

*Pre-trained to generate human-like text based on the inputs in the form of questions in Natural language.*

# Documents Source: <https://www.accessdata.fda.gov/scripts/cder/daf/index.cfm?event=reportsSearch.process>

# Use case:

# The drugs that have been approved in the USA is listed in the Drugs@FDA database. To find information about the licenced pharmaceuticals, businesses sift through a vast volume of these pdfs, all these pdfs are ingested in Cresen Pharmagpt, which will significantly cut down on search and research time. The user can type down their question in Natural Language and gets answered within few seconds without parsing manually through any of the data in documents.

# Introduction

# *A description of the necessary modules*

# Large Language Model

# Large language models (LLMs) are recent advances in deep learning models to work on human languages. Some great use case of LLMs has been demonstrated. A large language model is a trained deep-learning model that understands and generates text in a human-like fashion. Behind the scenes, it is a large transformer model that does all the magic.

# Transformer to a Large Language Model

As humans, we perceive text as a collection of words. Sentences are sequences of words. Documents are sequences of chapters, sections, and paragraphs. However, for computers, text is merely a sequence of characters. To enable machines to comprehend text, [a model based on recurrent neural networks](https://machinelearningmastery.com/define-encoder-decoder-sequence-sequence-model-neural-machine-translation-keras/) can be built. This model processes one word or character at a time and provides an output once the entire input text has been consumed. This model works well, except it sometimes “forgets” what happened at the beginning of the sequence when the end is reached.

In 2017, Vaswani et al. published a paper, “Attention is All You Need,” to establish a [transformer model](https://machinelearningmastery.com/building-transformer-models-with-attention-crash-course-build-a-neural-machine-translator-in-12-days/). It is based on the attention mechanism. Contrary to recurrent neural networks, the attention mechanism allows you to see the entire sentence (or even the paragraph) at once rather than one word at a time. This allows the transformer model to understand the context of a word better. Many state-of-the-art language processing models are based on transformers.

To process a text input with a transformer model, you first need to tokenize it into a sequence of words. These tokens are then encoded as numbers and converted into embeddings, which are vector-space representations of the tokens that preserve their meaning. Next, the encoder in the transformer transforms the embeddings of all the tokens into a context vector.

Below is an example of a text string, its tokenization, and the vector embedding. Note that the tokenization can be sub words, such as the word “nosegay” in the text is tokenized into “nose” and “gay”.

A screenshot of a computer

Description automatically generated

# The context vector is like the essence of the entire input. Using this vector, the transformer decoder generates output based on clues. For instance, you can provide the original input as a clue and let the transformer decoder produce the subsequent word that naturally follows. Then, you can reuse the same decoder, but this time the clue will be the previously produced next word. This process can be repeated to create an entire paragraph, starting from a leading sentence.

# A diagram of a process Description automatically generated

# This process is called autoregressive generation. This is how a large language model works, except such a model is a transformer model that can take very long input text, the context vector is large so it can handle very complex concepts, and with many layers in its encoder and decoder.

# Azure Open AI

# Azure OpenAI Service provides REST API access to OpenAI's powerful language models including the GPT-4, GPT-35-Turbo, and Embeddings model series. In addition, the new GPT-4 and gpt-35-turbo model series have now reached general availability. These models can be easily adapted to your specific task including but not limited to content generation, summarization, semantic search, and natural language to code translation. Users can access the service through REST APIs, Python SDK, or our web-based interface in the Azure OpenAI Studio.

# GPT 3.5

# GPT-3.5 models can understand and generate natural language or code. Our most capable and cost-effective model in the GPT-3.5 family is gpt-3.5-turbo which has been optimized for chat but works well for traditional completions tasks as well.

# GPT-35-turbo

# Most capable GPT-3.5 model and optimized for chat at 1/10th the cost of text-davinci-003. Will be updated with our latest model iteration 2 weeks after it is released.

# The maximum tokens allowed in the model is 4096.

# Text-Davinci-003

Can do any language task with better quality, longer output, and consistent instruction-following than the curie, babbage, or ada models. Also supports some additional features such as [inserting text](https://platform.openai.com/docs/guides/gpt/inserting-text).

# [Embeddings](https://platform.openai.com/docs/models/embeddings)

# OpenAI’s text embeddings measure the relatedness of text strings. Embeddings are commonly used for:

# Search (where results are ranked by relevance to a query string)

# Clustering (where text strings are grouped by similarity)

# Recommendations (where items with related text strings are recommended)

# Anomaly detection (where outliers with little relatedness are identified)

# Diversity measurement (where similarity distributions are analysed)

# Classification (where text strings are classified by their most similar label)

# An embedding is a vector (list) of floating-point numbers. The [distance](https://platform.openai.com/docs/guides/embeddings/which-distance-function-should-i-use) between two vectors measures their relatedness. Small distances suggest high relatedness and large distances suggest low relatedness.

# image

# LangChain

# LangChain is a framework for developing applications powered by language models. We believe that the most powerful and differentiated applications will not only call out to a language model via an api, but will also:

# Be data-aware: connect a language model to other sources of data.

# Be agentic: Allow a language model to interact with its environment.

# As such, the LangChain framework is designed with the objective in mind to enable those types of applications.

# There are two main value props the LangChain framework provides:

# Components: LangChain provides modular abstractions for the components necessary to work with language models. LangChain also has collections of implementations for all these abstractions. The components are designed to be easy to use, regardless of whether one is using the rest of the LangChain framework or not.

# Use-Case Specific Chains: Chains can be thought of as assembling these components in particular ways to best accomplish a particular use case. These are intended to be a higher-level interface through which people can easily get started with a specific use case. These chains are also designed to be customizable.

# Accordingly, we split the following documentation into those two value props. In this documentation, we go over components and use cases at high level and in a language-agnostic way. For language-specific ways of using these components and tackling these use cases, please see the language-specific sections linked at the top of the page.

# LangChain and Chroma

# A close up of a chain Description automatically generated

# LangChain provides a framework to easily prototype LLM applications locally, and Chroma provides a vector store and embedding database that can run seamlessly during local development to power these applications. Chroma makes it easier to build LLM apps by making knowledge, skills, and facts pluggable for LLMs. Together LangChain and Chroma are the perfect fit.

# *from langchain.vectorstores import Chroma*

# A screenshot of a computer Description automatically generated

# Embedding with Azure Embedding Model and storing it in ChromaDB Vectorstore

**Working**

# A screenshot of a computer Description automatically generated

# Flowchart describing the working with the output.

# A screenshot of a chat Description automatically generated

# Azure Web app User Interface

# Implementation

* Environment setup

# 

# Documents Loading

# A computer screen shot of a computer code Description automatically generated

# A screenshot of a computer program Description automatically generated

# A screen shot of a computer code Description automatically generated

# A screenshot of a computer code Description automatically generated

# A screenshot of a computer program Description automatically generated

* Splitting the text from the documents and creating chunks

# A close-up of a computer code Description automatically generated

# Vector store Creation (Embeddings and ChromaDB)

# A screen shot of a computer code Description automatically generated

# Loading, Initializing and Creating chatbot.

# A screen shot of a computer program Description automatically generated

# 

# A screen shot of a computer program Description automatically generated

# Deployment to Azure Web Application

# 