Introduction

There's been a lot of material on the internet for prompting with articles like 30 prompts everyone has to know. A lot of that has been focused on the chatGPT web user interface, which many people are using to do specific and often one-off tasks. But,the power of large language models (LLMs) as a developer tool is unparallel. Using API calls to LLMs to quickly build software applications, is still very underappreciated.

Course outcomes

- First, you'll learn some prompting best practices for software development
- then we'll cover some common use cases
- summarizing
- inferring
- transforming
- expanding,
- then you'll build a chatbot using an LLM

Large language machines

A large language model (LLM) is a type of machine learning model that can perform a variety of natural language processing (NLP) tasks, including generating and classifying text, answering questions in a conversational manner and translating text from one language to another.

The label "large" refers to the number of values (parameters) the model can change autonomously as it learns. Some of the most successful LLMs have hundreds of billions of parameters.

LLMs are trained with immense amounts of data and use self-supervised learning to predict the next token in a sentence, given the surrounding context. The process is repeated over and over until the model reaches an acceptable level of accuracy.

Once an LLM has been trained, it can be fine-tuned for a wide range of NLP tasks, including:

- Building conversational chatbots like ChatGPT.
- Generating text for product descriptions, blog posts and articles.
- Answering frequently asked questions (FAQs) and routing customer inquiries to the most appropriate human.
- Analyzing customer feedback from email, social media posts and product reviews.
- Translating business content into different languages.
- Classifying and categorizing large amounts of text data for more efficient processing and analysis.

Large language models typically have a transformer-based architecture. This type of AI architecture uses self-attention mechanisms to calculate a weighted sum for an input sequence and dynamically determine which tokens in the sequence are most relevant to each other.

What Are Large Language Models Used For?

Large language models are used for few-shot and zero-shot scenarios when there is little or no domain-tailored data available to train the model.

Both few-shot and zero-shot approaches require the AI model to have good inductive bias and the ability to learn useful representations from limited (or no) data.

How Are Large Language Models Trained?

Most LLMs are pre-trained on a large, general-purpose dataset that is similar in statistical distribution to the task-specific dataset. The purpose of pre-training is for the model to learn high-level features that can be transferred to the fine-tuning stage for specific tasks.

The training process of a large language model involves:

- Pre-processing the text data to convert it into a numerical representation that can be fed into the model.
- Randomly assigning the model's parameters.
- Feeding the numerical representation of the text data into the model.
- Using a loss function to measure the difference between the model's outputs and the actual next word in a sentence.
- Optimizing the model's parameters to minimize loss.
- Repeating the process until the model's outputs reach an acceptable level of accuracy.

How Do Large Language Models Work?

A large language model uses deep neural networks to generate outputs based on patterns learned from training data.

Typically, a large language model is an implementation of a transformer architecture. Transformer architectures allow a machine learning model to identify relationships between words in a sentence — regardless of their position in the text sequence — by using self-attention mechanisms.

Unlike recurrent neural networks (RNNs) which use recurrence as the main mechanism for capturing relationships between tokens in a sequence, transformer neural networks use self-attention as their main mechanism for capturing relationships. The relationships between tokens in a sequence are calculated using attention scores that represent how import a token is in regards to the other tokens in the text sequence.

Examples of Large Language Models

Some of the most popular large language models are:

- GPT-3 (Generative Pretrained Transformer 3) – developed by OpenAI.
- BERT (Bidirectional Encoder Representations from Transformers) – developed by Google.
- RoBERTa (Robustly Optimized BERT Approach) – developed by Facebook AI.
- T5 (Text-to-Text Transfer Transformer) – developed by Google.
- CTRL (Conditional Transformer Language Model) – developed by Salesforce Research.
- Megatron-Turing – developed by NVIDIA

There are broadly two types of LLMs:

1- Base LLMs

2- Instruction-tuned LLMs

Base LLMs :-

Base LLMs have been trained to predict the next word based on text training data, often trained on a large amount of data from the internet and other sources to figure out what's the next most likely word to follow. So, for example, if you were to prompt base LLM: `once upon a time there was a unicorn`, it may complete the sentence as, that is it may predict the next several words like `that live in a magical forest with all unicorn friends.`

But if you were to prompt: `with what is the capital of France`, then based on what articles on the internet might have, it's quite possible that the base LLM will complete this with: `what is France's largest city`, `what is France's population` and so on, because articles on the internet could quite plausibly be lists of quiz questions about the country of France.

Base LLMs :-

A lot of momentum of LLM research and practice has been going on in instruction-tuned LLMs. An instruction-tuned LLM has been trained to follow instructions. So, if you were to ask it `what is the capital of France`, it's much more likely to output something like, `the capital of France is Paris`.

Training of instruction-tuned LLMs:

The way that instruction-tuned LLMs are typically trained is you start off with a base LLM that's been trained on a huge amount of text data and further train it, further fine-tune it with inputs and outputs that are instructions and good attempts to follow those instructions, and then often further refine using a technique called reinforcement learning from human feedback (RLHF), to make the system better able to be helpful and follow instructions.

Because instruction-tuned LLMs have been trained to be helpful, honest, and harmless, so for example, they are less likely to output problematic text such as toxic outputs compared to base LLM, a lot of the practical usage scenarios have been shifting toward instruction-tuned LLMs. Some of the best practices you find on the internet may be more suited for a base LLM, but for most practical applications today, we would recommend most people instead focus on instruction-tuned LLMs which are easier to use and also, because of the work of OpenAI and other LLM companies becoming safer and more aligned.

When you use an instruction-tuned LLM, think of giving instructions to another person, say someone that's smart but doesn't know the specifics of your task. So, when an LLM doesn't work, sometimes it's because the instructions weren't clear enough. For example, if you were to say, `please write me something about Alan Turing`. Well, in addition to that, it can be helpful to be clear about whether you want the text to focus on his scientific work or his personal life or his role in history or something else. And if you specify what you want the tone of the text to be, should it take on the tone like a professional journalist would write. Or is it more of a casual note that you dash off to a friend? That helps the LLM generate what you want.

So, this course will focus on best practices for instruction-tuned LLMs, which is what we recommend you use for most of your applications.

APIs

API stands for application programming interface—a set of definitions and protocols to build and integrate application software.

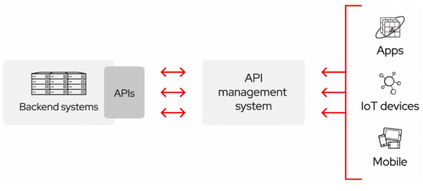
How do APIs work?

APIs let your product or service communicate with other products and services without having to know how they're implemented. This can simplify app development, saving time and money. When you're designing new tools and products—or managing existing ones—APIs give you flexibility; simplify design, administration, and use; and provide opportunities for innovation.

APIs are sometimes thought of as contracts, with documentation that represents an agreement between parties: If party 1 sends a remote request structured a particular way, this is how party 2's software will respond.

Because APIs simplify how developers integrate new application components into an existing architecture, they help business and IT teams collaborate. Business needs often change quickly in response to ever shifting digital markets, where new competitors can change a whole industry with a new app. In order to stay competitive, it's important to support the rapid development and deployment of innovative services. Cloud-native application development is an identifiable way to increase development speed, and it relies on connecting a microservices application architecture through APIs.

APIs are a simplified way to connect your own infrastructure through cloud-native app development, but they also allow you to share your data with customers and other external users. Public APIs represent unique business value because they can simplify and expand how you connect with your partners, as well as potentially monetize your data (the Google Maps API is a popular example).



For example, imagine a book-distributing company. The book distributor could give its customers a cloud app that lets bookstore clerks check book availability with the distributor. This app could be expensive to develop, limited by platform, and require long development times and ongoing maintenance.

Alternatively, the book distributor could provide an API to check stock availability. There are several benefits to this approach:

Letting customers access data via an API helps them aggregate information about their inventory in a single place.

The book distributor can make changes to its internal systems without impacting customers, so long as the behavior of the API doesn't change.

With a publicly available API, developers working for the book distributor, book sellers or third parties could develop an app to help customers find the books they're looking for. This could result in higher sales or other business opportunities.

In short, APIs let you open up access to your resources while maintaining security and control. How you open access and to whom is up to you. API security is all about good API management, which includes the use of an API gateway. Connecting to APIs, and creating applications that consume the data or functionality exposed by APIs, can be done with a distributed integration platform that connects everything—including legacy systems, and the Internet of Things (IoT).

API release policies

Private

The API is only for use internally. This gives companies the most control over their API.

Partner

The API is shared with specific business partners. This can provide additional revenue streams without compromising quality.

Public

The API is available to everyone. This allows third parties to develop apps that interact with your API and can be a source for innovation.

few-shot and zero-shot scenarios

If you want a machine to learn from data, you need to provide enough data to enable the machine to learn from. If you don't have enough data, the machine will have to find other ways to learn from data.

- **Zero-shot learning:** Zero-shot learning is when a machine is taught how to learn from data without ever needing to access the data itself. In zero-shot learning, a learner is given training data and asked to learn from it. This type of learning is often used for deep learning and machine learning applications, where the learner is not given any specific data to work with. `Zero-shot learning is a type of learning where a machine can learn from data without being explicitly taught how to do so.` Zero-shot learning is more common for deep learning and machine learning.
- **Few-shot learning:** Few-shot learning is when a machine is taught how to use data to learn from a specific point of view. In few-shot learning, the learner is given training data and asked to learn from it. This type of learning is often used for machine learning applications, where the learner is given a specific set of data to work with. `Few-shot learning is a type of learning where a machine can only learn from a few examples.` Few-shot learning is more common for rule-based learning.

Both zero-shot and few-shot learning can be used to teach a machine how to learn from data. However, each will have its unique strengths and weaknesses. Also, zero-shot learning could be more effective than few-shot learning because it may allow machines to learn from data without being explicitly told how to do so, leading to machines learning from various data sets and generalizing from examples.

In [ ]: