Introduction to AI and Large Language Models

A Practical Course in Python, APIs, and LLM Applications

Eirik Berger Abel

2025-09-10

Contents

Chapter 1

Welcome to AI and Large Language Models

Welcome to this comprehensive course on **Artificial Intelligence and Large Language Models (LLMs)**. This book is designed to take you from the fundamentals of Python programming to advanced applications of AI in real-world scenarios.

1.1 Course Overview

This course covers the essential skills needed to work with modern AI systems, particularly Large Language Models. You'll learn:

- **Python Programming**: Essential Python skills for AI and data analysis
- **LLM Fundamentals**: Understanding how language models work under the hood
- **API Integration**: Working with commercial and open-source LLM APIs
- **LangChain Framework**: Building sophisticated AI applications
- **Practical Applications**: Real-world projects including data scraping, analysis, and automation

1.2 Learning Objectives

By the end of this course, you will be able to:

1. Write Python code for data manipulation and analysis
2. Understand the technical foundations of Large Language Models
3. Integrate LLM APIs into your applications
4. Use the LangChain framework for complex AI workflows
5. Build practical AI applications for business and research

6. Apply AI techniques to real-world data analysis problems

1.3 Prerequisites

This course assumes:

- Basic familiarity with programming concepts (variables, functions, loops)
- Comfort with using computers and installing software
- No prior experience with AI or machine learning required
- Willingness to experiment and learn through hands-on practice

1.4 Course Structure

The course is organized into progressive chapters, each building upon the previous:

- **Chapter 1**: Introduction to Python for AI
- **Chapter 2**: Understanding Large Language Models
- **Chapter 3**: Working with LLM APIs
- **Chapter 4**: LangChain Framework
- **Chapter 5**: Advanced LangChain and Caching
- **Chapter 6**: Practical Applications and Case Studies

1.5 Getting Started

To get the most out of this course:

1. Set up a Python environment (we recommend Google Colab for beginners)
2. Create accounts for the required APIs (instructions provided in each chapter)
3. Follow along with the code examples
4. Complete the hands-on exercises
5. Experiment with variations of the examples

1.6 Contact and Support

For questions about this course, please contact the instructor at eirik.berger@gmail.com with 'BAN443' included in the subject line.

Chapter 2

Introduction to Python for AI

This chapter introduces the essential Python programming concepts you'll need to work with AI and Large Language Models. While the focus of this course is on understanding and applying AI technologies, having solid Python fundamentals will make everything else much easier.

2.1 Why Python for AI?

Python has become the de facto language for AI and machine learning for several reasons:

- **Rich ecosystem**: Extensive libraries for data science, machine learning, and AI
- **Readable syntax**: Easy to learn and understand, even for beginners
- **Strong community**: Large community with excellent documentation and support
- **Industry standard**: Most AI companies and researchers use Python

2.2 Setting Up Your Environment

2.2.1 Option 1: Google Colab (Recommended for Beginners)

Google Colab provides a free, cloud-based Python environment that's perfect for learning:

1. Go to colab.research.google.com
2. Sign in with your Google account

3. Create a new notebook
4. Start coding immediately!

2.2.2 Option 2: Local Installation

If you prefer to work locally, install Python and Jupyter:

```
# Install Python (if not already installed)
# Download from python.org or use a package manager

# Install Jupyter
pip install jupyter

# Install required packages
pip install pandas numpy matplotlib requests transformers torch
```

2.3 Essential Python Concepts

2.3.1 Variables and Data Types

Python uses dynamic typing, meaning you don't need to declare variable types:

```
# Numbers
age = 25
height = 5.9
is_student = True

# Strings
name = "Alice"
message = 'Hello, World!'

# Lists (arrays)
fruits = ["apple", "banana", "orange"]
numbers = [1, 2, 3, 4, 5]

# Dictionaries (key-value pairs)
person = {
    "name": "Alice",
    "age": 25,
    "city": "New York"
}
```

2.3.2 Functions

Functions allow you to organize and reuse code:

```python
def greet(name):
    """A simple function that greets someone."""
    return f"Hello, {name}!"

# Using the function
message = greet("Alice")
print(message)  # Output: Hello, Alice!
```

2.3.3 Working with Data

For AI work, you'll frequently work with structured data:

```python
import pandas as pd

# Create a simple dataset
data = {
    'name': ['Alice', 'Bob', 'Charlie'],
    'age': [25, 30, 35],
    'city': ['New York', 'London', 'Tokyo']
}

df = pd.DataFrame(data)
print(df)
```

2.3.4 Error Handling

Learning to handle errors gracefully is crucial:

```python
try:
    # Code that might fail
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
except Exception as e:
    print(f"An error occurred: {e}")
```

2.4 Working with APIs

APIs (Application Programming Interfaces) are how you'll interact with AI services:

```python
import requests

# Example API call
url = "https://api.example.com/data"
response = requests.get(url)
```

```python
if response.status_code == 200:
    data = response.json()
    print("Success!")
else:
    print(f"Error: {response.status_code}")
```

2.5 Key Libraries for AI Work

Here are the essential Python libraries you'll use throughout this course:

- **pandas**: Data manipulation and analysis
- **numpy**: Numerical computing
- **requests**: Making HTTP requests to APIs
- **transformers**: Working with pre-trained language models
- **torch**: Deep learning framework
- **langchain**: Framework for building LLM applications

2.6 Best Practices

1. **Use meaningful variable names**: `user_age` instead of `a`
2. **Add comments**: Explain what your code does
3. **Handle errors**: Always include error handling
4. **Test incrementally**: Run code frequently to catch errors early
5. **Use functions**: Break complex tasks into smaller functions

2.7 Next Steps

Now that you have the Python basics, you're ready to dive into the fascinating world of Large Language Models. In the next chapter, we'll explore how these models work and get hands-on experience with GPT-2.

Remember: Don't worry if you're not a Python expert yet. The most important thing is to start experimenting and learning through practice. Each chapter will build your skills progressively.

Chapter 3

Understanding Large Language Models

In this chapter, we'll dive deep into how Large Language Models (LLMs) work under the hood. Understanding these fundamentals will help you use LLMs more effectively and troubleshoot issues when they arise.

3.1 What Are Large Language Models?

Large Language Models are AI systems trained on vast amounts of text data to understand and generate human-like language. At their core, they are sophisticated pattern recognition systems that learn to predict the next word (or token) in a sequence.

3.1.1 Key Characteristics

- **Scale**: Trained on billions or trillions of parameters
- **Data**: Trained on massive text corpora from the internet
- **Architecture**: Based on transformer neural networks
- **Capability**: Can perform a wide range of language tasks

3.2 The Transformer Architecture

The transformer architecture, introduced in the paper "Attention Is All You Need" (Vaswani et al., 2017), is the foundation of modern LLMs:

3.2.1 Key Components

1. **Attention Mechanism**: Allows the model to focus on relevant parts of the input
2. **Self-Attention**: Enables the model to understand relationships between words
3. **Feed-Forward Networks**: Process the attended information
4. **Layer Normalization**: Stabilizes training and improves performance

3.3 Tokenization: From Text to Numbers

Before LLMs can process text, it must be converted into numbers. This process is called **tokenization**.

3.3.1 How Tokenization Works

```
from transformers import GPT2Tokenizer

# Load the GPT-2 tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Tokenize a word
inputs = tokenizer("inequality", return_tensors="pt")
print("Token IDs:", inputs.input_ids)
print("Token IDs as list:", inputs.input_ids[0].tolist())
```

3.3.2 Understanding Token IDs

Let's see what those numbers actually represent:

```
# Convert IDs back to tokens (strings)
ids = inputs.input_ids[0].tolist()
tokens = [tokenizer.decode([i]) for i in ids]

print("IDs:", ids)
print("Tokens:", tokens)
```

You'll notice that "inequality" becomes `[500, 13237]`. This means: - The word is split into subword tokens - Each token gets a unique ID number - The model works with these numbers, not the original text

3.4 Next Token Prediction: The Core of LLMs

At their most basic level, LLMs are **autocomplete systems**. They predict the next word given some input text.

3.4.1 How It Works

1. **Input**: A sequence of tokens (words/subwords)
2. **Processing**: The model processes this sequence through multiple layers
3. **Output**: A probability distribution over all possible next tokens
4. **Selection**: The model selects the most likely next token

3.4.2 Hands-On Example

Let's see this in action with GPT-2:

```
from transformers import GPT2TokenizerFast, GPT2LMHeadModel
import torch

# Load the model and tokenizer
tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")

def get_next_tokens(text, num_tokens=10):
    """Get the most likely next tokens for a given text."""
    # Tokenize input
    inputs = tokenizer(text, return_tensors="pt")

    # Get model predictions
    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits[0, -1, :]  # Last token's logits
        probabilities = torch.softmax(logits, dim=-1)

    # Get top predictions
    top_indices = torch.topk(probabilities, num_tokens).indices
    top_probs = torch.topk(probabilities, num_tokens).values

    results = []
    for idx, prob in zip(top_indices, top_probs):
        token = tokenizer.decode([idx])
        results.append((token, prob.item()))

    return results

# Example usage
text = "The future of artificial intelligence is"
predictions = get_next_tokens(text)

print(f"Input: '{text}'")
print("\nTop 10 most likely next tokens:")
for token, probability in predictions:
```

```
    print(f"  '{token}': {probability:.3f}")
```

3.5 Training Process

Understanding how LLMs are trained helps explain their capabilities and limitations:

3.5.1 1. Pre-training Phase

- **Data**: Massive text corpora (books, articles, websites)
- **Task**: Predict the next token in a sequence
- **Duration**: Weeks or months on powerful hardware
- **Cost**: Millions of dollars in compute resources

3.5.2 2. Fine-tuning Phase (Optional)

- **Data**: Smaller, task-specific datasets
- **Task**: Adapt the model for specific applications
- **Examples**: Instruction following, code generation, conversation

3.6 Model Sizes and Capabilities

Different model sizes offer different capabilities:

Model Size	Parameters	Capabilities
Small (117M)	117 million	Basic text completion
Medium (345M)	345 million	Better coherence
Large (774M)	774 million	Good performance
XL (1.5B)	1.5 billion	High-quality generation
2.5B+	2.5+ billion	State-of-the-art performance

3.7 Practical Implications

3.7.1 What This Means for You

1. **Context Windows**: Models have limits on how much text they can process at once
2. **Temperature**: Controls randomness in generation (0 = deterministic, 1 = creative)
3. **Token Limits**: Responses are limited by the model's maximum output length
4. **Cost**: Larger models are more expensive to use

3.7.2 Best Practices

1. **Be specific**: Clear prompts lead to better responses
2. **Provide context**: Give the model relevant background information
3. **Iterate**: Refine your prompts based on results
4. **Understand limitations**: Models can make mistakes or hallucinate

3.8 Common Misconceptions

3.8.1 What LLMs Are NOT

- **Not databases**: They don't store facts, they predict based on patterns
- **Not calculators**: Math abilities are limited and error-prone
- **Not always factual**: They can generate plausible but incorrect information
- **Not conscious**: They're sophisticated pattern matching systems

3.8.2 What LLMs ARE

- **Pattern recognition systems**: Excellent at finding patterns in text
- **Language models**: Trained to understand and generate human language
- **General-purpose tools**: Can be adapted for many different tasks
- **Statistical systems**: Based on probability and statistics

3.9 Next Steps

Now that you understand how LLMs work, you're ready to start using them through APIs. In the next chapter, we'll learn how to interact with commercial LLM services and build your first AI-powered applications.

The key takeaway is that LLMs are powerful tools, but they're not magic. Understanding their limitations and how they work will make you a much more effective user of these technologies.

Chapter 4

Applications

Some *significant* applications are demonstrated in this chapter.

4.1 Example one

4.2 Example two

Chapter 5

Final Words

We have finished a nice book.