category": "Location"}, {"entity": "Monday", "category": "Date"}]
```

# Structured JSON Output

## Enforce Output Format

```
from pydantic import BaseModel
from typing import List

class Entity(BaseModel):
    text: str
    category: str

class NERResult(BaseModel):
    entities: List[Entity]

# Request structured output
response = client.models.generate_content(
    model=MODEL,
    contents="Extract entities: Alice met Bob in Paris on Friday.",
    config={
        "response_mime_type": "application/json",
        "response_schema": NERResult
    }
)
```

# Text Summarization

## Condense Long Text

```
article = """  
[Long news article about climate change...]  
"""  
  
prompt = f"""  
Summarize this article in 3 bullet points:  
  
{article}  
"""  
  
response = client.models.generate_content(model=MODEL, contents=prompt)  
print(response.text)
```

### Tips for good summaries:

- Specify desired length (words, sentences, bullets)

# Question Answering

## Extract Information from Context

```
context = """  
Python is a high-level programming language created by Guido van Rossum  
in 1991. It emphasizes code readability and allows programmers to express  
concepts in fewer lines of code.  
"""
```

```
question = "Who created Python and when?"
```

```
prompt = f"""  
Context: {context}  
Question: {question}
```

```
Answer based only on the context above.  
"""
```

```
response = client.models.generate_content(model=MODEL, content=question)
```

# Part 3: Multimodal Capabilities

## What is Multimodal AI?

**Multimodal:** Understanding multiple types of data

- Text
- Images
- Audio
- Video
- Documents (PDFs)

## Gemini's Multimodal Features

1. **Vision:** Image understanding, OCR, object detection

2. **Audio:** Speech-to-text, text-to-speech, audio analysis

# Image Understanding Basics

## Analyze an Image

```
from PIL import Image
import requests
from io import BytesIO

# Load image
url = "https://example.com/cat.jpg"
response = requests.get(url)
image = Image.open(BytesIO(response.content))

# Ask about the image
result = client.models.generate_content(
    model=IMAGE_MODEL,
    contents=[
        "Describe this image in detail.",
        image
    ]
)
```

# Visual Question Answering

## Ask Specific Questions About Images

```
# Load product image
image = Image.open("product.jpg")

questions = [
    "What color is the product?",
    "What brand is visible?",
    "Is the product damaged?",
    "What is the approximate size?"
]

for question in questions:
    result = client.models.generate_content(
        model=IMAGE_MODEL,
        contents=[question, image]
    )
    print(f"Q: {question}")
```

# Object Detection with Bounding Boxes

## Detect and Locate Objects

```
image = Image.open("street_scene.jpg")

prompt = """
Detect all objects in this image.
For each object, provide:
1. Object name
2. Bounding box coordinates [x1, y1, x2, y2] normalized to 0-1000
3. Confidence score

Return as JSON array.
"""

result = client.models.generate_content(
    model=IMAGE_MODEL,
    contents=[prompt, image]
)
```

# Drawing Bounding Boxes

## Visualize Detections

```
from PIL import ImageDraw

def draw_boxes(image, detections):
    draw = ImageDraw.Draw(image)
    width, height = image.size

    for det in detections:
        # Convert normalized coords to pixels
        x1 = int(det['bbox'][0] * width / 1000)
        y1 = int(det['bbox'][1] * height / 1000)
        x2 = int(det['bbox'][2] * width / 1000)
        y2 = int(det['bbox'][3] * height / 1000)

        # Draw box
        draw.rectangle([x1, y1, x2, y2], outline='red', width=3)
        draw.text((x1, y1-20), det['object'], fill='red')

    return image
```

# OCR and Document Understanding

## Extract Text from Images

```
# Load document image
doc_image = Image.open("receipt.jpg")

prompt = """
Extract all text from this receipt.
Return as structured JSON with:
- merchant_name
- date
- items (array of {name, price})
- total
"""

result = client.models.generate_content(
    model=IMAGE_MODEL,
    contents=[prompt, doc_image]
)
```

# Chart and Graph Analysis

## Understanding Data Visualizations

```
# Load chart image
chart = Image.open("sales_chart.png")

prompt = """
Analyze this chart and provide:
1. Chart type
2. What data it shows
3. Key trends or insights
4. Approximate values for key data points
"""

result = client.models.generate_content(
    model=IMAGE_MODEL,
    contents=[prompt, chart]
)
```

# Mathematical Problem Solving

## Solve Math from Images

```
# Load image of handwritten math problem
math_image = Image.open("math_problem.jpg")

prompt = """
Solve this math problem step by step.
Show your work and explain each step.
"""

result = client.models.generate_content(
    model=IMAGE_MODEL,
    contents=[prompt, math_image]
)

print(result.text)
# Step 1: Identify the equation:  $2x + 5 = 13$ 
# Step 2: Subtract 5 from both sides:  $2x = 8$ 
```

# Audio Processing

## Speech Transcription

```
# Upload audio file
audio_file = client.files.upload(path="interview.mp3")

# Transcribe
result = client.models.generate_content(
    model=MODEL,
    contents=[
        "Transcribe this audio accurately. Include speaker labels if multiple speakers.",
        audio_file
    ]
)

print(result.text)
# Interviewer: Tell me about your experience...
# Candidate: I have 5 years of experience in...
```

**Supports:** MP3, WAV, OGG formats

# Video Understanding

## Analyze Video Content

```
# Upload video
video_file = client.files.upload(path="product_demo.mp4")

# Wait for processing
import time
while video_file.state == "PROCESSING":
    time.sleep(5)
    video_file = client.files.get(video_file.name)

# Analyze video
result = client.models.generate_content(
    model=MODEL,
    contents=[
        "Summarize this video. What product is being demonstrated and what are its key features?",
        video_file
    ]
)
print(result.text)
```

# Video Frame Analysis

## Extract Information from Specific Frames

```
prompt = """  
Analyze this video and:  
1. Identify the main subject  
2. Describe what happens in the first 10 seconds  
3. List any text visible in the video  
4. Describe the setting/location  
"""  
  
result = client.models.generate_content(  
    model=MODEL,  
    contents=[prompt, video_file]  
)  
  
print(result.text)
```

# PDF Document Intelligence

## Extract Information from PDFs

```
# Upload PDF  
pdf_file = client.files.upload(path="research_paper.pdf")
```

```
# Extract structured information  
prompt = """  
From this PDF, extract:  
1. Title and authors  
2. Abstract  
3. Main sections  
4. Key findings (as bullet points)  
5. References count
```

```
Return as JSON.  
"""
```

```
result = client.models.generate_content(  
    model=MODEL,  
    contents=[prompt, pdf_file]
```

# Multi-Page PDF Extraction

## Process Complex Documents

```
# Upload multi-page invoice
invoice_pdf = client.files.upload(path="invoice_multi.pdf")

prompt = """
Extract all line items from this invoice across all pages.
For each item provide: description, quantity, unit_price, total.
Also extract: invoice_number, date, vendor, grand_total.
```

Return as JSON.

"""

```
result = client.models.generate_content(
    model=MODEL,
    contents=[prompt, invoice_pdf]
)
```

```
invoice_data = json.loads(result.text)
```

# Advanced Features: Streaming

## Stream Responses in Real-Time

```
# Useful for long responses or chat interfaces
prompt = "Write a detailed explanation of quantum computing."

for chunk in client.models.generate_content_stream(
    model=MODEL,
    contents=prompt
):
    print(chunk.text, end='', flush=True)
```

### Benefits:

- Lower perceived latency
- Better user experience
- Can stop generation early

# Function Calling

## Let LLM Call Your Functions

```
def get_weather(location: str) -> dict:
    """Get current weather for a location"""
    # Call weather API
    return {"temp": 72, "condition": "sunny"}

# Define function for LLM
functions = [
    {
        "name": "get_weather",
        "description": "Get current weather",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {"type": "string", "description": "City name"}
            },
            "required": ["location"]
        }
    }
]

response = client.models.generate_content(
    model=MODEL,
    contents="What's the weather in Mumbai?",
    tools=functions)
```

# Search Grounding

## Ground Responses in Real-Time Web Search

```
from google.genai import types

# Enable Google Search grounding
result = client.models.generate_content(
    model=MODEL,
    contents="What were the latest developments in AI this week?",
    config=types.GenerateContentConfig(
        tools=[types.Tool(google_search=types.GoogleSearch())]
    )
)

print(result.text)
# Response will include recent, factual information from web search

# Access grounding metadata
for source in result.grounding_metadata.sources:
```

# Batch Processing

## Process Multiple Requests Efficiently

```
texts = [
    "This product is amazing!",
    "Terrible experience, very disappointed.",
    "It's okay, nothing special."
]

results = []
for text in texts:
    response = client.models.generate_content(
        model=MODEL,
        contents=f"Sentiment (Positive/Negative/Neutral): {text}"
    )
    results.append({
        'text': text,
        'sentiment': response.text.strip()
})
```

# Error Handling

## Robust API Calls

```
import time

def safe_generate(prompt, max_retries=3):
    for attempt in range(max_retries):
        try:
            response = client.models.generate_content(
                model=MODEL,
                contents=prompt
            )
            return response.text

        except Exception as e:
            if "RATE_LIMIT" in str(e) and attempt < max_retries - 1:
                wait_time = 2 ** attempt # Exponential backoff
                print(f"Rate limited. Waiting {wait_time}s...")
                time.sleep(wait_time)
                continue
            elif attempt == max_retries - 1:
                raise
            else:
                print(f"Error: {e}")
```

# Cost Management

## Understanding API Costs

### Gemini Pricing (approximate):

- Free tier: 15 requests/minute
- Input tokens: ~\$0.00025 per 1K tokens
- Output tokens: ~\$0.001 per 1K tokens
- Images: ~\$0.0025 per image

## Track Usage

```
response = client.models.generate_content(  
    model=MODEL,  
    contents=prompt  
)
```

# Best Practices

## Prompt Engineering

- 1. Be specific:** Clear instructions get better results
- 2. Provide examples:** Few-shot learning improves accuracy
- 3. Request format:** Specify desired output structure
- 4. Context first:** Give context before questions
- 5. Iterate:** Test and refine prompts

## Production Considerations

- Implement rate limiting
- Add retry logic with exponential backoff
- Cache responses when possible

# Comparison: Gemini vs OpenAI vs Claude

Feature	Gemini	GPT-4	Claude 3
Context Length	2M tokens	128K tokens	200K tokens
Multimodal	Text, Image, Audio, Video	Text, Image	Text, Image
Free Tier	15 req/min	No	No
Pricing	Lower	Higher	Medium
Strengths	Multimodal, long context	Reasoning	Safety, long context

## When to Use Each

- **Gemini:** Multimodal tasks, long documents, cost-effective
- **GPT-4:** Complex reasoning, code generation

# Real-World Use Cases

## Content Moderation

- Analyze images/videos for inappropriate content
- Detect spam and toxic text
- Classify user-generated content

## Document Processing

- Extract data from invoices, receipts
- Parse resumes and applications
- Analyze contracts and legal documents

## Customer Support

# Lab Preview

## What You'll Build Today

### Part 1: Text tasks (45 min)

- Sentiment analysis on your data
- Custom classification
- Information extraction

### Part 2: Vision tasks (60 min)

- Image description and tagging
- OCR on documents
- Object detection visualization

### Part 3: Multimodal applications (60 min)

# Questions?

## Get Ready for Lab!

### What to install:

```
pip install google-genai pillow requests matplotlib pandas numpy
```

### What you need:

- Gemini API key from [aistudio.google.com/apikey](https://aistudio.google.com/apikey)
- Sample images/documents to analyze
- Ideas for AI applications

### Resources:

- [Gemini API Docs](#)

# **See You in Lab!**

**Remember:** LLMs are powerful tools, but verify outputs for critical applications

Next week: Advanced AI topics and deployment