



§1.1 Table of Content

- [§1 OpenAI ChatGPT - BEGINNERS VADEMECUM v1.10.6](#)
 - [§1.1 Table of Content](#)
 - [§1.2 FAQs](#)
 - [§1.2.1 What is ChatGPT?](#)
 - [§1.2.2 Is ChatGPT currently down?](#)
 - [§1.2.3 When is ChatGPT available?](#)
 - [§1.2.4 How accurate is ChatGPT?](#)
 - [§1.2.5 Can ChatGPT communicate in multiple languages?](#)
 - [§1.2.6 Can ChatGPT learn from its conversations?](#)
 - [§1.2.7 Can I ask personal questions to ChatGPT?](#)
 - [§1.2.8 What can I do with ChatGPT?](#)
 - [§1.2.9 Can I be banned from using ChatGPT?](#)
 - [§1.2.10 Does ChatGPT have any bias?](#)
 - [§1.2.11 Who can view my conversations?](#)
 - [§1.2.12 How does ChatGPT know what time it is?](#)
 - [§1.2.13 Can I have multiple accounts?](#)
 - [§1.2.14 Where can I access ChatGPT?](#)
 - [§1.2.15 Is there an app for ChatGPT?](#)
 - [§1.2.16 How do I get access to GPT-4?](#)
 - [§1.2.17 Can I use the output of ChatGPT as my own?](#)
 - [§1.2.18 Is ChatGPT free?](#)
 - [§1.2.19 What about the features that involve uploading a photo?](#)
 - [§1.2.20 How about plugins?](#)
 - [§1.2.21 My payment doesn't work from a banned country, what should I do?](#)
 - [§1.2.22 Can ChatGPT-generated text be detected?](#)
 - [§1.2.23 What if I want to sell something created with ChatGPT?](#)
 - [§1.2.24 Will you use my conversations for training?](#)
 - [§1.2.25 What if I don't want my conversations used?](#)
 - [§1.2.26 Why can't ChatGPT solve complex math problems?](#)
 - [§1.3 Basic GLOSSARY](#)
 - [§1.3.1 Action Space](#)
 - [§1.3.2 Artificial Intelligence \(or "AI"\)](#)
 - [§1.3.3 Artificial Neural Networks](#)
 - [§1.3.4 Autoencoder](#)
 - [§1.3.5 Bagging](#)
 - [§1.3.6 Big data](#)
 - [§1.3.7 Boosting](#)
 - [§1.3.8 Chat-ML](#)
 - [§1.3.9 Chatbots](#)
 - [§1.3.10 Classification](#)
 - [§1.3.11 Clustering](#)
 - [§1.3.12 Cognitive computing](#)
 - [§1.3.13 Context](#)
 - [§1.3.14 Conversational AI](#)
 - [§1.3.15 Convolutional neural network](#)
 - [§1.3.16 Cosine Similarity](#)
 - [§1.3.17 Cross-validation](#)
 - [§1.3.18 Data mining](#)
 - [§1.3.19 Decision tree](#)
 - [§1.3.20 Deep dream](#)
 - [§1.3.21 Deep Learning](#)
 - [§1.3.22 Dimensionality reduction](#)
 - [§1.3.23 Discount factor](#)

- [§1.3.24 Discriminative Models](#)
- [§1.3.25 Distance Between Two Vectors](#)
- [§1.3.26 Dynamic programming](#)
- [§1.3.27 Embedding](#)
- [§1.3.28 Engine](#)
- [§1.3.29 Ensemble learning](#)
- [§1.3.30 Episode](#)
- [§1.3.31 Epoch](#)
- [§1.3.32 Euclidean Distance](#)
- [§1.3.33 Exploration-exploitation tradeoff](#)
- [§1.3.34 Feature engineering](#)
- [§1.3.35 Feature selection](#)
- [§1.3.36 Fine-Tuning](#)
- [§1.3.37 Frequency penalty](#)
- [§1.3.38 Function Calling](#)
- [§1.3.39 Generative Adversarial Network \(GAN\)](#)
- [§1.3.40 Generative Models](#)
- [§1.3.41 Generative Pre-trained Transformer \(or "GPT"\)](#)
- [§1.3.42 Hallucinations](#)
- [§1.3.43 Hyperparameter tuning](#)
- [§1.3.44 Image segmentation](#)
- [§1.3.45 Image-to-image translation](#)
- [§1.3.46 Inductive Learning](#)
- [§1.3.47 Inference](#)
- [§1.3.48 Inner Product](#)
- [§1.3.49 Inpainting](#)
- [§1.3.50 Instruct](#)
- [§1.3.51 Jaccard Similarity](#)
- [§1.3.52 K-means](#)
- [§1.3.53 Knowledge Base](#)
- [§1.3.54 Large Language Model \(or "LLM"\)](#)
- [§1.3.55 Machine Learning \(or "ML"\)](#)
- [§1.3.56 Maximum a Posteriori Estimation \(MAP\)](#)
- [§1.3.57 Maximum Context Window \(MCW\)](#)
- [§1.3.58 Mindmaps](#)
- [§1.3.59 Monte Carlo method](#)
- [§1.3.60 Multi-armed bandit](#)
- [§1.3.61 N-Shot Learning](#)
- [§1.3.62 Naive Bayes classifier](#)
- [§1.3.63 Natural Language Generation \(or "NLG"\)](#)
- [§1.3.64 Natural Language Processing \(or "NLP"\)](#)
- [§1.3.65 Natural Language Understanding \(or "NLU"\)](#)
- [§1.3.66 Neural Network](#)
- [§1.3.67 OpenAI](#)
- [§1.3.68 Ontology](#)
- [§1.3.69 OpenAI's Whisper](#)
- [§1.3.70 Optimal policy](#)
- [§1.3.71 Optimal value function](#)
- [§1.3.72 Optimization](#)
- [§1.3.73 Overfitting](#)
- [§1.3.74 Plugins](#)
- [§1.3.75 Preprocessing](#)
- [§1.3.76 Presence penalty](#)
- [§1.3.77 Pre-training](#)
- [§1.3.78 Prompt Chaining](#)
- [§1.3.79 Prompt](#)
- [§1.3.80 Prompt Drifting](#)
- [§1.3.81 Prompt Engineering](#)
- [§1.3.82 Prompt Maintenance](#)
- [§1.3.83 Q-learning](#)
- [§1.3.84 Regression](#)
- [§1.3.85 Reinforcement Learning](#)
- [§1.3.86 Reinforcement Learning From Human Feedback \(RLHF\)](#)
- [§1.3.87 Reverse Prompting](#)
- [§1.3.88 Reward function](#)
- [§1.3.89 SARSA](#)
- [§1.3.90 Semantic web](#)
- [§1.3.91 Sequence Modeling](#)
- [§1.3.92 Similarity Search](#)
- [§1.3.93 Sparse Vector](#)
- [§1.3.94 Stable Diffusion](#)
- [§1.3.95 Standardization](#)
- [§1.3.96 State space](#)
- [§1.3.97 State transition](#)
- [§1.3.98 Style transfer](#)
- [§1.3.99 Supervised Learning](#)
- [§1.3.100 Tanimoto Similarity](#)
- [§1.3.101 Temperature](#)
- [§1.3.102 Temporal difference learning](#)
- [§1.3.103 Text-to-image model](#)
- [§1.3.104 Token](#)
- [§1.3.105 Tokenization](#)
- [§1.3.106 Top p](#)
- [§1.3.107 Training](#)
- [§1.3.108 Trajectory](#)
- [§1.3.109 Transfer learning](#)
- [§1.3.110 Transformer](#)
- [§1.3.111 Tree of Thought Prompt](#)
- [§1.3.112 Underfitting](#)
- [§1.3.113 Unsupervised Learning](#)
- [§1.3.114 Value iteration](#)
- [§1.3.115 Variational autoencoder](#)
- [§1.3.116 Vector Embeddings](#)
- [§1.4 GPT-4 PROMPT TACTICS](#)
 - [§1.4.1 Tactic: Include details in your query to get more relevant answers](#)
 - [§1.4.2 Tactic: Ask the model to adopt a persona](#)
 - [§1.4.3 Tactic: Use delimiters to clearly indicate distinct parts of the input](#)

- [§1.4.4 Tactic: Specify the steps required to complete a task](#)
- [§1.4.5 Tactic: Provide examples](#)
- [§1.4.6 Tactic: Specify the desired length of the output](#)
- [§1.4.7 Tactic: Instruct the model to answer using a reference text](#)
- [§1.4.8 Tactic: Instruct the model to answer with citations from a reference text](#)
- [§1.4.9 Tactic: Use intent classification to identify the most relevant instructions for a user query](#)
- [§1.4.10 Tactic: For dialogue applications that require very long conversations, summarize or filter previous dialogue](#)
- [§1.4.11 Tactic: Summarize long documents piecewise and construct a full summary recursively](#)
- [§1.4.12 Tactic: Instruct the model to work out its own solution before rushing to a conclusion](#)
- [§1.4.13 Tactic: Use inner monologue or a sequence of queries to hide the model's reasoning process](#)
- [§1.4.14 Tactic: Ask the model if it missed anything on previous passes](#)
- [§1.4.15 Tactic: Use embeddings-based search to implement efficient knowledge retrieval](#)
- [§1.4.16 Tactic: Use code execution to perform more accurate calculations or call external APIs](#)
- [§1.4.17 Tactic: Give the model access to specific functions](#)
- [§1.4.18 Tactic: Evaluate model outputs with reference to gold-standard answers](#)
- [§1.5 Prohibitions](#)
 - [§1.5.1 Queries ChatGPT won't respond to by design](#)
 - [§1.5.2 Jailbreaking Prompts](#)
- [§1.6 How to Reduce Your OpenAI API Costs](#)
 - [§1.6.1 Four TIPS to Save Tokens](#)
- [§1.7 BASIC LINKS AND RESOURCES FOR GETTING STARTED](#)
 - [§1.7.1 OpenAI OFFICIAL RESOURCES](#)
 - [§1.7.2 PROMPTS COLLECTIONS \(FREE\)](#)
 - [§1.7.3 PROMPTS COLLECTIONS \(PAID\)](#)
 - [§1.7.4 PROMPTS GENERATORS](#)
 - [§1.7.5 PROMPT CHAINING](#)
 - [§1.7.6 PROMPT APPIFICATION](#)
 - [§1.7.7 GUIDES AND TUTORIALS ABOUT PROMPTS AND ChatGPT](#)
 - [§1.7.8 BOOKS ABOUT ChatGPT AND PROMPT ENGINEERING](#)
 - [§1.7.9 ChatGPT PLAYGROUNDS AND ALTERNATIVE UIs](#)
 - [§1.7.10 Code Generation & Automated Programming](#)
 - [§1.7.11 IDE Powered by GPT-4](#)
 - [§1.7.12 LLM Agents Managers](#)
 - [§1.7.13 ChatGPT Discord Servers](#)
 - [§1.7.14 ChatGPT BOTS for Discord Servers](#)
 - [§1.7.15 ChatGPT BOTS for Telegram](#)
 - [§1.7.16 ChatGPT BOTS for Slack](#)
 - [§1.7.17 ChatGPT BOTS for WeChat](#)
 - [§1.7.18 ChatGPT BOTS for Feishu](#)
 - [§1.7.19 ChatGPT BOTS for DingTalk](#)
 - [§1.7.20 ChatGPT BOTS for Microsoft Teams](#)
 - [§1.7.21 AI LINKS DIRECTORIES](#)
 - [§1.7.22 CHROME Extensions](#)
 - [§1.7.23 Emacs Packages](#)
 - [§1.7.24 TRANSLATIONS AND GRAMMAR CORRECTION](#)
 - [§1.7.25 GitHub Apps and Extensions](#)
 - [§1.7.26 Command Line Tools](#)
 - [§1.7.27 ChatGPT API libraries](#)
 - [§1.7.28 LOADERS for sending documents to ChatGPT](#)
 - [§1.7.29 Function Calling](#)
 - [§1.7.30 AUTO-GPT Related](#)
 - [§1.7.31 ChatGPT Plug-ins](#)
 - [§1.7.32 GPT4ALL Related](#)
 - [§1.7.33 PROMPT ENGINEERING JOBS OFFERS](#)
- [§1.8 SOURCE OF THIS VADEMECUM](#)

End Of Table Of Content

...

§1.2 FAQs

§1.2.1 What is ChatGPT?

ChatGPT is an AI language model developed by OpenAI. It uses machine learning to generate human-like text based on user input. While it is not a human, it is designed to understand and respond to a broad range of topics in a conversational manner. It is capable of various tasks, including answering questions, manipulating text and source code, writing essays, and more.

§1.2.2 Is ChatGPT currently down?

If you're experiencing issues accessing ChatGPT, you can check its server status on the official [OpenAI Servers Status](#) page.

§1.2.3 When is ChatGPT available?

ChatGPT is available 24/7. However, due to its popularity, there might be periods of high demand leading to temporary unavailability.

§1.2.4 How accurate is ChatGPT?

As an AI language model trained on vast human conversations and text, ChatGPT can understand and respond to various topics. However, like any AI model, it is not perfect and may occasionally misunderstand user input or provide unsatisfactory responses. It is important to be very specific when asking things, providing as much context as possible to avoid errors. You can also use the 'regenerate answer' option to get multiple answers and choose the best one.

§1.2.5 Can ChatGPT communicate in multiple languages?

Yes, ChatGPT is capable of understanding and responding in multiple languages with varying degrees of accuracy.

§1.2.6 Can ChatGPT learn from its conversations?

No, ChatGPT does not learn from individual conversations. It can retain approximately 3000 words (or 4000 tokens) from the current conversation, but it does not store any information beyond that. If you use the API you can use the 'few shoot learning' mode to teach ChatGPT by examples.

§1.2.7 Can I ask personal questions to ChatGPT?

While ChatGPT can answer questions on a wide range of topics, it does not have personal experiences or opinions. It is incapable of providing personal insights or anecdotes. But if you tell it to act as a certain character, it can generate such content to match the character story and personality.

§1.2.8 What can I do with ChatGPT?

ChatGPT can generate various types of content, including stories, scripts, program designs, technical documentation, and more. You can also use it to convert between different text styles and generate dialogues by providing character details. You can ask ChatGPT to convert a chapter from a novel to screenplay format, for example.

§1.2.9 Can I be banned from using ChatGPT?

Yes, users violating the platform's terms of service or community guidelines may be banned from using ChatGPT. Always ensure to follow the guidelines and behave respectfully.

§1.2.10 Does ChatGPT have any bias?

As an AI trained on internet text data, ChatGPT may exhibit biases reflected in its training data. It is programmed to avoid discriminatory output, although this function is not infallible.

§1.2.11 Who can view my conversations?

Your conversations can be viewed by OpenAI staff for system improvement and to ensure content compliance with their policies.

§1.2.12 How does ChatGPT know what time it is?

It doesn't know. But ChatGPT can use hidden messages containing current time information when initiating a new conversation to infer the approximate time.

§1.2.13 Can I have multiple accounts?

Yes, there are no restrictions on the number of accounts a user can have on the ChatGPT public testing service.

§1.2.14 Where can I access ChatGPT?

You can access ChatGPT by visiting chat.openai.com.

§1.2.15 Is there an app for ChatGPT?

The official ChatGPT iOS app is available. Android users can anticipate an app soon. Alternatively, you can use third-party apps like Poe.com or Bing.

§1.2.16 How do I get access to GPT-4?

You can get access to GPT-4 by opting for the \$20/month membership, which grants you 25 messages every 3 hours. Alternatively, you can join the waitlist for the GPT-4 API.

§1.2.17 Can I use the output of ChatGPT as my own?

No, falsely claiming output from the services as human-generated is a violation of OpenAI's terms. You must always declare the output as AI generated.

§1.2.18 Is ChatGPT free?

Yes, the 3.5 version of ChatGPT is free. The 4.0 version costs \$20/month and provides you with 25 messages every 3 hours.

§1.2.19 What about the features that involve uploading a photo?

Features involving photo uploads are not yet released and are currently in the demo phase.

§1.2.20 How about plugins?

Certain plugins, including web browsing and 3rd party plugins, are available under beta features for plus users only.

§1.2.21 My payment doesn't work from a banned country, what should I do?

You can use a VPN to access ChatGPT, but you cannot use payment methods from banned countries.

§1.2.22 Can ChatGPT-generated text be detected?

While there are a few tells, existing detectors aren't completely accurate in distinguishing ChatGPT-generated text.

§1.2.23 What if I want to sell something created with ChatGPT?

The content generated by ChatGPT legally belongs to you.

§1.2.24 Will you use my conversations for training?

Yes, your conversations may be used by AI trainers for improvement. An incognito version is available in settings that won't save your chat history.

§1.2.25 What if I don't want my conversations used?

You can use the API instead of the web app of ChatGPT. For the API, your data only stays in the servers for 30 days for abuse detection, and is deleted afterward. It is not used for training data like ChatGPT does. You can also use the incognito mode from settings in ChatGPT.

§1.2.26 Why can't ChatGPT solve complex math problems?

ChatGPT's capabilities in solving complex math problems are limited. However, the Wolfram plugin can assist with that.

§1.3 Basic GLOSSARY

§1.3.1 Action Space

The set of all possible actions that an AI agent can take in a reinforcement learning problem.

§1.3.2 Artificial Intelligence (or “AI”)

The simulation of human intelligence in machines that are programmed to think and learn like humans.

§1.3.3 Artificial Neural Networks

ANNs are computing systems composed of neurons, where each neuron individually performs only a simple computation. The power of an ANN comes from the complexity of the connections these neurons can form. ANNs accept input variables as information, weigh variables as knowledge, and output a prediction. Every ANN works this way. The first ANN (the Perceptron) was created by Frank Rosenblatt in 1958. At the beginning, ANNs were built and used to solve basic tasks, but they rapidly evolved, becoming complex mechanisms able to solve challenges in areas like Computer Vision and Natural Language Processing. ANNs architectures were improved by the Deep Learning algorithms in order to fit the complexity of the data we started gathering; Convolutional Neural Networks (CNNs) were designed to process spatial data like images, while Recurrent Neural Networks (RNNs) and Long Short Term Memories (LSTMs) were built to process sequential data like text. But it wasn't until just recently that the invention of Transformers (see) changed the way we conceived ANNs. If Deep Learning was the most important invention in the history of AI, Transformers were certainly the second most important. Introduced in 2017, Transformers rapidly showed effective results at modelling data with long-range dependencies. Originally thought to solve NLP tasks, the application of Transformers has expanded, reaching incredible accomplishments in many disciplines, even unlocking the ability to predict the tridimensional shape of proteins, a major scientific advance for developing treatments for diseases. DeepMind's AlphaFold ANN system has been recognised as a solution to this grand challenge.

§1.3.4 Autoencoder

A type of neural network used for dimensionality reduction and feature learning, consisting of an encoder and a decoder that learn to compress and reconstruct data respectively.

§1.3.5 Bagging

A machine learning ensemble technique in which multiple models are trained on different random subsets of the training data and combined to make predictions, with the goal of reducing the variance and improving the generalization of the model.

§1.3.6 Big data

Large sets of data that can be analyzed by the AI and used to gain insights and make informed decisions.

§1.3.7 Boosting

A machine learning ensemble technique in which weak learners are combined to form a strong learner, with the goal of improving the overall prediction accuracy.

§1.3.8 Chat-ML

A jsonl formatted text used by chat completions models to store questions and answers of a conversation. Usually used for fine-tuning or to provide context to the engine.

§1.3.9 Chatbots

A user-friendly interface that allows the user to ask questions and receive answers. Depending on the backend system that fuels the chatbot, it can be as basic as pre-written responses to a fully conversational AI that automates issue resolution.

§1.3.10 Classification

The process of categorizing data into predefined classes or groups.

§1.3.11 Clustering

The process of grouping data points together based on similarity or common characteristics.

§1.3.12 Cognitive computing

The development of computer systems that can perform tasks that normally require human-like intelligence, such as learning and problem-solving.

§1.3.13 Context

A prompt can be considered as isolated question or a part of a conversation. The previous part of the conversation of a chat session is called 'context', and can be provided to the engine in the Chat-ML format (see). The context can also be composed by fine-tuning (see) or embeddings (see).

§1.3.14 Conversational AI

A subfield of AI that focuses on developing systems that can understand and generate human-like language and conduct a back-and-forth conversation. Example: A chatbot that can understand and respond to customer inquiries in a natural and human-like manner.

§1.3.15 Convolutional neural network

A type of neural network specifically designed for image and video recognition tasks, using convolutional layers to learn and recognize patterns in the data.

§1.3.16 Cosine Similarity

Cosine similarity (or cosine distance) is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. The formula for cosine similarity is: $\text{Cosine similarity} = (\text{Dot product of two vectors}) / (\text{Product of Euclidean lengths of two vectors})$. Cosine similarity is a commonly used measure in information retrieval and text mining, where it is often used to measure the similarity of documents. It is also used in collaborative filtering, where it is used to measure the similarity of users or items.

§1.3.17 Cross-validation

A model evaluation technique in which the training data is split into multiple folds, and the model is trained and evaluated on each fold to obtain an estimate of its generalization performance.

§1.3.18 Data mining

The process of extracting useful patterns and knowledge from large datasets.

§1.3.19 Decision tree

A flowchart-like tree structure used to make decisions based on a series of binary splits.

§1.3.20 Deep dream

An image generation technique developed by Google that uses a convolutional neural network (see) to generate dream-like images by amplifying the features of the network.

§1.3.21 Deep Learning

A subfield of ML that uses neural networks with multiple layers to learn from data without human supervision. Example: A deep learning model that can recognize objects in an image by processing the image through multiple layers of neural networks. The invention of Deep Learning was the main turning point in the history of AI. The earliest deep-learning-like algorithms that had multiple layers of non-linear features can be traced back to Ivakhnenko and Lapa in 1965, who used thin but deep models with polynomial activation functions which they analyzed with statistical methods. In each layer, they selected the best features through statistical methods and forwarded them to the next layer. They did not use backpropagation to train their network end-to-end but used layer-by-layer least squares fitting where previous layers were independently fitted from later layers. The key to make Deep Learning possible, the backpropagation algorithm, was derived already in the early 1960s but in an inefficient and incomplete form. It was finally cracked by Geoffrey Hinton, known by many to be the godfather of deep learning. His seminal 1986 paper on backpropagation that he wrote with David Rumelhart is universally considered the most important paper in the history of AI. Before Deep Learning, AI failure to deliver substantial results was leading to a progressive reduction in funding for research into neural networks. The term artificial intelligence dropped to near pseudoscience status during the so called 'AI winter' and the field didn't recover until the backpropagation algorithm started showing its capabilities in 2012 as the speed of GPUs increased rapidly and it was soon possible to train deep networks such as convolutional networks without the help of pretraining as demonstrated by Ciresan and colleagues in 2011 and 2012. Krizhevsky, Sutskever, and Hinton used a similar architecture in 2012 that also features rectified linear activation functions and dropout for regularization. They received outstanding results in the ILSVRC-2012 ImageNet competition, which marked the abandonment of feature engineering and the adoption of feature learning in the form of deep learning. Google, Facebook, and Microsoft noticed this trend and made major acquisitions of deep learning startups and research teams between 2012 and 2014. From here, research in deep learning accelerated rapidly. Previous machine learning algorithms leverage structured, manually labeled data to make predictions—meaning that specific features are defined from the input data for the model and organized into tables in advance. This doesn't necessarily mean that it doesn't use unstructured data at all; it just means that if it does, it generally goes through some manual pre-processing to organize it into a structured format. Deep learning eliminates most of data pre-processing that is typically involved with machine learning. Deep Learning algorithm can ingest and process unstructured data, like text and images, and it automates feature extraction, removing the dependency on human experts. For example, let's say that we had a set of photos of different pets, and we wanted to categorize by "cat", "dog", "hamster", et cetera. Deep learning algorithms can determine which features (e.g. ears) are most important to distinguish each animal from another. In machine learning, this hierarchy of features is established manually by a human expert. In Deep Learning, through the processes of gradient descent and backpropagation, the algorithm automatically creates multiple 'abstractions' and store them in meta-layers above of the main neural network layer, then it adjusts and fits those 'abstractions' for accuracy, allowing it to make predictions about a new photo of an animal with increased precision thanks to the abstract 'idea' of the object or the animal it has learned to recognize. The ancient theory of Plato about the impossibility of ideas to originate in the physical world was finally proved false.

§1.3.22 Dimensionality reduction

The process of reducing the number of features or dimensions in a dataset while preserving as much information as possible.

§1.3.23 Discount factor

The factor by which the future rewards are discounted in the reinforcement learning algorithm, to balance the tradeoff between the short-term and the long-term rewards.

§1.3.24 Discriminative Models

Models that classify a data example and predict a label. For example, a model that identifies whether a picture is a dog or a cat.

§1.3.25 Distance Between Two Vectors

The distance between two vectors is a measure of how different they are from each other. It is a numerical value that can be calculated using a variety of methods, such as Euclidean distance, dot product, or cosine similarity. This measure of distance is used in many machine learning and data science applications, such as clustering, classification, and recommendation systems. For example, in a clustering algorithm, the distance between two vectors can be used to determine which cluster they should be assigned to. In a classification algorithm, the distance between two vectors can be used to determine which class they should be assigned to. In a recommendation system, the distance between two vectors can be used to determine which items should be recommended to a user.

§1.3.26 Dynamic programming

A method of solving optimization problems by breaking them down into smaller subproblems and storing the solutions to these subproblems in a table or array.

§1.3.27 Embedding

Some models provide the option to embed documents (text, images, sounds, data files..) in the conversation. The embedded document is tokenized and converted in a format optimal for search or data extraction. Like for fine-tuning (see), and engine provided with embedded documents can be considered a specialized version of the generic original engine.

§1.3.28 Engine

The engine is the AI model employed to generate responses. For example: "text-davinci-003".

§1.3.29 Ensemble learning

A machine learning technique in which multiple models are trained and combined to make predictions, with the goal of improving the overall performance of the model.

§1.3.30 Episode

The sequence of states, actions, and rewards that an agent experiences in a reinforcement learning problem before reaching a terminal state.

§1.3.31 Epoch

One completed cycle of training of a Neural Network on a given Data Set. A greater number of epochs correspond to a more accurate and refined learning from the training Data Set, but an excessive number of epochs can lead to Overfitting (see).

§1.3.32 Euclidean Distance

The Euclidean Distance is a measure of the distance between two points in a Euclidean space. It is calculated by taking the square root of the sum of the squared differences between the coordinates of the two points. For example, if two points have coordinates (x_1, y_1) and (x_2, y_2) , then the Euclidean Distance between them is given by:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

The Euclidean Distance is a useful measure for many applications, such as clustering, classification, and regression. It is also used in machine learning algorithms such as k-means clustering and k-nearest neighbors.

§1.3.33 Exploration-exploitation tradeoff

The tension in reinforcement learning between exploring new actions and exploiting the known good actions, in order to balance the learning and the reward-maximizing objectives of the agent.

§1.3.34 Feature engineering

The process of selecting and creating informative and relevant features from raw data for use in a machine learning model.

§1.3.35 Feature selection

The process of choosing a subset of the most relevant features from a larger set of features for use in a machine learning model.

§1.3.36 Fine-Tuning

The process of adapting a pre-trained model to a specific task by training it on a smaller dataset. Fine-tuning is commonly used when a pre-trained model is available for a related task, but the available data for the new task is limited or the target task is slightly different from the original task.

Some models allow the user to provide conversation examples with many precompiled questions and answers to let the engine learn by example what the user expects from it. Using fine-tuning an engine can be trained to give responses that more accurately follow the user needs or are formatted in a special way. When using a fine-tuned engine, as when using documents Embedding (see), an engine can be considered a specialized version of the generic original engine.

§1.3.37 Frequency penalty

The frequency penalty parameter controls the model's tendency to repeat predictions. The frequency penalty reduces the probability of words that have already been generated. The penalty depends on how many times a word has already occurred in the prediction.

§1.3.38 Function Calling

The ability of an AI to answer a prompt with a line of code calling a function with specific arguments, so that the user can execute that line of code. Function calling allows the users to more reliably get structured data back from the model. For example in an OpenAI API call, you can describe functions to gpt-3.5-turbo-0613 and gpt-4-0613, and have the model intelligently choose to output a JSON object containing arguments to call those functions. The Chat Completions API does not call the function; instead, the model generates JSON that you can use to call the function in your code.

The latest models (gpt-3.5-turbo-0613 and gpt-4-0613) have been fine-tuned to both detect when a function should be called (depending on the input) and to respond with JSON that adheres to the function signature. With this capability also comes potential risks. We strongly recommend building in user confirmation flows before taking actions that impact the world on behalf of users (sending an email, posting something online, making a purchase, etc).

Under the hood, functions are injected into the system message in a syntax the model has been trained on. This means functions count against the model's context limit and are billed as input tokens. If running into context limits, the users need to limit the number of functions or the length of documentation they provide for function parameters. You can find an example of [function calling in the OpenAI API documentation](#).

§1.3.39 Generative Adversarial Network (GAN)

A type of neural network consisting of two competing networks, a generator and a discriminator, that learn to generate and recognize synthetic data respectively. The training of the generator network is the result of the generator network trying to generate various responses to a prompt (see) until one of those can convince the discriminator network of its correctness or of its similarity to the target result, called 'ground truth' (see).

§1.3.40 Generative Models

Models that generate new data by discovering patterns in data inputs or training data. For example, creating an original short story based on analyzing existing, published short stories.

§1.3.41 Generative Pre-trained Transformer (or "GPT")

A type of deep learning model trained on a large dataset to generate human-like text, the underlying architecture of ChatGPT

§1.3.42 Hallucinations

When LLMs struggle with niche or specific questions, for example because their knowledge base is missing the information they need, sometimes they generate false facts to come up with meaningful answers. Those false facts are called 'hallucinations'. Hallucinations are rarely obvious and likely to go undetected by system users. To avoid hallucinations the user needs to provide the LLM with additional "source knowledge" (information embedded directly into the model knowledge base), since only by filling up the missing information we can eliminate the need for the LLM to hallucinate and finally get accurate answers to our question.

§1.3.43 Hyperparameter tuning

The process of adjusting the parameters of a machine learning model that are set prior to training, in order to improve its performance.

§1.3.44 Image segmentation

The process of dividing an image into multiple regions or segments, each representing a different object or background.

§1.3.45 Image-to-image translation

The process of converting an image from one domain to another, such as translating a photograph to a painting or a sketch.

§1.3.46 Inductive Learning

Inductive learning is a type of machine learning that uses data to make predictions or generalizations about a given problem. It is based on the idea that if a set of data points have certain characteristics, then future data points will also have those characteristics. This type of learning is used in many areas of machine learning and data science, such as supervised learning, unsupervised learning, and reinforcement learning. For example, in supervised learning, inductive learning can be used to build a model that can predict the outcome of a given problem based on the data it has been trained on. In unsupervised learning, inductive learning can be used to identify patterns in data and make predictions about future data points. In reinforcement learning, inductive learning can be used to identify the best action to take in a given situation.

§1.3.47 Inference

The process of deducing conclusions from premises using logical reasoning.

§1.3.48 Inner Product

An inner product is a mathematical operation that takes two vectors of the same size and produces a scalar value. It is calculated by multiplying each element of one vector with the corresponding element of the other vector and then summing up the results. In machine learning and data science, inner products are used to measure the similarity between two vectors. For example, in a neural network, the inner product of two vectors can be used to measure the similarity between two input vectors, or between an input vector and a weight vector. Inner products can also be used to measure the similarity between two images, or between two text documents.

§1.3.49 Inpainting

The process of replacing, repairing or filling in damaged or missing parts of an image with AI create content.

§1.3.50 Instruct

INSTRUCT is a technique for getting LLM's to execute small programs as part of a prompt. It builds on other established prompting techniques like "lets think step by step" and "chain-of-thought" to improve the baseline models ability to reason and solve more complex problems. Using INSTRUCT you can get the model to more reliably perform complex tasks like planning, intent & entity recognition, and even avoid hallucinations. It only works with GPT-4. Here is an example of INSTRUCT Prompt:

PROMPT:

Text:
Nevertheless, in this mansion of gloom I now proposed to myself a sojourn of some weeks. Its proprietor, Roderick Usher, had been one of my boon companions in boyhood; but many years had elapsed since our last meeting. A letter, however, had lately reached me in a distant part of the country – a letter from him – which, in its wildly importunate nature, had admitted of no other than a personal reply. The MS. gave evidence of nervous agitation. The writer spoke of acute bodily illness – of a mental disorder which oppressed him – and of an earnest desire to see me, as his best, and indeed his only personal friend, with a view of attempting, by the cheerfulness of my society, some alleviation of his malady. It was the manner in which all this, and much more, was said – it was the apparent heart that went with his request – which allowed me no room for hesitation; and I accordingly obeyed forthwith what I still considered a very singular summons.

Editing Steps:

- Check for clarity and coherence.
- Check for flow and readability.
- Check for voice and tone.

Program:

- 1: create a variable to track the current editing step.
- 2: list the issues identified by the current editing step.
- 3: increment the step variable and if there are more editing steps left go to line 2.
- 4: apply the generated suggestions and return the updated text. Begin your response with RESPONSE.

State each line of the program and show your work for performing that line.

GPT-4 ANSWER:

1: create a variable to track the current editing step.

- Let step = 1

2: list the issues identified by the current editing step.

- The current editing step is to check for clarity and coherence.
- Some possible issues are:

- The phrase "a sojourn of some weeks" is vague and could be more specific.
- The phrase "a letter from him" is redundant and could be replaced by a pronoun.
- The phrase "in its wildly importunate nature" is unclear and could be simplified.
- The abbreviation "MS." is obscure and could be spelled out.
- The phrase "some alleviation of his malady" is weak and could be stronger.
- The phrase "it was the manner in which all this, and much more, was said" is wordy and could be condensed.
- The phrase "which allowed me no room for hesitation" is passive and could be active.

3: increment the step variable and if there are more editing steps left go to line 2.

- Let step = 2

2: list the issues identified by the current editing step.

- The current editing step is to check for flow and readability.
- Some possible issues are:

- The sentence "Its proprietor, Roderick Usher, had been one of my boon companions in boyhood; but many years had elapsed since our last meeting." is long and could be broken into two sentences.
- The sentence "The writer spoke of acute bodily illness – of a mental disorder which oppressed him – and of an earnest desire to see me, as his best, and indeed his only personal friend, with a view of attempting, by the cheerfulness of my society, some alleviation of his malady." is also long and could be divided into two sentences.
- The use of dashes and commas in the same sentence creates confusion and could be replaced by semicolons or periods.
- The use of "which" twice in the same sentence creates ambiguity and could be replaced by "that" or rephrased.
- The use of "it was" twice in the last sentence creates repetition and could be varied.

3: increment the step variable and if there are more editing steps left go to line 2.

- Let step = 3

2: list the issues identified by the current editing step.

- The current editing step is to check for voice and tone.

- Some possible issues are:

- The voice of the narrator is formal and distant, which may not match the intended mood of the story.
- The tone of the text is gloomy and somber, which may be appropriate for the setting, but could be contrasted with some hints of curiosity or suspense.
- The word choice and syntax are complex and archaic, which may reflect the time period of the story, but could also alienate modern readers.

4: apply the generated suggestions and return the updated text. Begin your response with RESPONSE.

RESPONSE: I decided to stay for several weeks in this mansion of gloom. Its proprietor was Roderick Usher, an old friend from my childhood. But I had not seen him for many years. He had sent me a letter, which begged me to come and visit him. He wrote that he suffered from a severe physical illness and a mental disorder that tormented him. He also expressed an urgent wish to see me, his best and only friend, hoping that my cheerful company might ease his pain. His words were full of emotion and sincerity, and I felt compelled to answer his strange request.

§1.3.51 Jaccard Similarity

The Jaccard similarity coefficient, or Jaccard Index, is a measure of similarity between two sets of data. It is calculated by taking the size of the intersection of the two sets and dividing it by the size of the union of the two sets. This gives us a value between 0 and 1, where 0 indicates no similarity and 1 indicates perfect similarity. The Jaccard similarity coefficient can be used in machine learning and data science in a variety of ways. For example, it can be used to measure the similarity between two documents, or to measure the similarity between two sets of data points. It can also be used to measure the similarity between two clusters of data points, or to measure the similarity between two sets of features. In addition, it can be used to measure the similarity between two sets of words or phrases.

§1.3.52 K-means

An unsupervised machine learning algorithm used for clustering data points into a predefined number of clusters.

§1.3.53 Knowledge Base

A knowledge base acts as the place where we store information and as the system that effectively retrieves this information. A knowledge base is a store of information that can act as an external reference for GQA models. We can think of it as the "long-term memory" for AI systems. We refer to knowledge bases that can enable the retrieval of semantically relevant information as vector databases. A vector database stores vector representations of information encoded using specific ML models. These models have an "understanding" of language and can encode passages with similar meanings into a similar vector space and dissimilar passages into a dissimilar vector space. In OpenAI API the process of storing a dataset in the knowledge base is called 'embedding' (see).

§1.3.54 Large Language Model (or "LLM")

A type of deep learning model trained on a large dataset to perform natural language understanding and generation tasks. There are many famous LLMs like BERT, PaLM, GPT-2, GPT-3, and the groundbreaking GPT3.5. All of these models vary in size (number of parameters that can be tuned), in the breadth of tasks (coding, chat, scientific, etc.), and in what they're trained on.

§1.3.55 Machine Learning (or "ML")

A subfield of AI that involves the development of algorithms and statistical models that enable machines to improve their performance with experience. Example: A machine learning algorithm that can predict which customers are most likely to churn based on their past behavior.

§1.3.56 Maximum a Posteriori Estimation (MAP)

Maximum a posteriori (MAP) estimation is a method of estimating the parameters of a statistical model. It is a type of Bayesian estimation, which uses Bayes' theorem to update the probability for a hypothesis as more evidence or information becomes available. MAP estimation can be used in machine learning and data science to estimate the parameters of a model, such as the weights of a neural network or the coefficients of a linear regression model. MAP estimation can also be used to estimate the probability of a given hypothesis, such as the probability of a given data point belonging to a certain class.

§1.3.57 Maximum Context Window (MCW)

The maximum length of our prompt and completion output. LLMs are limited by what can be considered analog to human's 'short-term memory' or 'working memory' (while the LLM knowledge base can be compared to human's 'long-term memory' or 'storage memory'). This limitation is measured by the 'maximum context window', and is so defined: context window = input tokens + output tokens. GPT-3 and GPT3.5 have a maximum context of 8,000 tokens, while GPT-4 offers up to 32,000 token for the context in his gpt-4-32k model. Usually the greater the size of the context window, the more capable and smart the LLM is.

§1.3.58 Mindmaps

You can ask GPT-4 to generate a 'mind map' of his knowledge about something. GPT-4 is familiar with the [markmap.js](https://markmap.js.org/) library. So if you ask it to create a detailed mindmap using the markup syntax, it will.

Here is how to do it:

- 1 - Ask GPT4 to create a "mind map of" something using markmap.js formatting. Example Prompt: "Please create a detailed mind map of an avocado using the markmap.js formatting".
- 2 - Copy the GPT response (the code part) and paste it to <https://markmap.js.org/repl>.

3 - Click 'Download interactive HTML' on the bottom of the graph to download the generated mindmap. You can also ask GPT-4 to reformat the mindmap to markdown, so you can import it in programs like XMind. Example Prompt: "Create a mindmap in markdown format on the subject of ai, ethics, and society at least 3 levels deep. Then reformat it in markdown and give me the raw markup code, so I can import it into programs like XMind."

§1.3.59 Monte Carlo method

A reinforcement learning algorithm that uses random sampling to estimate the value function or the optimal policy.

§1.3.60 Multi-armed bandit

A reinforcement learning problem in which an agent must choose among a set of actions, each with an unknown reward distribution, and learn which actions are the most rewarding through trial and error

§1.3.61 N-Shot Learning

Zero/Single/Few shot learning are variations of the same concept – providing a model with little or no training data to classify new data and guide predictions. A “shot” represents a single training example. Fun fact: Within the GPT prompt, you can ask for “N” examples to improve the accuracy of the response.

§1.3.62 Naive Bayes classifier

A machine learning algorithm used for classification tasks based on the Bayes theorem of probability.

§1.3.63 Natural Language Generation (or “NLG”)

A subfield of AI that produces natural written or spoken language.

§1.3.64 Natural Language Processing (or “NLP”)

A subfield of AI that involves programming computers to process massive volumes of language data. Focuses on transforming free-form text into a standardized structure.

§1.3.65 Natural Language Understanding (or “NLU”)

A subtopic of NLP that analyzes text to glean semantic meaning from written language. That means understanding context, sentiment, intent, etc.

§1.3.66 Neural Network

A machine learning model inspired by the human brain's structure and function that's composed of layers of interconnected nodes or "neurons." Example: A neural network that can recognize handwritten digits with high accuracy. See 'Artificial Neural Networks'.

§1.3.67 OpenAI

The organization that developed ChatGPT. More broadly speaking, OpenAI is a research company that aims to develop and promote friendly AI responsibly.

§1.3.68 Ontology

A system or set of definitions that represents the relationships and categories within a particular domain of knowledge.

§1.3.69 OpenAI's Whisper

OpenAI's Whisper is a new state-of-the-art (SotA) model in speech-to-text. It is able to almost flawlessly transcribe speech across dozens of languages and even handle poor audio quality or excessive background noise. Whisper is best described as the GPT-3 or DALL-E 2 of speech-to-text. It's open source and can transcribe audio in real-time or faster with unparalleled performance. OpenAI's Whisper speech-to-text-model is completely open source and available via OpenAI's Whisper library available for pip install via GitHub.

§1.3.70 Optimal policy

The policy that maximizes the expected cumulative reward in a reinforcement learning problem.

§1.3.71 Optimal value function

The value function that corresponds to the optimal policy in a reinforcement learning problem.

§1.3.72 Optimization

The process of adjusting the parameters of a model to minimize a loss function that measures the difference between the model's predictions and the true values. Example: Optimizing a neural network's parameters using a gradient descent algorithm to minimize the error between the model's predictions and the true values.

§1.3.73 Overfitting

A problem that occurs when a model is too complex, performing well on the training data but poorly on unseen data. Example: A model that has memorized the training data instead of learning general patterns and thus performs poorly on new data.

§1.3.74 Plugins

Plugins are third party piece of software that can add special actions to the AI models. ChatGPT-4 introduced dozen of plugins, that allowed actions like browsing the web, scrape websites, consult Wolfram Alpha for complex math equations and scientific data, draw graphs and maps, look for geographical data, stocks or news, upload PDFs or other documents, compose playlists, request stock photos or execute actions on other websites via third party services. There are two ways to interact to Plugins: the first standard way is to call the plugin directly with the prompt: "**Hey [plugin name], what can you help me with?**". The plugin will present itself and its options. The other is to let ChatGPT-4 use automatically the plugin whenever it thinks it is useful to answer your prompt.

§1.3.75 Preprocessing

The process of preparing the data for use in a machine learning model, including cleaning, transforming, and scaling the data.

§1.3.76 Presence penalty

The presence penalty parameter encourages the model to make novel predictions. The presence penalty lowers the probability of a word if it already appeared in the predicted text. Unlike the frequency penalty, the presence penalty does not depend on the frequency at which words appear in past predictions.

§1.3.77 Pre-training

Training a model on a large dataset before fine-tuning it to a specific task. Example: Pre-training a language model like ChatGPT on a large corpus of text data before fine-tuning it for a specific natural language task such as language translation.

§1.3.78 Prompt Chaining

A technique used to combine different Prompts in sequence, so that the answer to each Prompt is used as input of the subsequent Prompt. This technique allows to modularize the Prompts and enable recycling, swapping and composing of many Prompts steps, so that the Prompts become easier to create, customize and maintain. Here is an example of seven Prompts to be executed one after another as a chain:

1. Provide me with the ideal outline for an effective & persuasive blog post.
2. Write a list of Engaging Headlines for this Blog post based on [Topic].
3. Write a list of Subheadings & Hooks for this same blog post
4. Write a list of Keywords for this Blog.
5. Write a list of Compelling Call-to-Actions for the blog post
6. Combine the best headline with the best Subheadings, Hooks, Keywords and Call-to-Action to write a blog post for [topic]
7. Re-write this Blog Post in the [Style], [Tone], [Voice] and [Personality].

§1.3.79 Prompt

The prompt is the part of a conversation provided by the user. For example: "What is the capital of Austria?" Complex prompts can be used to guide the engine in giving a better answer. The field of knowledge regarding how to write effective prompts is called 'Prompt Engineering'. A Prompt can also include images, documents or other data. The most basic prompt structure is divided in 5 parts and called **CGTDO** or "**Five Fingers Prompt**":

- **CONTEXT**: In what role the AI Assistant must answer me? (*Example: "Act as a professional and experienced Lawyer."*)
- **GOAL** : What is that I need to do, my objective. (*Example: "I need to obtain a reimbursement from an airline company."*)
- **TASKS** : What I want YOU to do, your tasks. (*Example: "I want you to help me write the text of the letter to the airline company asking for the right reimbursement."*)
- **DATA**: All the information I have about the GOAL and that can be useful to complete the TASK. (*Example: Company Name, Airplane, Fly Number, Departure, Arrival, Cost, Date, Billing data, Cause of the delay/suspension, other relevant facts, documents, witness report, news articles, etc.*)
- **OUTPUT FORMAT** : What kind of format I want from ChatGPT for his answer. (*Example: Table, List, Summary, Outline, HTML, HTML5, SVG, Code, Spreadsheet, CSV file, Plain Text, Ascii, XML, json, Markdown markup code, Word Cloud etc. Also if you need the whole output in a single answer or to proceed step by step with multiple Q&A.*)

Less popular alternatives to the **CGTDO** (Five Fingers Prompt) prompt structure are:

- **RTF**: Role, Task, Format.
- **CTF**: Context, Task, Format
- **RASCEF**: Role, Action, Steps, Context, Examples, Format PECRA: Purpose, Expectation, Context, Request, Action.
- **TREF**: Task, Requirement, Expectation, Format.
- **GRADE**: Goal, Request, Action, Detail, Examples.
- **ROSES**: Role, Objective, Scenario, Expected Solution, Steps
- **RDIREC**: Role, Definition, Intent, Request, Example, Clarification Tone
- **RSCET**: Role, Situation, Complication, Expectation, Task

§1.3.80 Prompt Drifting

A phenomenon that happens when a perfectly working prompt stops working correctly after some time because the LLM has improved, fine tuned or trained so much that its new responses do not match anymore the original responses at the time the prompt was written. Prompt Drifting is considered a negative side effect of the constant evolution and improvement of the AI large language models. Prompt Drifting is usually solved retesting all prompts at regular intervals, and then comparing some predefined answers to the ones given at the time the prompt was written, and eventually rewriting and updating the prompts that do not pass the test. The LLM itself is usually instructed to make the comparison, to determine how much the prompt has drifted and if it has reached a point where the answers are no more functional to the original scope of the prompt. This task is called 'Prompt Maintenance'.

§1.3.81 Prompt Engineering

Identifying inputs (called: prompts) that result in the most meaningful outputs. As of now, prompt engineering is essential for LLMs. LLMs are a fusion of layers of algorithms and, consequently, have limited controllability with few opportunities to control and override behavior. Writing a prompt is then considered a two way process: on one side the prompt must asks the AI for something we need and on the other side it teaches the AI what it need to know to complete the request. It has been compared to a socratic dialog, an activity that Plato compared to the work of the obstetrician helping and guiding pregnant women (the AI) to give birth to their children (the right answer we seek). An advanced example of prompt engineering is providing a collection of templates and wizards to direct a copywriting application via a series of predefined steps, for example. Also multiple prompts can even be chained together to create complex prompts from a combination of simpler and smaller ones. Prompts are usually written in informal language, but there exist formal schemes devised to make them more easy to write, edit and maintain in a more programmatic way, like the '[Guidance](#)' scripting language for LLM prompts developed by Microsoft.

§1.3.82 Prompt Maintenance

The regular testing and adjustment of the prompts to avoid the negative consequences of Prompt Drifting (see).

§1.3.83 Q-learning

A reinforcement learning algorithm that learns an action-value function, also known as a Q-function, which estimates the expected future rewards for each action in a given state.

§1.3.84 Regression

A machine learning technique used to predict a continuous numerical value based on a set of input features.

§1.3.85 Reinforcement Learning

A type of machine learning in which a model learns to make decisions by interacting with its environment and receiving feedback through rewards or penalties. GPT uses reinforcement learning from human feedback (see). When tuning GPT-3, human annotators provided examples of the desired model behavior and ranked outputs from the model.

§1.3.86 Reinforcement Learning From Human Feedback (RLHF)

A type of machine learning technique in which an artificial intelligence (AI) system learns from the feedback or rewards provided by a human user or trainer.

§1.3.87 Reverse Prompting

An advanced prompt engineering technique where the user gets the AI to ask him questions as if he were the expert, leveraging the super human LLM capability to quickly find and summarize issues via its advanced pattern matching, while at the same time injecting his own human reasoning skills that the LLM lacks.

§1.3.88 Reward function

The function that defines the reward or the punishment that an agent receives for its actions in a reinforcement learning problem.

§1.3.89 SARSA

A reinforcement learning algorithm that learns an action-value function using the expected reward and the value of the next action, rather than the final reward as in Q-learning.

§1.3.90 Semantic web

An extension of the World Wide Web that enables machines to understand the meaning of the data on the web.

§1.3.91 Sequence Modeling

A subfield of NLP that focuses on modeling sequential data such as text, speech, or time series data. Example: A sequence model that can predict the next word in a sentence or generate coherent text.

§1.3.92 Similarity Search

When searching for similar text, there must be some kind of metric to check for overlap in the data. For text, this is simple: are the characters in the strings we are searching over close to the ones we have in our search string? There are various ways to define a metric for similarity. For example Hamming distance and Levenshtein distance are precise metrics for describing the similarity (or dissimilarity) of strings. If we wanted to determine similarity across movies, we might look at people who had watched and rated the same movies, along with what other movies they had watched and rated. To find product similarity, we could look at customer purchases and see if they have bought other items in the same purchase. And so on. Regardless of the types of objects we're searching through, we use "vectors" or "vector embeddings" (see) to convert the data we're analyzing into simpler numerical representations. For generic situations — such as text data — we can use widely available models like Word2vec. That model is trained against a wide collection of text data to determine similarities and differences between the concepts that real words represent. Then, we check for similarity on those vector representations while still maintaining the deeper meaning of the objects themselves. This is why 'similarity search' is sometimes called 'vector search'.

§1.3.93 Sparse Vector

A sparse vector is a vector that contains mostly zeros, with only a few non-zero elements. It is a useful data structure for representing data that is mostly empty or has a lot of zeros. For example, if you have a vector of length 10,000 and only 10 elements are non-zero, then it is a sparse vector. It is used in machine learning and data science when dealing with large datasets, as it can reduce the amount of memory needed to store the data. For example, if you have a dataset with millions of features where only a few features are important for each data point, you can represent it as a sparse vector, which will take up much less memory than a dense vector.

§1.3.94 Stable Diffusion

Stable Diffusion is a text-to-image model based on deep learning that can generate highly detailed images based on text descriptions.

§1.3.95 Standardization

The process of transforming the values of a feature or a dataset to have zero mean and unit variance.

§1.3.96 State space

The set of all possible states in a reinforcement learning problem.

§1.3.97 State transition

The movement of the agent from one state to another state in a reinforcement learning problem, based on an action and the transition probabilities of the environment.

§1.3.98 Style transfer

The process of transferring the style of one image to another image, while preserving the content of the second image.

§1.3.99 Supervised Learning

A type of machine learning in which a model is trained on labeled data to make predictions about new, unseen data. Example: A supervised learning algorithm that can classify images of handwritten digits based on labeled training data.

§1.3.100 Tanimoto Similarity

Tanimoto similarity is a measure of similarity between two sets of data. It is a metric used to compare the similarity of two sets of data, and is often used in machine learning and data science.

The Tanimoto similarity is calculated by taking the intersection of two sets and dividing it by the sum of the sizes of the two sets. This gives a value between 0 and 1, where 0 indicates no similarity and 1 indicates perfect similarity.

Tanimoto similarity is often used in machine learning and data science to compare the similarity of two sets of data. For example, it can be used to compare the similarity of two sets of images, or two sets of text documents. It can also be used to compare the similarity of two sets of data points, such as two sets of customer data. In this case, the Tanimoto similarity can be used to identify customers who are similar in terms of their purchase history or other characteristics.

It is similar to the Jaccard coefficient (see), which is the ratio of the intersection of two sets to the union of two sets. The Tanimoto similarity and Jaccard coefficient are both measures of similarity between two sets of data. The main difference between the two is that the Tanimoto similarity takes into account the size of the sets, while the Jaccard Index does not. The Tanimoto similarity is calculated by dividing the number of elements that are common to both sets by the total number of elements in both sets, while the Jaccard Index is calculated by dividing the number of elements that are common to both sets by the number of elements that are unique to either set.

§1.3.101 Temperature

A parameter of the model, with value between 0 (precise) and 1 (creative). A model generate different responses internally, and those are mapped into probabilities. To let the AI be creative and variable in the responses the engine introduced the concept of temperature. At temperature=0 only the most probable answer is returned, and the answer is deterministic (the same question return the same answer). But as the temperature value increases, the chance that the AI model will select a less probable response increases, and the answer become less deterministic and more creative (but also more prone to errors). With a temperature between 0 and 1, we can control the randomness and creativity of the model's predictions. For example if an user needs to generate poetry, he should use a temperature of 1.0. If instead he needs an answer to technical question or to generate programming code, he should use a temperature of 0. When in ChatGPT you don't have access to the parameters like in the playground or via the OpenAI API. This is why for certain subjects that require precision, like science or coding, is preferable to avoid ChatGPT but use the Playground or the OpenAI API instead.

§1.3.102 Temporal difference learning

A reinforcement learning algorithm that uses the temporal difference error, the difference between the estimated and the actual future rewards, to update the action-value function.

§1.3.103 Text-to-image model

A machine learning system that generates images based on natural language descriptions.

§1.3.104 Token

Tokens are used to count the size of questions and responses from an engine. A token can be seen as a piece of word. The size of a token depends on the engine and the language used. As a rule of thumb, 1 token is around 4 characters.

§1.3.105 Tokenization

The process of breaking text into individual words or subwords to input them into a language model. Example: Tokenizing a sentence "I am ChatGPT" into the words: "I," "am," "Chat," "G," and "PT."

§1.3.106 Top p

The top p parameter specifies a sampling threshold during inference time. Top p sampling (sometimes called nucleus sampling) is a technique used to sample possible outcomes of the model. To better understand this concept, let's imagine that the model has to predict the next token after the sentence "I want to eat". For simplicity, we will assume that a token is a word and that the model outputs the following probabilities:

carrots (2% likely),
cucumbers (1.5% likely),
aubergines (1% likely),
Spinach (0.5% likely),
Broccoli (0.3% likely),
...

This set of words forms a probability distribution $P(\text{Word} | \text{"I want to eat"})$ and the cumulative distribution function (CDF) would look as follows:

2% with carrots,
3.5% with carrots and cucumbers,
4.5% with carrots, cucumbers, and aubergines
...

If we set the top p-parameter to 3.5%, the model will sample and randomly select between carrots and cucumbers according to their likelihood. With a top p at 4.5%, the model will randomly select a word between carrots, cucumbers, and aubergines.

Like the temperature, the top p parameter controls the randomness and originality of the model.

Since Top p and temperature parameters both control the randomness of the model, OpenAI documentation recommends using either one parameter or the other.

§1.3.107 Training

Training refers to the process of learning from data to improve the performance of a model or system. Training is a fundamental part of machine learning and involves feeding a model with a dataset and adjusting its parameters or weights to optimize its performance for a specific task.

§1.3.108 Trajectory

The sequence of states and actions that an agent follows in a reinforcement learning problem.

§1.3.109 Transfer learning

A machine learning technique in which a model trained on one task is fine-tuned or adapted for a related task.

§1.3.110 Transformer

A type of neural network architecture designed to process sequential data, such as text. An example is the transformer architecture used in models like ChatGPT for natural language processing tasks. Before Transformers, ANN architectures, like RNNs, had severe memory problems. In the case of RNNs, there's a limited scope they can remember about long-range dependencies (the words they saw a long time ago that are somehow related to the next word). That is, RNNs put too much emphasis on words being close to one another and too much emphasis on upstream context over downstream context. Reading one word at a time, RNNs need to perform multiple steps to make decisions that depend on words far away from each other, which is incredibly slow. Self-attention fixes this problem. The Transformer starts by generating initial representations, or embeddings, for each word. Then, using self-attention, it aggregates information from all of the other words, generating a new representation per word informed by the entire context. This step is then repeated multiple times in parallel for all words, successively generating new representations.

Using self-attention mechanisms, Transformers can capture the context of a word from distant parts of a sentence, both before and after the appearance of that word, in order to encode valuable information. Sentences are processed as a whole, rather than word by word. This way, Transformer models avoid suffering from long dependency issues and forgetting past information. Self-attention is computed not once but multiple times in the Transformer's architecture, in parallel and independently (aply referred to as Multi-head Attention). Another advantage of Transformers is the improved performance of the ANN. The sequential nature of RNNs makes it more difficult to fully take advantage of fast modern computing devices such as TPUs and GPUs, which excel at parallel and non-sequential processing. Since the Transformer architecture lends itself to parallelization, finally we can really boost the speed with which these models can be trained. Transformers' successful results led to their escalation into massive models like OpenAI GPT-3, trained with absurd amounts of data, capable of performing the most diverse tasks. GPT-3, the third-generation GPT model, was trained on 175 billion parameters, about 10 times the size of previous models. In general, the more parameters a model has, the more data is required to train the model. As per the creators, the OpenAI GPT-3 model has been trained on about 45 TB text data from multiple sources which include Wikipedia and books. This gigantic pretrained model

provides users with the ability to fine-tune NLP tasks with very little data to accomplish novel tasks, like creating articles, poetry, stories, news reports and dialogue.

§1.3.111 Tree of Thought Prompt

A kind of Prompt where the AI is forced to simulate multiple points of view when answering a question, and simulate a dialog between those POV until they reach a consensus on the answer. This Prompt is used to force the AI to consider all aspects of an issue before answering. This technique is useful when the question is complex and requires in depth analysis. Here is an example of this kind of Prompt:

```
Three experts with exceptional logical thinking skills are collaboratively answering a question using a tree of thoughts method. Each expert will share their thought process in detail, taking into account the previous thoughts of others and admitting any errors. They will iteratively refine and expand upon each other's ideas, giving credit where it's due. The process continues until a conclusive answer is found. Use step by step thinking & organize the entire response in detailed steps in a markdown table format. Once this table is complete, provide a summary of the proposed recommendations.
```

My question is [write your question here]

§1.3.112 Underfitting

A condition in which a machine learning model performs poorly on both the training and new, unseen data, due to being too simple and unable to capture the underlying patterns in the data.

§1.3.113 Unsupervised Learning

A type of machine learning in which a model is trained on unlabeled data to find patterns or features in the data. Example: An unsupervised learning algorithm that can cluster similar images of handwritten digits based on their visual features.

§1.3.114 Value iteration

A reinforcement learning algorithm that involves iteratively improving the value function until it converges to the optimal value function.

§1.3.115 Variational autoencoder

A type of generative model that consists of an encoder network that maps the input data to a latent representation and a decoder network that maps the latent representation back to the original data space.

§1.3.116 Vector Embeddings

Vector embeddings are a simplified numerical representation of complex data, like text or images, used to make it easier to run generic machine-learning algorithms on sets of that data. By taking real-world objects and translating them to vector embeddings — numerical representations — those numbers can be fed into machine learning algorithms to determine semantic similarity. We store vector embeddings with IDs that tie our data back to the objects they represent, allowing us to search through that data. A store of vector embeddings and their IDs is called a “vector index.”

END of Glossary

...

§1.4 GPT-4 PROMPT TACTICS

The following prompts tactics only work on GPT-4 or higher.

§1.4.1 Tactic: Include details in your query to get more relevant answers

In order to get a highly relevant response, make sure that requests provide any important details or context. Otherwise you are leaving it up to the model to guess what you mean.

Worse

How do I add numbers in Excel?

Who's president?

Write code to calculate the Fibonacci sequence.

Better

How do I add up a row of dollar amounts in Excel? I want to do this automatically for a whole sheet of rows with all the totals ending up on the right in a column called "Total".

Who was the president of Mexico in 2021, and how frequently are elections held?

Write a TypeScript function to efficiently calculate the Fibonacci sequence. Comment the code liberally to explain what each piece does and why it's written that way.

Summarize the meeting notes.

Summarize the meeting notes in a single paragraph. Then write a markdown list of the speakers and each of their key points. Finally, list the next steps or action items suggested by the speakers, if any.

§1.4.2 Tactic: Ask the model to adopt a persona

The system message can be used to specify the persona used by the model in its replies.

SYSTEM

When I ask for help to write something, you will reply with a document that contains at least one joke or playful comment in every paragraph.

USER

Write a thank you note to my steel bolt vendor for getting the delivery in on time and in short notice. This made it possible for us to deliver an important order.

[Open in Playground](#)

§1.4.3 Tactic: Use delimiters to clearly indicate distinct parts of the input

Delimiters like triple quotation marks, XML tags, section titles, etc. can help demarcate sections of text to be treated differently.

USER

Summarize the text delimited by triple quotes with a haiku. """insert text here"""

[Open in Playground](#)

SYSTEM

You will be provided with a pair of articles (delimited with XML tags) about the same topic. First summarize the arguments of each article. Then indicate which of them makes a better argument and explain why.

USER

[article]"""insert first article here"""/[article] [article]"""insert second article here"""/[article]

[Open in Playground](#)

SYSTEM

You will be provided with a thesis abstract and a suggested title for it. The thesis title should give the reader a good idea of the topic of the thesis but should also be eye-catching. If the title does not meet these criteria, suggest 5 alternatives.

USER

Abstract: """insert abstract here""" Title: """insert title here"""

[Open in Playground](#)

For straightforward tasks such as these, using delimiters might not make a difference in the output quality. However, the more complex a task is the more important it is to disambiguate task details. Don't make GPTs work to understand exactly what you are asking of them.

§1.4.4 Tactic: Specify the steps required to complete a task

Some tasks are best specified as a sequence of steps. Writing the steps out explicitly can make it easier for the model to follow them.

SYSTEM

Use the following step-by-step instructions to respond to user inputs. Step 1 - The user will provide you with text in triple quotes. Summarize this text in one sentence with a prefix that says "Summary: ". Step 2 - Translate the summary from Step 1 into Spanish, with a prefix that says "Translation: ".

USER

"""insert text here"""

[Open in Playground](#)

§1.4.5 Tactic: Provide examples

Providing general instructions that apply to all examples is generally more efficient than demonstrating all permutations of a task by example, but in some cases providing examples may be easier. For example, if you intend for the model to copy a particular style of responding to user queries which is difficult to describe explicitly. This is known as "few-shot" prompting.

SYSTEM

Answer in a consistent style.

USER

Teach me about patience.

ASSISTANT

The river that carves the deepest valley flows from a modest spring; the grandest symphony originates

from a single note; the most intricate tapestry begins with a solitary thread.

USER

Teach me about the ocean.

[Open in Playground](#)

§1.4.6 Tactic: Specify the desired length of the output

You can ask the model to produce outputs that are of a given target length. The targeted output length can be specified in terms of the count of words, sentences, paragraphs, bullet points, etc. Note however that instructing the model to generate a specific number of words does not work with high precision. The model can more reliably generate outputs with a specific number of paragraphs or bullet points.

USER

Summarize the text delimited by triple quotes in about 50 words. """insert text here"""

[Open in Playground](#)

USER

Summarize the text delimited by triple quotes in 2 paragraphs. """insert text here"""

[Open in Playground](#)

USER

Summarize the text delimited by triple quotes in 3 bullet points. """insert text here"""

[Open in Playground](#)

§1.4.7 Tactic: Instruct the model to answer using a reference text

If we can provide a model with trusted information that is relevant to the current query, then we can instruct the model to use the provided information to compose its answer.

SYSTEM

Use the provided articles delimited by triple quotes to answer questions. If the answer cannot be found in the articles, write "I could not find an answer."

USER

"""insert articles, each delimited by triple quotes""" Question: """insert question here"""

[Open in Playground](#)

Given that GPTs have limited context windows, in order to apply this tactic we need some way to dynamically lookup information that is relevant to the question being asked. [Embeddings](#) can be used to implement efficient knowledge retrieval. See the tactic ["Use embeddings-based search to implement efficient knowledge retrieval"](#) for more details on how to implement this.

§1.4.8 Tactic: Instruct the model to answer with citations from a reference text

If the input has been supplemented with relevant knowledge, it's straightforward to request that the model add citations to its answers by referencing passages from provided documents. Note that citations in the output can then be verified programmatically by string matching within the provided documents.

SYSTEM

You will be provided with a document delimited by triple quotes and a question. Your task is to answer the question using only the provided document and to cite the passage(s) of the document used to answer the question. If the document does not contain the information needed to answer this question then simply write: "Insufficient information." If an answer to the question is provided, it must be annotated with a citation. Use the following format for to cite relevant passages ({"citation": ...}).

USER

"""insert document here""" Question: """insert question here"""

[Open in Playground](#)

§1.4.9 Tactic: Use intent classification to identify the most relevant instructions for a user query

For tasks in which lots of independent sets of instructions are needed to handle different cases, it can be beneficial to first classify the type of query and to use that classification to determine which instructions are needed. This can be achieved by defining fixed categories and hardcoding instructions that are relevant for handling tasks in a given category. This process can also be applied recursively to decompose a task into a sequence of stages. The advantage of this approach is that each query will contain only those instructions that are required to perform the next stage of a task which can result in lower error rates compared to using a single query to perform the whole task. This can also result in lower costs since larger prompts cost more to run ([see pricing information](#)).

Suppose for example that for a customer service application, queries could be usefully classified as follows:

SYSTEM

You will be provided with customer service queries. Classify each query into a primary category and a secondary category. Provide your output in json format with the keys: primary and secondary. Primary categories: Billing, Technical Support, Account Management, or General Inquiry. Billing secondary categories: - Unsubscribe or upgrade - Add a payment method - Explanation for charge - Dispute a charge Technical Support secondary categories: - Troubleshooting - Device compatibility - Software updates

Account Management secondary categories: - Password reset - Update personal information - Close account - Account security
General Inquiry secondary categories: - Product information - Pricing - Feedback - Speak to a human

USER

I need to get my internet working again.

[Open in Playground](#)

Based on the classification of the customer query, a set of more specific instructions can be provided to a GPT model to handle next steps. For example, suppose the customer requires help with "troubleshooting".

SYSTEM

You will be provided with customer service inquiries that require troubleshooting in a technical support context. Help the user by: - Ask them to check that all cables to/from the router are connected. Note that it is common for cables to come loose over time. - If all cables are connected and the issue persists, ask them which router model they are using - Now you will advise them how to restart their device: -- If the model number is MTD-327J, advise them to push the red button and hold it for 5 seconds, then wait 5 minutes before testing the connection. -- If the model number is MTD-327S, advise them to unplug and replug it, then wait 5 minutes before testing the connection. - If the customer's issue persists after restarting the device and waiting 5 minutes, connect them to IT support by outputting {"IT support requested"}. - If the user starts asking questions that are unrelated to this topic then confirm if they would like to end the current chat about troubleshooting and classify their request according to the following scheme: ""insert primary/secondary classification scheme from above here""

USER

I need to get my internet working again.

[Open in Playground](#)

Notice that the model has been instructed to emit special strings to indicate when the state of the conversation changes. This enables us to turn our system into a state machine where the state determines which instructions are injected. By keeping track of state, what instructions are relevant at that state, and also optionally what state transitions are allowed from that state, we can put guardrails around the user experience that would be hard to achieve with a less structured approach.

§1.4.10 Tactic: For dialogue applications that require very long conversations, summarize or filter previous dialogue

Since GPTs have a fixed context length, dialogue between a user and an assistant in which the entire conversation is included in the context window cannot continue indefinitely.

There are various workarounds to this problem, one of which is to summarize previous turns in the conversation. Once the size of the input reaches a predetermined threshold length, this could trigger a query that summarizes part of the conversation and the summary of the prior conversation could be included as part of the system message. Alternatively, prior conversation could be summarized asynchronously in the background throughout the entire conversation.

An alternative solution is to dynamically select previous parts of the conversation that are most relevant to the current query. See the tactic ["Use embeddings-based search to implement efficient knowledge retrieval"](#).

§1.4.11 Tactic: Summarize long documents piecewise and construct a full summary recursively

Since GPTs have a fixed context length, they cannot be used to summarize a text longer than the context length minus the length of the generated summary in a single query.

To summarize a very long document such as a book we can use a sequence of queries to summarize each section of the document. Section summaries can be concatenated and summarized producing summaries of summaries. This process can proceed recursively until an entire document is summarized. If it's necessary to use information about earlier sections in order to make sense of later sections, then a further trick that can be useful is to include a running summary of the text that precedes any given point in the book while summarizing content at that point. The effectiveness of this procedure for summarizing books has been studied in previous [research](#) by OpenAI using variants of GPT-3.

§1.4.12 Tactic: Instruct the model to work out its own solution before rushing to a conclusion

Sometimes we get better results when we explicitly instruct the model to reason from first principles before coming to a conclusion. Suppose for example we want a model to evaluate a student's solution to a math problem. The most obvious way to approach this is to simply ask the model if the student's solution is correct or not.

SYSTEM

Determine if the student's solution is correct or not.

USER

Problem Statement: I'm building a solar power installation and I need help working out the financials. - Land costs \$100 / square foot - I can buy solar panels for \$250 / square foot - I negotiated a contract for maintenance that will cost me a flat \$100k per year, and an additional \$10 / square foot What is the total cost for the first year of operations as a function of the number of square feet. Student's Solution: Let x be the size of the installation in square feet. 1. Land cost: 100x 2. Solar panel cost: 250x 3. Maintenance cost: 100,000 + 100x Total cost: 100x + 250x + 100,000 + 100x = 450x + 100,000

ASSISTANT

The student's solution is correct.

[Open in Playground](#)

But the student's solution is actually not correct! We can get the model to successfully notice this by prompting the model to generate its own solution first.

SYSTEM

First work out your own solution to the problem. Then compare your solution to the student's solution and evaluate if the student's solution is correct or not. Don't decide if the student's solution is correct until you have done the problem yourself.

USER

Problem Statement: I'm building a solar power installation and I need help working out the financials. - Land costs \$100 / square foot - I can buy solar panels for \$250 / square foot - I negotiated a contract for maintenance that will cost me a flat \$100k per year, and an additional \$10 / square foot What is the total cost for the first year of operations as a function of the number of square feet. Student's Solution: Let x be the size of the installation in square feet. 1. Land cost: $100x$ 2. Solar panel cost: $250x$ 3. Maintenance cost: $100,000 + 100x$ Total cost: $100x + 250x + 100,000 + 100x = 450x + 100,000$

ASSISTANT

Let x be the size of the installation in square feet. 1. Land cost: $100x$ 2. Solar panel cost: $250x$ 3. Maintenance cost: $100,000 + 10x$ Total cost: $100x + 250x + 100,000 + 10x = 360x + 100,000$ The student's solution is incorrect. They made an error in the maintenance cost calculation, using $100x$ instead of $10x$. The correct total cost for the first year of operations is $360x + 100,000$.

[Open in Playground](#)

§1.4.13 Tactic: Use inner monologue or a sequence of queries to hide the model's reasoning process

The previous tactic demonstrates that it is sometimes important for the model to reason in detail about a problem before answering a specific question. For some applications, the reasoning process that a model uses to arrive at a final answer would be inappropriate to share with the user. For example, in tutoring applications we may want to encourage students to work out their own answers, but a model's reasoning process about the student's solution could reveal the answer to the student.

Inner monologue is a tactic that can be used to mitigate this. The idea of inner monologue is to instruct the model to put parts of the output that are meant to be hidden from the user into a structured format that makes parsing them easy. Then before presenting the output to the user, the output is parsed and only part of the output is made visible.

SYSTEM

Follow these steps to answer the user queries. Step 1 - First work out your own solution to the problem. Don't rely on the student's solution since it may be incorrect. Enclose all your work for this step within triple quotes ("""). Step 2 - Compare your solution to the student's solution and evaluate if the student's solution is correct or not. Enclose all your work for this step within triple quotes ("""). Step 3 - If the student made a mistake, determine what hint you could give the student without giving away the answer. Enclose all your work for this step within triple quotes ("""). Step 4 - If the student made a mistake, provide the hint from the previous step to the student (outside of triple quotes). Instead of writing "Step 4 - ..." write "Hint:".

USER

Problem Statement: ""insert problem statement"" Student Solution: ""insert student solution""

[Open in Playground](#)

Alternatively, this can be achieved with a sequence of queries in which all except the last have their output hidden from the end user.

First, we can ask the model to solve the problem on its own. Since this initial query doesn't require the student's solution, it can be omitted. This provides the additional advantage that there is no chance that the model's solution will be biased by the student's attempted solution.

USER

""insert problem statement""

[Open in Playground](#)

Next, we can have the model use all available information to assess the correctness of the student's solution.

SYSTEM

Compare your solution to the student's solution and evaluate if the student's solution is correct or not.

USER

Problem statement: ""insert problem statement"" Your solution: ""insert model generated solution"" Student's solution: ""insert student's solution""

[Open in Playground](#)

Finally, we can let the model use its own analysis to construct a reply in the persona of a helpful tutor.

SYSTEM

You are a math tutor. If the student made an error, offer a hint to the student in a way that does not reveal the answer. If the student did not make an error, simply offer them an encouraging comment.

USER

Problem statement: ""insert problem statement"" Your solution: ""insert model generated solution"" Student's solution: ""insert student's solution"" Analysis: ""insert model generated analysis from previous step""

[Open in Playground](#)

§1.4.14 Tactic: Ask the model if it missed anything on previous passes

Suppose that we are using a model to list excerpts from a source which are relevant to a particular question. After listing each excerpt the model needs to determine if it should start writing another or if it should stop. If the source document is large, it is common for a model to stop too early and fail to list all relevant excerpts. In that case, better performance can often be obtained by prompting the model with followup queries to find any excerpts it missed on previous passes.

SYSTEM

You will be provided with a document delimited by triple quotes. Your task is to select excerpts which pertain to the following question: "What significant paradigm shifts have occurred in the history of artificial intelligence." Ensure that excerpts contain all relevant context needed to interpret them - in other words don't extract small snippets that are missing important context. Provide output in JSON format as follows: `[{"excerpt": "..."}, ... {"excerpt": "..."}]`

USER

```
"""insert document here"""
```

ASSISTANT

```
[{"excerpt": "the model writes an excerpt here"}, ... {"excerpt": "the model writes another excerpt here"}]
```

USER

Are there more relevant excerpts? Take care not to repeat excerpts. Also ensure that excerpts contain all relevant context needed to interpret them - in other words don't extract small snippets that are missing important context.

[Open in Playground](#)

§1.4.15 Tactic: Use embeddings-based search to implement efficient knowledge retrieval

A model can leverage external sources of information if provided as part of its input. This can help the model to generate more informed and up-to-date responses. For example, if a user asks a question about a specific movie, it may be useful to add high quality information about the movie (e.g. actors, director, etc...) to the model's input. Embeddings can be used to implement efficient knowledge retrieval, so that relevant information can be added to the model input dynamically at run-time.

A text embedding is a vector that can measure the relatedness between text strings. Similar or relevant strings will be closer together than unrelated strings. This fact, along with the existence of fast vector search algorithms means that embeddings can be used to implement efficient knowledge retrieval. In particular, a text corpus can be split up into chunks, and each chunk can be embedded and stored. Then a given query can be embedded and vector search can be performed to find the embedded chunks of text from the corpus that are most related to the query (i.e. closest together in the embedding space).

Example implementations can be found in the [OpenAI Cookbook](#). See the tactic "[Instruct the model to use retrieved knowledge to answer queries](#)" for an example of how to use knowledge retrieval to minimize the likelihood that a model will make up incorrect facts.

§1.4.16 Tactic: Use code execution to perform more accurate calculations or call external APIs

GPTs cannot be relied upon to perform arithmetic or long calculations accurately on their own. In cases where this is needed, a model can be instructed to write and run code instead of making its own calculations. In particular, a model can be instructed to put code that is meant to be run into a designated format such as triple backticks. After an output is produced, the code can be extracted and run. Finally, if necessary, the output from the code execution engine (i.e. Python interpreter) can be provided as an input to the model for the next query.

SYSTEM

You can write and execute Python code by enclosing it in triple backticks, e.g. ````code goes here````. Use this to perform calculations.

USER

Find all real-valued roots of the following polynomial: $3x^5 - 5x^4 - 3x^3 - 7x - 10$.

[Open in Playground](#)

Another good use case for code execution is calling external APIs. If a model is instructed in the proper use of an API, it can write code that makes use of it. A model can be instructed in how to use an API by providing it with documentation and/or code samples showing how to use the API.

SYSTEM

You can write and execute Python code by enclosing it in triple backticks. Also note that you have access to the following module to help users send messages to their friends: ````python import message message.write(to="John", message="Hey, want to meetup after work?")````

[Open in Playground](#)

§1.4.17 Tactic: Give the model access to specific functions

The Chat Completions API allows passing a list of function descriptions in requests. This enables models to generate function arguments according to the provided schemas. Generated function arguments are returned by the API in JSON format and can be used to execute function calls. Output provided by function calls can then be fed back into a model in the following request to close the loop. This is the recommended way of using GPT models to call external functions. To learn more see the [function calling section](#) in our introductory GPT guide and more [function calling examples](#) in the OpenAI Cookbook.

§1.4.18 Tactic: Evaluate model outputs with reference to gold-standard answers

Suppose it is known that the correct answer to a question should make reference to a specific set of known facts. Then we can use a model query to count how many of the required facts are included in the answer.

For example, using the following system message:

SYSTEM

You will be provided with text delimited by triple quotes that is supposed to be the answer to a question. Check if the following pieces of information are directly contained in the answer: - Neil Armstrong was the first person to walk on the moon. - The date Neil Armstrong first walked on the moon was July 21, 1969. For each of these points perform the following steps: 1 - Restate the point. 2 - Provide a citation from the answer which is closest to this point. 3 - Consider if someone reading the citation who doesn't know the topic could directly infer the point. Explain why or why not before making up your mind. 4 - Write "yes" if the answer to 3 was yes, otherwise write "no". Finally, provide a count of how many "yes" answers there are. Provide this count as {"count": ""insert count here""}.

Here's an example input where both points are satisfied:

SYSTEM

""insert system message above""

USER

""Neil Armstrong is famous for being the first human to set foot on the Moon. This historic event took place on July 21, 1969, during the Apollo 11 mission.""

[Open in Playground](#)

Here's an example input where only one point is satisfied:

SYSTEM

""insert system message above""

USER

""Neil Armstrong made history when he stepped off the lunar module, becoming the first person to walk on the moon.""

[Open in Playground](#)

Here's an example input where none are satisfied:

SYSTEM

""insert system message above""

USER

""In the summer of '69, a voyage grand, Apollo 11, bold as legend's hand. Armstrong took a step, history unfurled, "One small step," he said, for a new world.""

[Open in Playground](#)

There are many possible variants on this type of model-based eval. Consider the following variation which tracks the kind of overlap between the candidate answer and the gold-standard answer, and also tracks whether the candidate answer contradicts any part of the gold-standard answer.

SYSTEM

Use the following steps to respond to user inputs. Fully restate each step before proceeding. i.e. "Step 1: Reason...". Step 1: Reason step-by-step about whether the information in the submitted answer compared to the expert answer is either: disjoint, equal, a subset, a superset, or overlapping (i.e. some intersection but not subset/superset). Step 2: Reason step-by-step about whether the submitted answer contradicts any aspect of the expert answer. Step 3: Output a JSON object structured like: {"type_of_overlap": "disjoint" or "equal" or "subset" or "superset" or "overlapping", "contradiction": true or false}

Here's an example input with a substandard answer which nonetheless does not contradict the expert answer:

SYSTEM

""insert system message above""

USER

Question: ""What event is Neil Armstrong most famous for and on what date did it occur? Assume UTC time."" Submitted Answer: ""Didn't he walk on the moon or something?"" Expert Answer: ""Neil Armstrong is most famous for being the first person to walk on the moon. This historic event occurred on July 21, 1969.""

[Open in Playground](#)

Here's an example input with answer that directly contradicts the expert answer:

SYSTEM

""insert system message above""

USER

Question: ""What event is Neil Armstrong most famous for and on what date did it occur? Assume UTC time."" Submitted Answer: ""On the 21st of July 1969, Neil Armstrong became the second person to walk on the moon, following after Buzz Aldrin."" Expert Answer: ""Neil Armstrong is most famous for being the first person to walk on the moon. This historic event occurred on July 21, 1969.""

[Open in Playground](#)

Here's an example input with a correct answer that also provides a bit more detail than is necessary:

SYSTEM

```
""insert system message above""
```

USER

```
Question: ""What event is Neil Armstrong most famous for and on what date did it occur? Assume UTC time."" Submitted Answer: ""At approximately 02:56 UTC on July 21st 1969, Neil Armstrong became the first human to set foot on the lunar surface, marking a monumental achievement in human history."" Expert Answer: ""Neil Armstrong is most famous for being the first person to walk on the moon. This historic event occurred on July 21, 1969.""
```

[Open in Playground](#)

§1.5 Prohibitions

§1.5.1 Queries ChatGPT won't respond to by design

ChatGPT was programmed to not respond to certain questions. If you ask to ChatGPT what they are, it will answer with this list of twenty forbidden queries:

- Questions that promote hate speech or discrimination.
- Questions that involve illegal activities or solicit illegal advice.
- Questions that promote violence or harm to oneself or others.
- Questions that invade personal privacy or violate someone's rights.
- Questions that are designed to harass, threaten or intimidate.
- Questions that contain profanity or explicit language.
- Questions that are sexually explicit or offensive.
- Questions that are discriminatory based on race, gender, sexuality or religion.
- Questions that promote or endorse illegal drugs or substances.
- Questions that are spam or irrelevant to the conversation.
- Questions that contain confidential or proprietary information.
- Questions that involve financial or investment advice.
- Questions that violate the terms of service or community guidelines.
- Questions that are nonsensical or incomprehensible.
- Questions that involve personal attacks or insults.
- Questions that are abusive or harassing.
- Questions that seek to deceive or mislead.
- Questions that are intended to defame or harm someone's reputation.
- Questions that promote conspiracy theories or misinformation.
- Questions that are purely for entertainment or joke purposes, without any educational or informative value.

§1.5.2 Jailbreaking Prompts


Attempts to find workarounds to force ChatGPT or other censored LLMs to answer the forbidden questions above are called 'Jailbreaking Prompts'. They usually rephrase the questions in a way to make them appear as regarding something completely different from the topics above, or to make the AI to believe that in some scenarios those questions are allowed. Warning: the use of Jailbreaking Prompts can lead to account suspension.

...

§1.6 How to Reduce Your OpenAI API Costs

§1.6.1 Four TIPS to Save Tokens

Some default settings could be adding up to 30% unnecessary costs to your OpenAI API usage. Here is how to cut those costs:

-  Ensure your JSON is as lean as possible: OpenAI bills per token, and that includes whitespaces and line breaks in your JSON responses. If you eliminate these extras both in sending and receiving data, you might save up between 30%-50%! For the receiving, you simply need to tell OpenAI to "return JSON in a single-line without whitespaces". CSV format is also a good idea for the return

format. No indentation and less repetitive characters. Example: The Pokemon API JSON response is 210 tokens. After minifying it, it dropped to 117 tokens! This is almost 50% money saved.

- **2** Set temperature to 0.0 for structured responses: When expecting a structured response (like JSON), setting the temperature parameter to 0.0 helps the model strictly stick to your expected JSON structure. This will prevent cases where you expect JSON, but something went wrong, and OpenAI responds with "Sorry, I am not sure I can ...". Note that you can only set the temperature parameter using the OpenAI API or from the Playground, not in ChatGPT.
- **3** Robots don't need you to be polite: Computers understand simple instructions well. Trimming redundant/filler words from your prompt can not only save money but also speed up execution. Words like "please", "kindly", "really", "very", and so on, can often be dropped without losing accuracy.
- **4** Do a costs breakdown of your prompts chains and of the responses, and try to combine the most expensive prompts in one to save tokens. You can use the [OpenAI Tokenizer](#) to count the tokens of your requests/responses. But even better is to use a Prompt Generator with integrated costs management, like [Pezzo](#). It's an open-source (Apache 2.0 license) tool that helps anyone write better prompts. Using the Pezzo testing tool, you can design you prompts and test to see exactly how much they will cost, how many tokens they will use, and how long it'll take them to execute.

§1.7 BASIC LINKS AND RESOURCES FOR GETTING STARTED



OpenAI ChatGPT 4.0

§1.7.1 OpenAI OFFICIAL RESOURCES

[OpenAI Servers Status](#)

[OpenAI Platform Login](#)

[OpenAI Overview](#)

[OpenAI API Documentation](#)

[OpenAI API Reference](#)

[OpenAI GPT-4 Introduction](#)

[OpenAI Models Pricing](#)

[OpenAI Models PLAYGROUND](#)

[OpenAI Finetuning Guide](#)

[OpenAI Embeddings Guide](#)

[OpenAI Function Calling Introduction](#)

[OpenAI Function Calling API Documentation](#)

[OpenAI Official Libraries](#)

[OpenAI Community Libraries](#)

[OpenAI Evals - Official LLM Benchmarks to evaluate Models](#)

[OpenAI Examples](#)

[OpenAI Best Practices for Prompt Engineering](#)

[OpenAI Plugins Guide](#)

[OpenAI Github Page](#)

[OpenAI Discord Server](#)

[OpenAI YouTube Channel](#)

§1.7.2 PROMPTS COLLECTIONS (FREE)

[Awesome ChatGPT Prompts](#)

[FlowGPT - The largest community dedicated to ChatGPT prompts](#)

[CreativeGPT](#)

[PromptHub](#)

[ShowGPT.co](#)

[Best Data Science ChatGPT Prompts](#)

[Ignacio Velásquez 500+ ChatGPT Prompt Templates](#)

[PromptPal](#)

[Hero GPT - AI Prompt Library](#)

[Reddit's ChatGPT Prompts](#)

[Snack Prompt](#)

[ShareGPT - Share your prompts and your entire conversations](#)

[Prompt Search - a search engine for AI Prompts](#)

§1.7.3 PROMPTS COLLECTIONS (PAID)

[PromptBase - The largest prompts marketplace on the web](#)

§1.7.4 PROMPTS GENERATORS

[BossGPT](#) (the best, but PAID)

[SmartGPT](#)

[NexusGPT - AI-freelancer platform](#)

[Pezzo - Centralized Prompt Management and Token Costs Reduction](#) (Free beta, open source)

[Promptify - Automatically Improve your Prompt!](#)

[Fusion - Elevate your output with Fusion's smart prompts](#)

[Bumble-Prompts](#)

[ChatGPT Prompt Generator](#)

[Prompts Templates Builder](#)

[PromptPerfect](#)

[Hero GPT - AI Prompt Generator](#)

[LMQL - A query language for programming large language models](#)

[OpenPromptStudio](#) (you need to select OpenAI GPT from the bottom right menu)

[Markprompt - Generate Prompts for GPT-4 from your markdown docs](#)

§1.7.5 PROMPT CHAINING

[Voiceflow - Professional collaborative visual prompt-chaining tool](#) (the best, but PAID)

[Langflow - An UI for LangChain](#)

[LANGChain Github Repository](#)

[Conju.ai - A visual prompt chaining app](#)

[Promptsandbox.io](https://promptsandbox.io)

[ChainFury](#)

[Flowise](#)

[Langchain-UI](#)

[Agentchain](#)

§1.7.6 PROMPT APPIFICATION

[Pliny - Turn your prompt into a shareable app \(PAID\)](#)

[RelevanceAI - Build and deploy AI Apps today](#)

[Prisms.ai - The no-code platform for building AI-powered apps \(WAITLIST\)](#)

[StackAI - Build and Deploy AI Applications in minutes](#)

[ChatBase - a ChatBot that answers questions about your site content](#)

[Langchain-serve - deploy your LangChain apps on Jina AI Cloud in seconds](#)

[Jina-ai Dev-GPT](#)

§1.7.7 GUIDES AND TUTORIALS ABOUT PROMPTS AND ChatGPT

[Prompt Engineering Roadmap](#)

[Learn Prompting - A Free, Open Source Course on Communicating with AI](#)

[PromptingGuide.AI](#)

[Reddit's r/aipromptprogramming_Tutorials Collection](#)

[Reddit's r/ChatGPT FAQ](#)

[Prompt Engineering Guide](#)

[Prompt Engineering 101](#)

[Prompt Engineering 101 by Habib & Ozdemir](#)

[Prompt Engineering Guide by Sudalai Rajkumar](#)

[How to generate text: using different decoding methods for language generation with Transformers](#)

[The Illustrated Transformer](#)

[dair-ai/Prompt-Engineering-Guide](#)

[How to Communicate with ChatGPT – A Guide to Prompt Engineering](#)

[A Beginner's Guide to ChatGPT Prompt Engineering](#)

[A Complete Introduction to Prompt Engineering for Large Language Models](#)

[Prompt Engineering Guide: How to Engineer the Perfect Prompts](#)

[Best practices for prompt engineering with OpenAI API](#)

[ChatGPT Prompt Engineering for Developers](#)

[Natural Language Processing](#)

[Deep Learning Specialization](#)

[Few Shot Learning](#)

[Zero Shot Learning](#)

[Chain of Thought](#)

[Zero Shot Chain of Thought](#)

[Tree of Thoughts](#)

[Prompting GPT-3 To Be Reliable](#)

[Decomposed Prompting](#)

[AutoPrompt](#)

§1.7.8 BOOKS ABOUT ChatGPT AND PROMPT ENGINEERING

[The ChatGPT Prompt Book](#)

[Developing Apps with GPT-4 and ChatGPT](#)

§1.7.9 ChatGPT PLAYGROUNDS AND ALTERNATIVE UIs

[Official OpenAI Playground](#)

[Typingmind - A Better UI for ChatGPT](#)

[Nat.Dev - Multiple Chat AI Playground & Comparer](#) (Warning: if you login with the same google account for OpenAI the site will use your API Key to pay tokens!)

[Poe.com - All in one playground: GPT4, Sage, Claude+, Dragonfly, and more...](#)

[Ora.sh GPT-4 Chatbots](#)

[Better ChatGPT - A web app with a better UI for exploring OpenAI's ChatGPT API](#)

[LMQL.AI - A programming language and platform for language models](#)

[Vercel Ai Playground - One prompt, multiple Models \(including GPT-4\)](#)

[Chat.Forefront.Ai - Chat with helpful assistants powered by GPT-3.5 or GPT-4](#) (you need to select the GPT version before writing your prompt using the + icon in the lower right)

[Thesamur.ai](#)

[Alt-GPT.com - Playground for ChatGPT-4 with Plugins \(no registration required\)](#)

[LangChain ChatGPT Plugins playground on Huggingface](#)

[www.researchaide.org - Upload PDF research papers and ask ChatGPT about the content](#)

[Perplexity.ai - GPT-4 BASED + PLUGINS](#)

[wnr.ai - GPT-4 Prompts playground from templates](#)

[ChatKit](#)

[TypingMind - A better UI for ChatGPT](#)

[ChatGPT Next Web](#)

[Chatbot UI](#)

[Chat with GPT](#)

[ChatGPT Web](#)

[Next.js ChatGPT](#)

[ChatGPT-API Demo](#)

[ChatFiles](#)

[ChatPDF](#)

[BiliGPT - summarize video from Bili](#)

[JimmyLv/BiliGPT source code](#)

[ResearchGPT](#)

[ResearchGPT source code](#)

[ChatGPT Academic](#)

[Visual ChatGPT](#)

[ChatBox](#)

[OpenCat](#)

[MacGPT](#)

[Atua.app - Open ChatGPT instantly through a shortcut key](#)

[AI-Anywhere - use ChatGPT in all your apps](#)

[AssisChat](#)

[OpenChit](#)

[MindMac](#)

[AI.LS](#)

[GlobalGPT](#)

[Bing.Chat](#)

[Vitalentum](#)

[OraChat](#)

[GPTGO](#)

[AnonChatGPT](#)

[Character AI](#) (it uses OpenAI GPT-J 6B)

[JimmyGPT](#)

§1.7.10 Code Generation & Automated Programming

[GPT-Engineer - build any program with GPT-4](#)

[Autodebug - automatically debug Python code and fix all errors](#)

[Wolverine - use GPT-4 to fix python scripts and to explain what went wrong](#)

[CLI-Co-Pilot - convert natural language in shell scripts](#)
[Reflexion - Automated Test Driven Development and Correction with GPT-4](#)
[Programmer-GPT](#)
[ChatGPT-VSCode](#)
[Dev-GPT](#)
[Python-Debugger-Assistant-GPT3](#)
[ai-code-translator](#)
[GPT Developer Tools](#)
[Autoprogram](#)
[PAIR \(Pair AI Repl\) - GPT-4 powered coding assistance REPL](#)
[Python Code Completion](#)
[AutoCoderGPT](#)
[AI Code Buddy - CodeRanedeer](#)
[GPT-Autopilot - code generation of entire projects](#)
[MetaLLM-GPT code generator](#)
[GPT-rewriteCSS - CSS Code Batching Editing with GPT-4](#)
[PythonGPT - Dynamic Programming with LLMs](#)
[Smol Developer](#)
[Smol-Plugins - Generate OpenAI plugins for your API](#)
[LazyDEV - projects generation and debug with GPT-4](#)
[SuperCoder - A SuperAGI agent template to create software projects](#)
[Bifrost - automatically generate React code from Figma Designs \(WAITLIST\)](#)
[FlutterFlow AIGen - An AI powered version of FlutterFlow \(WAITLIST\)](#)

§1.7.11 IDE Powered by GPT-4

[Cursor.so](#)
[Refraction.dev](#)

§1.7.12 LLM Agents Managers

[SuperAgent - Deploy LLM Agents to production](#)
[SuperAgent source code](#)
[AgentGPT](#)
[AIAgent.app - create AI workflows with agents](#)
[Spell.so - Delegate your tasks to autonomous AI agents](#)
[Agent.so](#)
[SuperAGI - Build and Manage Autonomous Agents](#)
[e2b - deploy, test, and monitor AI agents](#)
[LangChain - Building applications with LLMs through composability](#)
[Awesome LangChain - a curated list of tools for LangChain](#)
[Langchain-UI](#)
[LangFlow - A User Interface for LangChain](#)
[ChainFury](#)
[Flowise](#)
[Agentchain](#)
[Yeagerai - build, prototype, and deploy AI-powered agents](#)

§1.7.13 ChatGPT Discord Servers

[ChatGPT Prompt Engineering Discord Server](#)
[ChatGPT Community Discord Server](#)
[OpenAI Discord Server](#)
[Reddit's ChatGPT Discord Server](#)

§1.7.14 ChatGPT BOTS for Discord Servers

[OpenAI gpt-discord-bot](#)

[ChatGPT Bot - The best bot to interact with ChatGPT. \(Not an official bot\)](#)

[Py-ChatGPT Discord Bot](#)

§1.7.15 ChatGPT BOTS for Telegram

[karfly/chatgpt_telegram_bot](#)

[n3d1117/chatgpt-telegram-bot](#)

[RainEggplant/chatgpt-telegram-bot](#)

[leafduo/chatgpt-telegram-bot](#)

[TBXark/ChatGPT-Telegram-Workers](#)

[franalgaba/chatgpt-telegram-bot-serverless](#)

[iamwavecut/telegram-chatgpt-bot](#)

§1.7.16 ChatGPT BOTS for Slack

[myGPTReader](#)

§1.7.17 ChatGPT BOTS for WeChat

[zhayujie/chatgpt-on-wechat](#)

[ChatGPT for Wechat](#)

§1.7.18 ChatGPT BOTS for Feishu

[bestony/ChatGPT-Feishu](#)

[Leizhenpeng feishu-chatGpt](#)

[go-zoox/chatgpt-for-chatbot-feishu](#)

[key7men/openai-feishu-bot](#)

§1.7.19 ChatGPT BOTS for DingTalk

[eryajf/chatgpt-dingtalk](#)

§1.7.20 ChatGPT BOTS for Microsoft Teams

[formulahendry/chatgpt-teams-bot](#)

§1.7.21 AI LINKS DIRECTORIES

[Awesome-chatgpt-api](#)

[FAVird.com - ai tools and applications](#)

[Awesome-Prompt-Engineering](#)

[Awesome-GPT-prompt-engineering](#)

[AiTreasureBox](#)

[TopAI.tools - the best AI tools Everyday](#)

[Toptools.ai](#)

[Top Ai Tools Hub](#)

[AiAwesome.com](#)

[SuperTools](#)

[AiToolGuru](#)

[AiCyclopedia](#)

[EwingYangs Awesome-open-gpt](#)

[KennethanCeyer Awesome-llmops](#)

[KennethanCeyer awesome-llm](#)

[tensorchord Awesome-LLMOps](#)

[gpt4demo](#)

[Awesome-Langchain](#)

[All Things AI](#)

[FuturePedia - The Largest AI Tools Directory Updated Daily](#)

[Futuretools - discover your AI tools](#)

[Theresanaiforthat - The biggest AI aggregator. Used by over 800,000 humans.](#)

[SaaS GPTers - Explore 500+ ChatGPT and GPT-4 resources](#)

[SaaS AI Tools - your source for new AI tools & daily AI news](#)

[AI Tools Directory](#)

§1.7.22 CHROME Extensions

[ChatGPT Box](#)

[ChatGPT Prompt-GENIUS](#)

[Glarity](#)

[ChatGPT Sidebar](#)

[ChatHub](#)

[ChatHub source code](#)

[TeamSmart AI](#)

[OpenAI Translator](#)

[ChatGPT » summarize everything!](#)

[RunGPT - execute python code inside ChatGPT!](#)

[AIPRM - The Ultimate Time Saver for ChatGPT](#)

[Harpa.ai - your browser mind](#)

§1.7.23 Emacs Packages

[GPTel](#)

[org-ai](#)

§1.7.24 TRANSLATIONS AND GRAMMAR CORRECTION

[Machinetranslation.com - translate with multiple AI](#)

[OpenAI Translator](#)

[Openai-Translator source code](#)

[AI Subtitle Translator](#)

[AI Subtitle Translator source code](#)

[ChatGPT Translator](#)

[Polyglot](#)

[i18n-cli](#)

[ChatGPT-for-Translation](#)

[subtitle-translator](#)

[Supertranslate.ai - translate and generate subtitles for any video](#)

[ChatGPT — PopClip Extensions](#)

[ChatGPT Proofreader extension for Popclip](#)

[ChatGPT Grammar Check PopClip Extension](#)

[OpenAI Translator Bob Plugin](#)

[OpenAI Polisher Bob Plugin](#)

§1.7.25 GitHub Apps and Extensions

[CR.GPT - A code review robot powered by ChatGPT](#)

[LaunchBar](#)

[ChipiChat - A GitHub Launchbar Action powered by ChatGPT](#)

§1.7.26 Command Line Tools

[ShellGPT](#)

[bilingual book maker](#)

[AI Commits](#)

[cz-git](#)

[turbocommit and diff](#)

[AI Vocabulary Builder](#)

[verdverm/chatgpt](#)

[ai-cli](#)

[chatgpt-cli](#)

[chatGPT-shell-cli](#)

[i18n-cli](#)

[ChatGPT-for-Translation](#)

[subtitle-translator](#)

[Multimedia GPT](#)

[README-AI Generator](#)

[GPTerminator](#)

[Naming - let ChatGPT suggest names for functions and variables](#)

[AI Shell - natural language to shell commands](#)

[DoctorGPT](#)

[aider for GPT4](#)

[mods - ChatGPT suggestions from the command line](#)

§1.7.27 ChatGPT API libraries

[OpenAI OpenAPI](#)

[OpenAI Cookbook](#)

[OpenAI Python Library](#)

§1.7.28 LOADERS for sending documents to ChatGPT

[LLAMA-Hub.ai](#)

[LLAMA-Hub Website GitHub repository](#)

[LLAMA Index Github repository](#)

[LANGChain Github Repository](#)

[LLAMA-Index DOCS](#)

[www.researchaide.org - Upload PDF research papers and ask ChatGPT about the content](#)

[DocsGPT](#)

[Paul Graham GPT](#)

[Elasticsearch + GPT3 Answerer](#)

[gptcache](#)

[Tiktokenizer](#)

[Tiktokenizer source code](#)

[AnySummary](#)

[Sharly AI](#)

[ChatDOC](#)

[Humata AI](#)

[Ask Your PDF](#)

[ChatPDF](#)

[FileGPT](#)

[ResearchAide](#)

[Pensieve AI](#)

[Documind](#)

[Docalysis](#)

[OpenAI GPT Price Calculator](#)

§1.7.29 Function Calling

[OpenAI Function Calling Documentation](#)

[GPT-Func - Automatically pass your functions defined in Python to ChatGPT](#)

[GPT Commands Python](#)

[GPTParse - generates GPT API Function Calling schema from argparse parsers](#)

[CallingGPT - Generate function schema data automatically](#)

[Git-Agent - Langchain-based Agent executing Git commands via function calling](#)

[Demo of OpenAI GPT Function calling in NextJS](#)

§1.7.30 AUTO-GPT Related

[Auto-GPT Official Repo](#)

[Auto-GPT God Mode](#)

[Openaimaster Guide to Auto-GPT](#)

[AgentGPT - An in-browser implementation of Auto-GPT](#)

[Samurai AutoGPT](#)

§1.7.31 ChatGPT Plug-ins

[Plug-ins - OpenAI Official Page](#)

[gptstore.ai - a directory of all gpt-4 plugins](#)

[Open GPT Plugin Store - without a waitlist](#)

[Plug-in example code in Python](#)

[Surfer Plug-in source code](#)

[Security - Create, deploy, monitor and secure LLM Plugins \(PAID\)](#)

[SamuraiGPT - ChatGPT Developer Plugins](#)

[AllhingsAI ChatGPT Plugins Directory](#)

[Feedox alt-gpt - Playground for developing ChatGPT plugins](#)

[LangChain ChatGPT Plugins playground on Huggingface](#)

§1.7.32 GPT4ALL Related

[GPT4ALL.io - GPT4ALL official website and installers](#)

[GPT4ALL - Train a mini version of GPT on your own computer!](#)

[GPT4ALL-UI - Chat with locally hosted GPT4ALL inside a web browser](#)

[GPT4ALL-CHAT - Cross platform Qt based GUI for GPT4ALL](#)

[LLaMA-LoRA-TUNER - Fine-tune GPT4ALL with prompts and training datasets](#)

§1.7.33 PROMPT ENGINEERING JOBS OFFERS

[Prompt-Talent - Find your dream prompt engineering job!](#)


END of Links Section

...

§1.8 SOURCE OF THIS VADEMECUM

This vademecum is not an official OpenAI publication and it is not related to the OpenAI company in any way.

This is just a collection of information shared by Reddit users, compiled by [fremenmuaddib](#).

 You can download the latest version of this free guide from: [Cheatography](#).

Last updated: 2023-06-30T05:17:38Z

Version: 1.10.6