



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Computer Systems and Computation

Exploring GPT's Capabilities in Chess-Puzzles

Master's Thesis

Master's Degree in Software Systems Engineering and Technology

AUTHOR: Albert Gramaje, Borja

Tutor: Ferri Ramírez, César

Cotutor: Hernández Orallo, José

External cotutor: GARCIA PIQUERA, MANUEL

ACADEMIC YEAR: 2022/2023

A mi madre, por su esfuerzo y apoyo en mi educación.

Resumen

Los modelos de lenguaje no han obtenido su popularidad solo por sus capacidades de generación de textos sino también por la capacidad de aprendizaje que tienen. En este trabajo se realiza una exploración de las capacidades resolutivas de los modelos de lenguaje sobre el ajedrez. Este caso de uso nos permite mediante notaciones específicas procesar un problema visual como el ajedrez en uno relativo a texto, pudiendo así realizar diversos experimentos con diversos enfoques de 'prompts'. Conceptos de *'in-context learning'* y *'fine-tuning'* relativo a *'prompts'* serán introducidos y estudiados.

Palabras clave: IA, ML, DL, Generative AI, GPT, Chess.

Resum

Els models de llenguatge no han obtingut la seua popularitat només per les seues capacitats de generació de textos sino també per la capacitat d'aprenentatge que tenen. En aquest treball es realitza una exploració de les capacitats resolutives dels models de llenguatge sobre els escacs. Aquest cas d'ús ens permet mitjançant notacions específiques processar un problema visual com els escacs en un relatiu a text, podent así realitzar diversos experiments amb diversos enfocaments de 'prompt'. Conceptes de *'in-context learning'* i *'fine-tuning'* relatiu a *'prompts'* seran introduïts i estudiats .

Paraules clau: IA, ML, DL, Generative AI, GPT, Chess.

Abstract

These Language Models have not acquired their popularity based only on their text-generation capabilities, but also for the ability of learning they do have. An exploration of these capabilities over chess is carried out. With chess, it allows to process the game as a Natural Language text problem. Using specific chess notations, a visual problem is transformed into a base generated text problem. Analysing its capabilities of reasoning and solving puzzles with different prompt approaches will be the aim of this thesis. *'In-context learning'* and *'fine-tuning'* concepts will be introduced and studied.

Keywords: IA, ML, DL, Generative AI, GPT, Chess.

Content table

List of Figures	VII
List of Tables.....	IX
1. Introduction	1
1.1 Objectives	1
1.2 Structure.....	2
2. Artificial Intelligence.....	3
2.1 Machine Learning (ML)	4
2.1.1 Training	4
2.1.1.1 Supervised Learning	5
2.1.1.2 Unsupervised Learning.....	5
2.1.1.2.1 Reinforcement Learning	6
2.1.1.2.2 Reinforcement Learning from Human Feedback.....	6
2.1.2 Deep Learning	6
2.1.2.1 Artificial Neural Networks	7
2.1.2.1.1 Transformers	8
3. Generative Artificial Intelligence	11
3.1 Language Models (LM).....	11
3.1.1 Large Language Models (LLM)	12
3.1.2 Instruction Tuned LLM	12
3.2 Training	13
3.2.1 Natural Language Processing (NLP).....	13
3.2.1.1 Stopwords.....	14
3.2.1.2 Stemming.....	14
3.2.1.3 Lemmatization	14
3.2.1.4 Tokenization	15
3.2.1.5 Word Embeddings	15
3.2.1.6 Temperature	17
3.3 Inference.....	18
3.3.1 Prompts	19
3.3.2 Iterative Prompt Development.....	19
3.4 GPT Language Models.....	19
3.4.1 Models.....	20

3.4.1.1	‘GPT-3.5-turbo’ Language Model	20
3.4.1.2	‘GPT-4’ Language Model.....	24
3.4.2	In-context Learning	25
3.4.2.1	Zero, One and Few-Shot Learning	26
3.4.3	ChatML	27
3.4.3.1	Roles.....	27
3.4.3.1.1	System	28
3.4.3.1.2	User	28
3.4.3.1.3	Assistant.....	28
3.4.4	Moderation	29
3.5	Fine-tuning.....	29
3.6	Language Model Applications.....	30
4.	Artificial Intelligence for Chess.....	31
4.1	Chess Engines	31
5.	Experiments	33
5.1	Chess to Natural Language Problem.....	33
5.1.1	Forsyth-Edwards Notation (FEN).....	33
5.1.2	Universal Chess Interface (UCI).....	34
5.2	‘python-chess’ Library	34
5.3	Evaluation	35
5.4	Data retrieval and processing.....	36
5.4.1	Retrieval.....	36
5.4.2	Processing.....	36
5.5	Experiment set-up	37
5.6	Prompt design.....	39
5.6.1	In-context learning.....	42
6.	Results.....	43
6.1	Experiments outputs format	43
6.2	Accuracies.....	43
6.3	Sub-experiments results.....	44
6.3.1	<i>GPT-3.5-turbo</i> results	46
6.3.2	<i>Ft-GPT-3.5-turbo</i> results	46
6.3.3	<i>GPT-4</i> results.....	47
6.3.4	Further observations.....	47
7.	Conclusions	49
8.	References	51

List of figures

Figure 1. Artificial Intelligence vs Machine Learning vs Deep Learning (Anon n.d.-b) ..	4
Figure 2. Graphical representation of a training process in an ANN (Ringa Tech 2021).	5
Figure 3. Comparative table between supervised and unsupervised learning (Kumar 2021).....	6
Figure 4. Simple Artificial Neural Network Diagram, Perceptron (Rowe n.d.)	7
Figure 5. Complex Artificial Multi-Layer Neural Network Diagram (Team 2023)	8
Figure 6. Recurrent vs Feed-Forward Neural Network (Donges 2023)	9
Figure 7. Self-attention mechanism output diagram visualisation (Anon 2023f)	9
Figure 8. Self-attention mechanism output diagram visualization (Anon n.d.-e).....	10
Figure 9. Output example from an input to a base LLM (Anon n.d.-c)	12
Figure 10. Output example on a base LLM, with no capabilities to answer questions (Anon n.d.-c).	12
Figure 11. Output example from an instruction tuned LLM answering questions (Anon n.d.-c).....	13
Figure 12. Stemming example in python with nltk corpus python library.	14
Figure 13. Tokenize sentence example in python with nltk corpus python library.	15
Figure 14. Load pretrained Word2Vec model for purposed showcases.....	16
Figure 15. Word embeddings of the 'king' word.....	16
Figure 16. Output of the words that are most like the 'king' word.....	16
Figure 17. 3D Visualization of the king and most similar words to it.	17
Figure 18. Graphical explanation of the temperature attribute (Anon n.d.-c).....	18
Figure 19. Iterative Prompt Development graphical explanation (Anon n.d.-c).	19
Figure 20. Graphical representation of how OpenAI RLHF works (Gokultechninza 2023b)	21
Figure 21. Comparative of hallucinations with and without RLHF (OpenAI 2023b).....	21
Figure 22. GPT-3 training dataset composition (Anon 2023d).	22
Figure 23. Brief descriptions of each model of GPT-3.5 series (OpenAI 2023b).....	22
Figure 24. Evolution from GPT-3 series into GPT-3.5 series (Fu, Peng, and Khot 2022).	22
Figure 25. Brief explanation of both GPT-3.5-Turbo models (OpenAI 2023b).....	23
Figure 26. GPT-4 parameters comparison with other LLM (Kerner 2023)	24
Figure 27. GPT-4 token pricing (OpenAI 2023b).	25
Figure 28. GPT-3.5-turbo token pricing (OpenAI 2023b).....	25

Figure 29. Zero, One, Few-shot vs Fine-tuning (Brown et al. 2020)	26
Figure 30. Showcase of roles in GPT-3.5-turbo with ChatML roles	27
Figure 31. Diagram of roles in GPT-3.5-turbo with ChatML roles (Anon n.d.-c)	28
Figure 32. Showcase of ChatGPT response to harmful content	29
Figure 33. FEN notation example (Feng et al. 2023a)	33
Figure 34. UCI moves notation example (Feng et al. 2023a)	34
Figure 35. python-chess library showcases example (Fiekas 2023)	35
Figure 36. python-chess library showcase example (Fiekas 2023)	35
Figure 37. Initial prompt iteration	39
Figure 38. Second iteration of prompt	40
Figure 39. Third and final iteration of prompt	41
Figure 40. GPT error log during execution	41

List of tables

Table 1. GPT LLM experiment setup	37
Table 2. ICL sub-experiment with GPT-3.5-turbo and temp 0	38
Table 3. ICL sub-experiment with GPT-3.5-turbo and temp 1	38
Table 4. ICL sub-experiment with fine-tuned GPT-3.5-turbo and temp 0	38
Table 5. ICL sub-experiment with fine-tuned GPT-3.5-turbo and temp 1	38
Table 6. ICL sub-experiment with GPT-4 and temp 0	38
Table 7. ICL sub-experiment with GPT-4 and temp 1	38
Table 8. Output format of each sub-experiment	43
Table 9. ICL sub-experiment results with GPT-3.5-turbo and temp 0	44
Table 10. ICL sub-experiment results with GPT-3.5-turbo and temp 1	45
Table 11. ICL sub-experiment results with fine-tuned GPT-3.5-turbo and temp 0	45
Table 12. ICL sub-experiment results with fine-tuned GPT-3.5-turbo and temp 1	45
Table 13. ICL sub-experiment results with GPT-4 and temp 0	45
Table 14. ICL sub-experiment results with GPT-4 and temp 1	45
Table 15. Number of matching moves per sub-experiment	48

1. Introduction

Nowadays Artificial Intelligence (*AI*) has achieved a prominent role in various domains of our society, transforming the way we interact with technology and opening new opportunities for development and innovation of newer products and services. One of the most fascinating aspects of *AI* is its generative capability, known as ‘*Generative Artificial Intelligence*’.

‘Artificial Intelligence is the theory and development of computer systems able to perform tasks normally requiring human intelligence (Qwiklabs n.d.-a).

Generative Artificial Intelligence refers to the ability of *AI* systems to produce new content, whether in the form of text, images, or music from some input. These inputs are also referenced by the term ‘*prompt*’. A prompt is a short piece of text that is given to the system and can be used to control the given output from the system without modifying its natural behaviour (Qwiklabs n.d.-a).

There are several systems of Generative Artificial Intelligence (*GenAI*), such as generative models of images from text (text-to-image); generative models of videos from images (image-to-video); generative models of text from text (text-to-text) known as Language Models.

Language Models (*LM*) are statistical tools that anticipate the next word(s) in a sequence by computing the probability distribution of a given sequence. Language models are typically trained on a large dataset with data from various sources such as books, news articles, web pages and so on (Qwiklabs n.d.-b).

Among Generative Language Models, Generative Pretrained Transformer (*GPT*) Language Models, developed by *OpenAI*, has shown an exceptional performance and results. The model uses Neural Networks and large datasets to learn linguistic patterns and generate coherent and relevant text in response to a given prompt.

GPT LLM’s have gained worldwide recognition this year, showcasing diverse applications through prompts and significantly influencing daily life. In this thesis, an experiment over these models is carried exploring its own capabilities over an abstract concept like chess. Chess puzzles will be supplied targeting a potential next move leading to a checkmate. Their *in-context learning* will be tested with different conversational input prompts. A fine-tuning process over one of these *LLM*’s will be done as well. A comparison with the effectiveness of both approaches with different *GPT*’s will be studied.

1.1 Objectives

An introduction Generative Artificial Intelligence (*GenAI*) field, over the text-generative field with Language Models (*LM*) and Large Language Models (*LLM*) is carried. Furthermore, this thesis aims to study of capacity and limitations of these models over an abstract concept like chess.

Experiments will be conducted over these *LLMs* in a non-Natural Language use case, like chess puzzles. Different prompt techniques will be supplied to the model, including concepts of *fine-tuning* and *in-context learning*. In-context learning refers to the ability of the model to learn given a conversational dataset into him; without affecting to the natural behaviour of the models. In contrast of fine-tuning, that does affect this natural behaviour.

Fine-tuning OpenAI's *LLMs* is available from September 2023. An experiment setup using both *GPT-3.5-turbo* and *GPT-4* base models and *GPT-3.5-turbo* fine-tuned version is done, analysing learning, and solving capabilities over chess puzzles.

1.2 Structure

For the consecution of the proposed objectives above, a brief explanation in Chapter 2 introducing key concepts on *AI* such as *AI* itself, Machine Learning (*ML*) and Deep Learning (*DL*); the latter is particularly described when it is related with Artificial Neural Networks (*ANN*).

Additionally, in Chapter 3, a brief explanation of Generative Artificial Intelligence (*GenAI*) will be introduced among its key concepts like Language Models (*LM*). An explanation of what are *LM*, and its training procedure will be briefly explained in Section 3.1. In Section 3.2, different Natural Language Processing techniques required for a model to understand human natural language will be explained including Tokenization, Lemmatization, and Embeddings.

In Section 3.3, an explanation of the inferring concept and how to infer into *LMs* via prompts will be carried out. In Section 3.4, the evolution of OpenAI language models will be briefly covered, with a focus on the *GPT* dialog chat *LLMs*.

Over Chapter 4, a brief introduction of chess on the *AI* field is introduced. Chess engines will be briefly defined and explained. Some history of this engines will be exposed.

Chapter 5 includes all the experiments design. Since the definition of each of experiments and sub-experiment to the chess notation used. These notations transform a chess game in a text generation problem. Prompt design process is also detailed, with different iterations done.

These experiments consist of a series of chess puzzles, finding the move that produces a checkmate situation over it.

Results section will be Chapter 6, where results are shown and explained. An analysis of the results given per each model is done, with an addition of some further observations that could be made over these results.

Chapter 7 concludes the work, summarizing results given and briefly explained why of these results, including a future work related to it.

2. Artificial Intelligence

The advancement of technologies in terms of computational capacity, progress and development of new algorithms are bringing the concept of *AI* to a greater use in our daily lives. There are many examples using *AI* such as the development of autonomous vehicles, data analysis; for predictions, or the generation of smart chatbots.

One of the fields that have gained more popularity during this year over other *AI* topics has been the branch of Generative Artificial Intelligence: through the release of language models such as Generative Pretrained Transformer (*GPT*) Large Language Models (*LLM*).

Building machines with the ability to reason, learn, and behave in ways that would need human intelligence, or involving data beyond what people can examine is the focus of the study of *AI*. *AI* is a broad field that includes many different topics; computer science, data analysis and statistics, and software engineering (Anon 2023h).

But *AI* is not the only main concept involved in this scenario; other concepts are involved in this field. However, there has been a certain lack of knowledge about the different concepts and their relationships among them. The three key concepts are Deep Learning (*DL*), *ML*, and *AI* itself (Oracle 2018a).

When discussing *AI* and its purpose, various learning algorithm techniques are applied. These techniques include a learning process; empowering the system to produce outputs like human responses when given specific inputs (Qwiklabs n.d.-a).

These learning techniques belong to the subfield of *ML*, which is one of the key concepts mentioned earlier; they also fall under the umbrella of the *AI* concept. There are various learning techniques. One of these techniques, which has gained popularity, efficiency, and power, is *DL* (Oracle 2018a).

DL is an *ML* technique. It's based on Artificial Neural Networks (*ANN*); which will be explained later. Its main goal is to replicate human learning. These techniques require prior training with specific datasets; this trains the *AI* system using the provided algorithms. There are various types of learning techniques; further on explained (Anon n.d.-f; Oracle 2018a).

In summary, *AI* pertains to algorithms grounded in *ML*, facilitating self-directed and automated learning, employed for diverse purposes (Alonso 2023). This leads to *AI* being the overarching notion encompassing *ML*. Moreover, the notion of *DL*, being a segment of *ML*; also encompassed within the *AI* framework.

To sum up:

$$AI \supset ML \supset DL$$

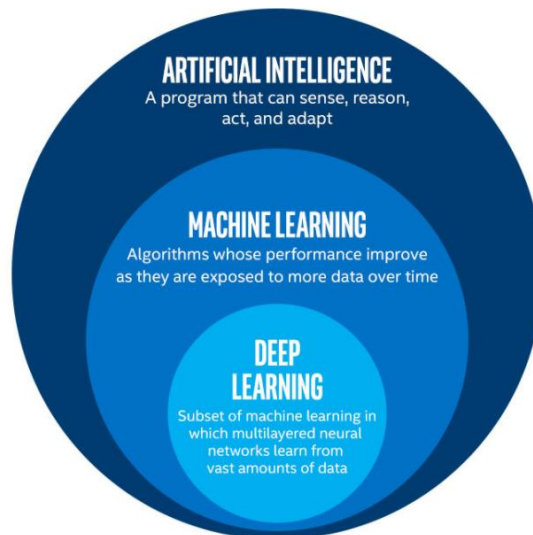


Figure 1. Artificial Intelligence vs Machine Learning vs Deep Learning (Anon n.d.-b)

Collectively, these pivotal ideas are interconnected and mutually enhance each other, propelling progress in crafting intelligent systems able to execute tasks autonomously with minimal human intervention (Oracle 2018a).

2.1 Machine Learning (ML)

Machine Learning (*ML*) is a subset of *AI*; it can autonomously learn (hence the term "Machine Learning"). The goal is to create a model where human intervention is minimized, if not eliminated altogether; producing accurate answers/outputs closely aligned with a given context. In essence, these models can "learn" from data patterns without explicit human guidance (Anon 2023c; Oracle 2018b).

Rather than being explicitly programmed to perform a specific task, it learns automatically through exposure to data and identification of patterns and relationships in it (Anon 2023g).

Machine learning encompasses different training approaches, such as Supervised Learning, Unsupervised Learning, Reinforcement Learning (*RL*) or Reinforcement Learning from Human Feedback (*RLHF*) and so on. The aim of each training approach depends on the kind of model we are trying to achieve and the purpose of it (Copeland 2016).

2.1.1 Training

Training is necessary for a model to learn from data and perform specific tasks. It enables the model to acquire knowledge finding patterns and relationships from a set of training data; allowing to make predictions or decisions on new data given (GraphEverywhere 2019).

Notably, training stands as a pivotal *ML* phase; it significantly shapes the model's performance and its capacity to extrapolate from fresh data. Effective training hinges on a comprehensive and enriched dataset, suitable algorithmic and methodological

selections, and adept tuning of the model's hyperparameters (parameters preset prior to training, like learning rate, batch size, or epoch count) (Delua 2021).

Training is over and finished when the learning curve of the model remains constant. An example of how the learning curve of a sample model looks like can be seen in Figure 2.

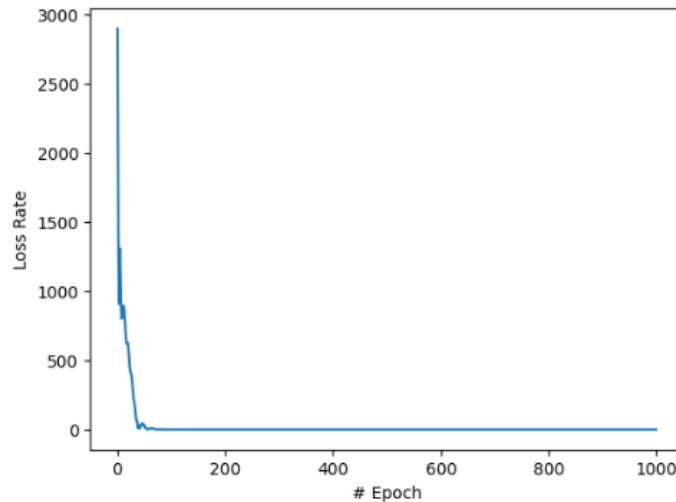


Figure 2. Graphical representation of a training process in an ANN (Ringa Tech 2021).

The image depicted above illustrates the training process of a simple ANN (aimed at learning the conversion of Fahrenheit to Celsius degrees, utilizing a labelled dataset). At Epoch 40 (a full iteration over the model's training data), the learning curve plateaus. This signifies the model's training process concludes at this point, rendering additional epochs unnecessary (Delua 2021).

2.1.1.1 Supervised Learning

Supervised Learning entails an algorithm furnished with a dataset composed of labelled instances presented as pairs of {data, label}. These labels signify the desired output the model should generate when given an input, thus supervising the necessary output corresponding to the given input (Delua 2021)

The concept behind this learning technique involves deriving insights from past examples to predict future values; examining relationships within this labelled data and enable the system to make decisions autonomously (Delua 2021).

An instance of a model utilizing a Supervised Learning approach could involve categorizing images, like animal pictures, by labelling them as dog, cat, and more. The model would return us an output (tag) derived from the probabilities associated with all feasible tags presented to the model during the training stage.

2.1.1.2 Unsupervised Learning

In Unsupervised Learning, the algorithm receives a dataset of non-labelled examples. The essence of this learning technique lies in examining raw data, analysing it (spotting patterns), and observing if it naturally organizes into clusters (Delua 2021). The

algorithms autonomously unveil hidden patterns and data groupings, independent of human involvement.

The primary distinction between Unsupervised and Supervised Learning, lies in the absence of a labelled dataset. The algorithm independently provides an output based on what the model has received during its training phase.

A comparative image between both the Supervised Learning and Unsupervised Learning based on the input data, accuracy, and complexity can be seen in Figure 3.

Type	Supervised	Unsupervised
Input Data	Labelled	Unlabelled
Computational Complexity	Simpler	Computationally Complex
Accuracy	Higher	Lesser

Figure 3. Comparative table between supervised and unsupervised learning (Kumar 2021)

2.1.1.2.1 Reinforcement Learning

Reinforcement Learning (*RL*) forms a subset of *ML*; it revolves around an agent mastering interactions with an environment to maximize cumulative rewards. The agent learns by taking actions within the environment and receiving feedback in the form of rewards or punishments. Consequently, the agent fine-tunes itself, guided by these rewards or punishments, to produce the anticipated output, aligning with the objectives set by *AI* system developers (Bajaj 2018)

2.1.1.2.2 Reinforcement Learning from Human Feedback

Reinforcement Learning from Human Feedback (*RLHF*) constitutes a *ML* strategy that merges *RL* techniques—rewards and comparisons—with human direction to educate an *AI* agent. The goal of *RLHF* is to elevate the effectiveness and aptitude of machine learning algorithms by integrating human expertise and knowledge (Bajaj 2018; Mandour 2023)

This approach strives to address the limitations of Supervised Learning and Unsupervised Learning, in which machine learning algorithms exclusively learn from labelled or unlabelled data (Mandour 2023)

2.1.2 Deep Learning

Deep Learning (*DL*) constitutes a subset of *ML*. It relies on utilizing Artificial Neural Networks (*ANN*), which will be explained in the upcoming section, to simulate the cognitive processes of the human brain. This enables *DL* to tackle more intricate patterns beyond the capabilities of traditional *ML* algorithms (Delua 2021)

DL algorithms emerged with the intention of increasing the effectiveness of traditional ML techniques. Under traditional machine learning methods, significant human effort is required to train the software.

2.1.2.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are made up of collection of layers of nodes (*neurons*). There are three types of layers, the input layer, the hidden/s layer, and the output layer. An ANN consists of at least three layers, including the input/output layer and at least one hidden layer (Anon 2023i). An example of a simple representation, also called perceptron, can be found in Figure 4.

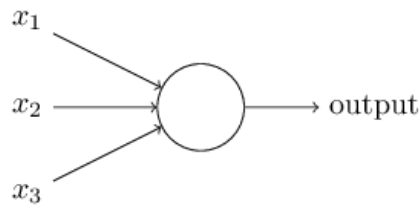


Figure 4. Simple Artificial Neural Network Diagram, Perceptron (Rowe n.d.)

Input layer stands for the information from the outside that is given as input to the ANN. *Output layer* is the result of all the data processed by the ANN. *Hidden layer/s* are layers that takes the input from the input layer or from previous hidden layer (Anon 2023i).

Each layer is composed or built up with several artificial neurons. The design of artificial neurons was inspired by real biological neurons. Our neurons processes environment data or stimulations and sends electrical pulses to other neurons, producing an output being a human reaction. Artificial neurons tend to do the same behaviour as human neurons (Rowe n.d.).

Each artificial neuron is built up with its inputs, outputs and two private variables, an activation function; that produces the output of the artificial neuron, and a threshold. If the result of the activation function is greater than the threshold, then the neuron gets activated and sends its result to the next hidden layer (Team 2023).

A diagram of how layers are interconnected, and a graphical representation of a complex multi-layer network can be seen in Figure 5. As the layers progress, each neuron in each layer can make decisions at a more intricate and abstract level, compared to the neurons in preceding layers. This is because it considers the output of the previous neurons as its input (Team 2023).

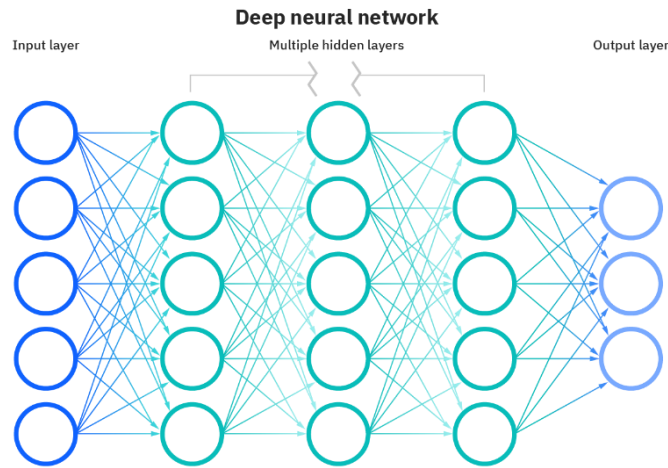


Figure 5. Complex Artificial Multi-Layer Neural Network Diagram (Team 2023)

Each arrow that connects two nodes (artificial neurons) represents a connection. These connections hold associated numerical values, referred to as *weights*. A weight signifies the strength of the connection between the neurons. Both weights and thresholds are computed by the neural network during the training phase, assigning values that best align with the network's operations (Rowe n.d.; Team 2023).

The input of each neuron in its respective hidden layer consists of the addition of every input node from the previous layer times the respective weight value of the connection. The following formula represents the calculation.

$$Z = \sum_i X_i \times W_i$$

Among all activation functions that can be used, one to mention and has been widely used in artificial neural networks is *ReLU*. ReLU stands for "Rectified Linear Unit".

Since using this activation formula, a certain number of neurons are activated, the function is computationally more efficient than other functions (BerenLuthien 2016; DaemonMaker 2014; RockTheStar 2019).

2.1.2.1.1 Transformers

Normal or Feed-Forward Neural Networks (which progress unidirectionally from the input layer through hidden layers to the output layer) lack any recollection of previous inputs as they solely focus on the current input. This makes such networks inadequate for forecasting forthcoming elements based solely on the current input; a necessity to retain contextual information arises (Donges 2023).

In the history of *AI*, various methods have emerged to "retain" or maintain data context. One noteworthy approach is Recurrent Neural Networks (*RNNs*), which operate with an information "loop." Neurons in an *RNN* consider both the current input and the knowledge accumulated from preceding inputs when making decisions (Donges 2023). A diagram illustrating the distinction between feed-forward and recurrent architectures can be observed in Figure 6.

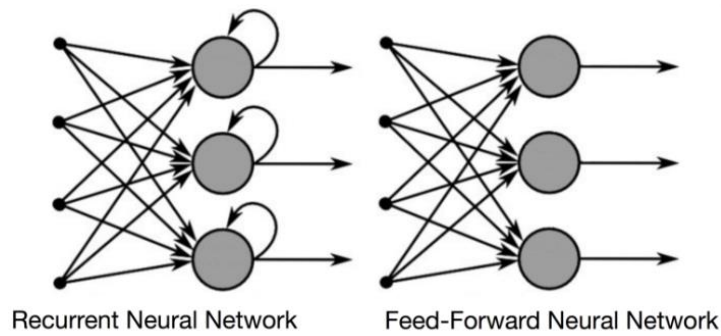


Figure 6. Recurrent vs Feed-Forward Neural Network (Donges 2023)

While this network type addresses context preservation, it carries a drawback: a limited short-term memory capability (Donges 2023). In the realm of *RNN* and *NLP* field, the model retains context between words. However, the initial output might not correlate with the conclusion, as these networks fail to monitor the ongoing context of the entire conversation. This underscores the need to sustain context consistently throughout the process.

Regarding Generative Artificial Intelligence, which will be elaborated upon, there arises a necessity to maintain conversational context, necessitating a long-term memory to generate a coherent output root-ed in that context.

To address the short-term memory issue, a 2017 paper titled "*Attention Is All You Need*" presented a solution through an encoder-decoder architecture named *Transformer*, incorporating attention layers. This design aimed to overcome the constraints of *RNNs* in handling sequential data, notably upgrading the processing of natural language text (Vaswani et al. 2017)

Self-attention models facilitate the association of each specific word in the input prompt with other words within the same prompt. This establishes connections between individual words and the entirety of the prompt. This mechanism empowers the model to concentrate on the most pertinent sections of the input sequence for each prediction, in contrast to processing the entire sequence simultaneously (Vaswani et al. 2017). The visual representation of these relationships is depicted in Figure 7.

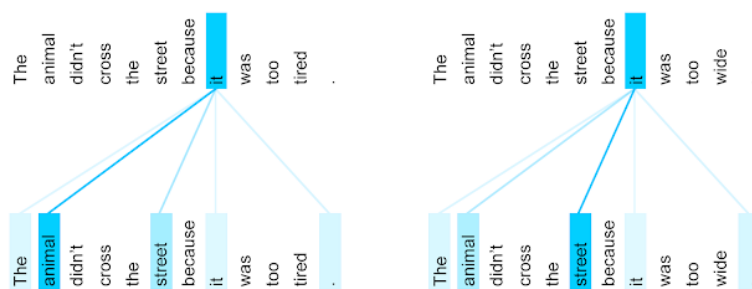


Figure 7. Self-attention mechanism output diagram visualization (Anon 2023f)

The architecture comprises an encoder and a decoder, both consisting of multiple layers of self-attention and feed forward *ANNs*. The encoder handles the input sequence, yielding a series of hidden states. These states are subsequently employed by the decoder to craft the output sequence. Additionally, the decoder employs self-attention to gauge

the significance of the input sequence while producing each output token (Anon 2023f). Figure 8 is a depiction of such model's architecture.

One of the key advantages of the architecture is that it allows for parallel processing of the input sequence, which makes it much faster to train compared to other Neural Networks, such as the recurrent ones. This is because the self-attention mechanism allows each token in the sequence to be processed independently of the others, rather than sequentially (Vaswani et al. 2017)

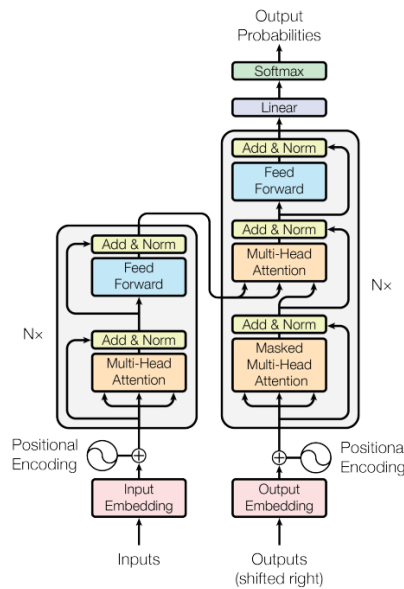


Figure 8. Self-attention mechanism output diagram visualization (Anon n.d.-e)

Generative Artificial Intelligence, as per Language Models (LM) and Large Language Models (LLM) such as GPT LLM are based on the Transformer's architecture explained above.

3. Generative Artificial Intelligence

AI capable of producing novel content spanning text, images, music, and video is referred to as "*Generative Artificial Intelligence*," often abbreviated as "*GenAI*." A recent inclusion in this AI domain comprises the introduction of tools like *ChatGPT* and Google *Bard*. These are AI chatbots enabling users to interact by posing queries, prompts, or questions to a language model and subsequently obtaining a distinctive output generated by the model (Qwiklabs n.d.-a).

GenAI generates fresh content by drawing upon the entirety of existing data that the system has been trained on. It crafts new and distinct outputs using the inputs provided, commonly referred to as *prompts* (Qwiklabs n.d.-a).

It diverges from conventional AI, which focuses on discerning patterns and formulating predictions from preexisting data. *GenAI* models initiate with a prompt, which might be presented as text, an image, a video, or any other processable input within the AI system's scope. *GenAI* employs a statistical model to anticipate a plausible response, thereby generating novel output (Qwiklabs n.d.-a).

The main distinction between *GenAI* and traditional AI is the nature of the model's output. If the output is a numeric value, a probability score, or a predefined class, it signifies a traditional AI model. Conversely, if the output takes the form of a natural language response, a video, an audio clip, or similar content, the model falls under the category of *GenAI* (Qwiklabs n.d.-a).

The significance of the *GenAI* domain extends beyond its individual applications. It brings us closer to a point where communication with computers occurs in natural language, bridging the gap between human communication and programming languages.

3.1 Language Models (LM)

As previously detailed, *GenAI* has the capability to process a wide array of input types (*prompts*) and subsequently produce novel responses. Language Models (*LM*) are a prominent subset within the *GenAI* domain that typically generate text-based outputs. Notable examples of such models include *OpenAI's LLMs* like *GPT-3.5-turbo* and *GPT-4*.


“Language models are models that use machine learning (ML) to conduct a probability distribution over words to predict the most likely next word in a sentence based on the previous entry (Kapronczay 2023).”

They are simply statistical tools that anticipate the next word(s) in a sequence by computing the probability distribution of a sequence of words (Kapronczay 2023). They learn about patterns in language through training data.

Beyond *LM*, there exist other categories of *GenAI* models, like image models. These models, when given a textual prompt, generate images in accordance with the input description. An illustration of this kind is the *DALL·E 2* model by OpenAI.

3.1.1 Large Language Models (LLM)

Also, *LM*, but they've undergone a training process using extensive datasets, earning them the label "Large Language Models (*LLM*)". These models possess the ability to generate superior-quality output compared to standard *LM*. This elevated quality can be attributed to the considerably larger and higher-quality data that these models have been exposed to during their training, as opposed to the datasets provided to basic *LM* (Peng et al. 2023a; Qwiklabs n.d.-b)



Once upon a time, there was a unicorn
that lived in a magical forest with
all her unicorn friends

Figure 9. Output example from an input to a base LLM (Anon n.d.-c)

In the depicted image, an instance of input (highlighted in red) and its corresponding output (displayed in black) is illustrated. Given that *LLMs* predict the subsequent word relying on training data, the generated output maintains a correlation with the input supplied (Qwiklabs n.d.-b)

When provided with questions as input, the model could generate new questions linked to the input. However, it's important to note that the model lacks the capability to comprehend intentions deeply, such as providing answers to questions.



What is the capital of France?
What is France's largest city?
What is France's population?
What is the currency of France?

Figure 10. Output example on a base LLM, with no capabilities to answer questions (Anon n.d.-c).

When a question is posed, the expectation is to receive an answer rather than just additional related questions. This is where Instruction-Tuned *LLMs* come into play. These models can be tailored to a particular context, enabling the model to adhere to provided instructions and offer responses aligned with the given scenario. Peng et al. 2023; Qwiklabs n.d.-b).

3.1.2 Instruction Tuned LLM

Tuning refers to the process of adjusting a model to suit a new domain or a customized set of cases, achieved by training the model on fresh data. Instruction tuning, on the other hand, is a technique employed to align *LLMs* with human intentions. This is accomplished by training these models on data designed for following specific instructions. Through instruction tuning, developers can furnish precise directives to the

LLM, guiding its behaviour and tailoring it for specific tasks (Peng et al. 2023a). Essentially, these models are trained to predict responses based on the instructions they've learned and the input they receive.

Instructional tuning encompasses various methods. One approach involves employing Supervised Learning, where a collection of labelled examples is utilized to train the foundational model. Another technique involves the use of *RLHF*, which merges *RL* principles with human guidance to instructively fine-tune the model. This approach combines reinforcement learning techniques with human direction to educate an *AI* system (Peng et al. 2023a)



What is the capital of France?
The capital of France is Paris.

Figure 11. Output example from an instruction tuned *LLM* answering questions (Anon n.d.-c).

In the output presented in the above image, the provided response directly addresses the question posed in the input. Notably, it refrains from generating additional questions linked to the original query. This outcome is achieved through the instruction tuning process.

OpenAI's *GPT LLMs*, including both *GPT-3.5-turbo* and *GPT-4*, undergo instruction tuning to adopt a chatbot-like conversational way. In contrast to earlier OpenAI models that operated primarily as base *LM*, predicting subsequent words based on provided input, these newer iterations are refined to engage in interactive conversations as chatbots (OpenAI 2022, 2023b)

3.2 Training

Language models are usually trained on extensive text datasets encompassing sources like books, news articles, and web pages. Unsupervised Learning serves as the foundational learning technique. Via this approach, the model acquires vocabulary and establishes connections among word concepts. The essence of Unsupervised Learning is to directly examine raw data, analyse it to uncover patterns, and determine if it naturally clusters into groups or relationships among words (Delua 2021; Qwiklabs n.d.-a)

While computers fundamentally use binary code (comprised of 0s and 1s), *LM* can interpret human-written plain text through Natural Language Processing (*NLP*) algorithms (Kerner 2023). These algorithms facilitate the transformation of words into numerical sequences, enabling machines to process data. This conversion process, termed encodings, is an essential concept for developers working on language models.

3.2.1 Natural Language Processing (NLP)

An area of computer science and *AI* called Natural Language Processing (*NLP*) uses computational linguistics and rule-based models of human language to help computers comprehend text (Anon 2023k; Wolff 2020).

NLP empowers computers to identify letters, words, and sentences, subsequently attributing meaning, and comprehension to the information. This capability allows machines to understand natural language in a manner akin to human cognition (Wolff 2020)

NLP encompasses a range of techniques designed to enable machines to process and comprehend natural language text. Fundamental methods include tokenization, lemmatization, stemming, stopwords removal, and word embeddings.


3.2.1.1 Stopwords

Stopwords refer to frequently used words in a language that are often excluded from text, enhancing the accuracy and efficiency of text analysis. Such words encompass articles, prepositions, conjunctions, and other terms that lack substantial individual meaning (GeeksforGeeks 2017).

Frequent occurrences of stopwords in text can escalate the computational intricacy of *NLP* algorithms, resulting in sluggish processing speeds. These words do not contribute significantly to a text's meaning and might even introduce noise or ambiguity into the analysis (GeeksforGeeks 2017).

3.2.1.2 Stemming

Stemming is a pre-processing technique employed *NLP* to truncate a word to its core or root form, referred to as the stem. Stemming algorithms strive to ascertain shared root bases by eliminating word endings or beginnings (Pykes 2023).



```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()

words = ['cooking', 'cooked', 'cooks']
print([stemmer.stem(w) for w in words])
# output: ['cook', 'cook', 'cook']
```

Figure 12. Stemming example in python with nltk corpus python library.

In the depicted image, three distinct words share the same lemma, as these words are derived from a common stem.

3.2.1.3 Lemmatization

Lemmatization serves as a text pre-processing technique within *NLP* that condenses a word to its core or dictionary form, recognized as the lemma. This approach aims to unify various inflected variations of the same word, identifying similarities and enhancing the precision of *NLP* models (Pykes 2023).

In contrast to Stemming, Lemmatization hinges on precisely discerning the intended part-of-speech and meaning of a word within its context. It factors in the neighbouring words of the target word in the sentence. For instance, while stemming could potentially

reduce "programmers" to "program", which could be either a verb or a noun, introducing ambiguity; Lemmatization would consider the context to ascertain whether "programmers" works as a noun or a verb on its context (Pykes 2023).

Stemming algorithms frequently adopt a simplistic approach, rendering them swift and efficient, but not always accurate. In contrast, lemmatization algorithms prioritize correctness over speed and efficiency, yielding meaningful base roots by sacrificing computational speed (Pykes 2023).

3.2.1.4 Tokenization

Tokenization is a fundamental process in *NLP* that involves breaking down a text or sequence into smaller units called tokens. By breaking down the text into tokens, we can analyse and process the data using computational techniques (Gupta 2019; Menzli 2022).

Those tokens involve a range of possibilities, including words, phrases, sentences, or even individual characters, contingent upon the task and demands. In Figure 13, the `nlTK` python library splits the sentence provided into an array of tokens.



```
from nltk.tokenize import word_tokenize
s = '''Good muffins cost $3.88\nin New York.'''
print(word_tokenize(s))
#output: ['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.']
```

Figure 13. Tokenize sentence example in python with `nlTK` corpus python library.

3.2.1.5 Word Embeddings

Given that computers comprehend numerical entities like scalars, vectors, and matrices rather than words, word embeddings offer a technique to transform natural language words into numerical vectors. This transformation enables the application of mathematical and *ML* algorithms to the data (Anon 2023l)

Word embeddings represent a significant advancement in *NLP*, enhancing computers' capacity to comprehend text-based content more effectively. These embeddings encapsulate the semantic essence of words, enabling the establishment of relationships between words sharing similar meanings (Anon 2023l; shristikotaiah 2020).

Various *ANN* models exist to extract word embeddings for specific words, such as "Word2Vec". Word2Vec is a *ANN* model that produces embeddings for the entire vocabulary it encounters during the training phase (Anon 2023l; shristikotaiah 2020).

An illustration of *ANN*, trained on a text corpus, has been presented as a use case. This model, along with its associated training process, is accessible on Kaggle ¹.

¹ 'Google's Trained Word2Vec Model in Python | Kaggle', accessed 8 July 2023, <https://www.kaggle.com/datasets/umbertogriffo/googles-trained-word2vec-model-in-python>.


```

from gensim.models import Word2Vec, KeyedVectors
pretrained_path = "model/GoogleNews-vectors-negative300.bin"

Word2VecModel = KeyedVectors.load_word2vec_format(pretrained_path, binary = True)

```

Figure 14. Load pretrained Word2Vec model for purposed showcases.

The image depicted above illustrates the procedure for loading this pre-trained model from the Kaggle website.

```

pprint(Word2VecModel['king'])
# output
# array([ 1.25976562e-01,  2.97851562e-02,  8.60595703e-03,  1.39648438e-01,
#        -2.56347656e-02, -3.61328125e-02,  1.11816406e-01, -1.98242188e-01,
#         5.12695312e-02,  3.63281250e-01, -2.42187500e-01, -3.02734375e-01,
#        -1.77734375e-01, -2.49023438e-02, -1.67968750e-01, -1.69921875e-01,
#         3.46679688e-02,  5.21850586e-03,  4.63867188e-02,  1.28906250e-01,
#        ...,
#        -2.79296875e-01, -8.59375000e-02,  9.13085938e-02,  2.51953125e-01],
#        dtype=float32)

```

Figure 15. Word embeddings of the 'king' word.

Figure 15 showcases the embeddings for the word 'king.' Each value within the array corresponds to a distinct feature or dimension within the embedding space. In simpler terms, these values encapsulate specific characteristics or attributes of the word.

Embeddings facilitate the utilization of mathematical algorithms for various objectives. One such application involves determining the words most closely associated with a target word. This is achieved by employing cosine similarity between the embedding vector of the target word and the embedding vectors of the entire vocabulary encompassed by the trained model (Barla 2022; shristikotaiah 2020)

Within this model, a function named '*most_similar()*' implements the algorithm explained above. This function furnishes the top ten words that are most akin to the provided word, determined by a probability list spanning from 0 to 1 (Barla 2022).

```

from pprint import pprint

pprint(Word2VecModel.most_similar('king'))
# output
# [('kings', 0.7138045430183411),
#  ('queen', 0.6510956883430481),
#  ('monarch', 0.6413194537162781),
#  ('crown_prince', 0.6204220056533813),
#  ('prince', 0.6159993410110474),
#  ('sultan', 0.5864824056625366),
#  ('ruler', 0.5797567367553711),
#  ('princes', 0.5646552443504333),
#  ('Prince_Paras', 0.5432944297790527),
#  ('throne', 0.5422105193138123)]

```

Figure 16. Output of the words that are most like the 'king' word.

It's feasible to create a visual representation of the present embedding space for the word "king" and its closely related words. Furthermore, it's possible to encounter a comprehensive *Word2Vec* model on the internet ², that has undergone extensive training with a substantial dataset. This model can be employed to infer embeddings for specific words and visualize their corresponding embedding spaces.

Figure 17 depicts the embeddings of the word "king" in a three-dimensional graph visualization, accompanied by the words most closely associated with it. Words positioned closer to the target word hold stronger relationships or similarities to it.

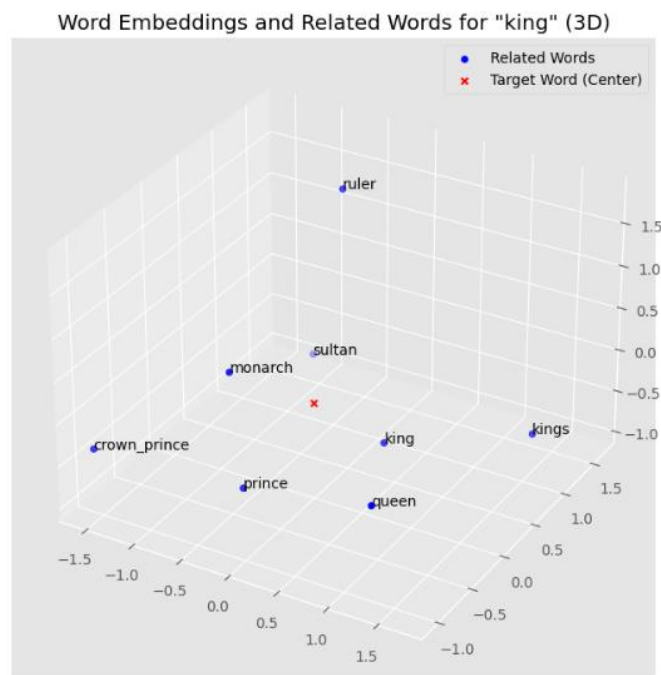


Figure 17. 3D Visualization of the king and most similar words to it.

3.2.1.6 Temperature

Within NLP, temperature (θ) serves as a parameter employed to regulate the extent of randomness and innovation within the output of a language model (Salamone 2021).

This value ranges between 0 and 1. A temperature of 0 outputs the highest probability response computed. Conversely, a temperature of 1 enhances the model's creativity, enabling it to choose words with lower probabilities (Salamone 2021).

Temperature modulation introduces an extra variable θ that impacts the *softmax* distribution. *Softmax* distribution will calculate the probabilities of each target class over all possible target classes. Main advantage of using Softmax is the range of output probabilities. The range will be from 0 to 1, and the sum of all probabilities will be equal to one (Salamone 2021; Vicente n.d.).

² Daniel Smilkov Picture Nikhil Thorat, Charles Nicholson, Big, 'Embedding Projector - Visualization of High-Dimensional Data', accessed 10 July 2023, <http://projector.tensorflow.org>.

A higher temperature essentially "boosts" outputs that had lower probabilities previously, whereas a lower temperature diminishes the relatively smaller outputs compared to the largest ones (Salamone 2021).

Figure 18 provides a visual description of the temperature attribute within a *LM*. As the model endeavours to predict the next word in a sequence, it internally generates a probabilistic distribution among the words it has acquired familiarity with during training. When the temperature attribute is set to zero, the model consistently selects the word with the highest calculated probability. Conversely, when the temperature is set to one, the model may choose a word that doesn't necessarily possess the highest probability among the computed words.

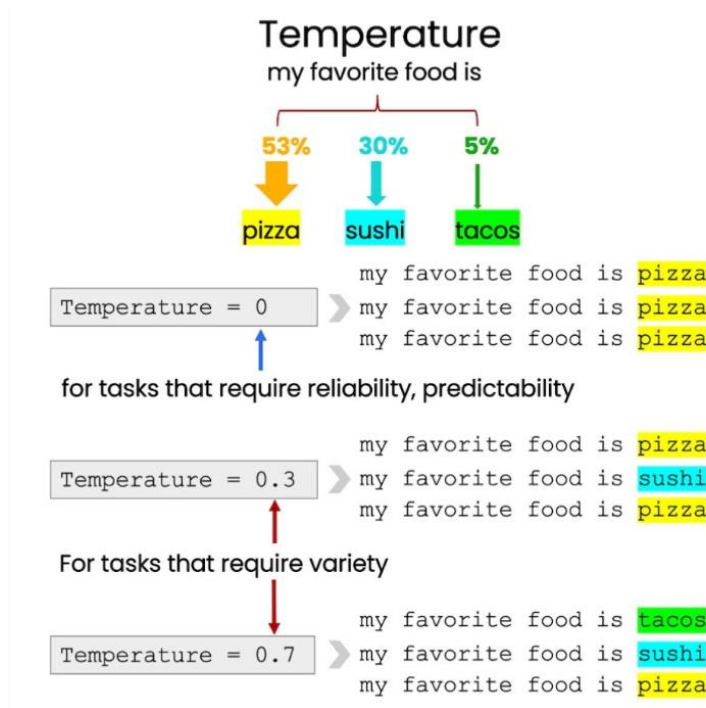


Figure 18. Graphical explanation of the temperature attribute (Anon n.d.-c)

3.3 Inference

In *AI*, inference refers to the process of logical reasoning and decision-making based on available information or data within a model. This procedure involves deriving new insights or conclusions from existing knowledge or data. Inference is of paramount importance in *AI*; it enables machines to engage in rational deduction and decision-making, guided by the information at hand (Gupta 2023; Ltd n.d.).

During the inference phase, a machine leverages the knowledge acquired and stored throughout the training phase to comprehend new data. This stage empowers the machine to recognize fresh data even if it has not encountered it previously (Gupta 2023)

Each system possesses a distinct method of conducting inference, contingent upon its training and intended purpose. In the context of *LM* particularly *LLM*, inference is achieved through the utilization of "prompts".

3.3.1 Prompts

In the realm of *LLM*, a "*prompt*" constitutes a fragment of text employed to steer the model toward a particular task or desired output. Prompts exhibit diverse forms, encompassing descriptions of sought-after outcomes, commands, questions, or succinct sets of instructions (Varshney and Suria 2023).

Prompts serve as instruments for interacting with and inferring from *LLMs* to fulfil specific tasks. They hold the potential to enhance the performance of *LLMs* in addressing the in-tended task. While prompts are effective for refining the model's outputs, they don't modify the model itself. To refine the model itself, a new training process must be undertaken (Tianyu 2021).

Prompt engineering entails the deliberate curation and optimization of prompts to effectively utilize *LLMs*, aiming to extract the optimal outcomes from the model (Tianyu 2021)

3.3.2 Iterative Prompt Development

Iterative Prompt Development encapsulates the process of identifying an ideal prompt tailored for a particular task. Analogous to coding, it involves experimenting with prompts, refining them, and iteratively retrying until the output aligns with the task the model aims to achieve (Kuyper 2023). Figure 19 offers a visual depiction of this concept.

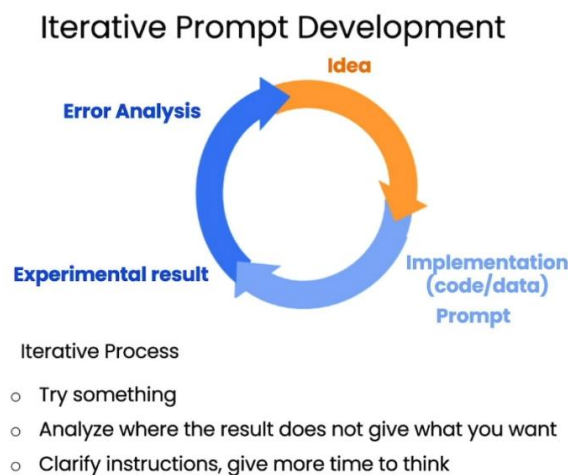


Figure 19. Iterative Prompt Development graphical explanation (Anon n.d.-c).

This process is labour-intensive and demands a significant investment of time. The intricacy of prompts hinges on the kind of desired outcome, necessitating a cycle of generating, refining, and assessing prompts for optimal results (Kuyper 2023)

3.4 GPT Language Models

Generative Pre-trained Transformer (*GPT*), an innovation from *OpenAI*, denotes *LM* employing *DL* algorithms to produce text akin to human language (OpenAI 2023c). It operates on an *ANN* rooted in the *Transformers* architecture, pre-trained using an

extensive corpus of textual data, and can be fine-tuned to tackle distinct *NLP* undertakings (Anon 2023j).

GPT models are designed to generate text that is coherent, grammatically correct, and contextually appropriate. They can be used for a variety of *NLP* tasks such as text completion, summarization, question answering, and language translation (Mandour 2023)

The evolution of *DL* techniques has facilitated the creation of *GPT LLMs*. These models involve training *ANN* with extensive datasets, enabling them to grasp intricate patterns and produce realistic human-like text.

In essence, *GPT LLMs* signify a remarkable breakthrough in the realm of *NLP*. They empower machines to comprehend and generate language with an unprecedented level of fluency and precision.

3.4.1 Models

OpenAI has recently unveiled two enhanced *LLMs*, namely *GPT-3.5-turbo* and *GPT-4*, which have garnered substantial traction in the domain of *GenAI*. These state-of-the-art models belong to the latest generation of *LLMs*, and they have gained substantial usage, especially with the launch of the *ChatGPT* application. It's worth noting that *GPT-3.5-turbo* and *GPT-4* are not the only *LLMs* available. *OpenAI* has progressed through various iterations, culminating in the current *GPT-4* release for now (OpenAI 2022, 2023b).

Well-known *ChatGPT* application leverages both the *GPT-3.5-turbo* and *GPT-4 LLMs*. *GPT-3.5-turbo* is a refined iteration derived from the *GPT-3.5* series (*text-davinci-003*), which interacts in a conversational way. On the other hand, *GPT-4* shares a similar nature with *GPT-3.5-turbo* but emerges from *GPT-3.5* series, signifying a progression from the *GPT-3* iteration (Anon 2023a; OpenAI 2023b).

3.4.1.1 ‘GPT-3.5-turbo’ Language Model

OpenAI introduced *GPT-3.5-turbo* in November 2022, as a refined iteration of the *GPT-3 LLMs* series. This version is meticulously fine-tuned to engage in conversational interactions, resulting in outputs that possess a natural conversational tone. *GPT-3.5-turbo* has gathered significant importance for its efficacy in constructing user-friendly chatbots (Mandour 2023; OpenAI 2022; Ouyang et al. 2022).

This model is a product of a series of advancements built upon previous *LLMs* and has undergone meticulous human-guided fine-tuning. This refinement process uses *RLHF* techniques. *OpenAI*'s *RLHF* approach involves employing base *LLM*, *GPT-3* in this instance, alongside a dataset fine-tuned by *OpenAI* experts. The combination of these elements gives rise to a *LLM* that inherits the capabilities of the base model while adopting a conversational tone in its output. The training process and model development were executed utilizing the substantial computational power of an *Azure AI* supercomputing infrastructure (Anon 2023a; OpenAI 2022; Ouyang et al. 2022; Gokultechninza 2023a).

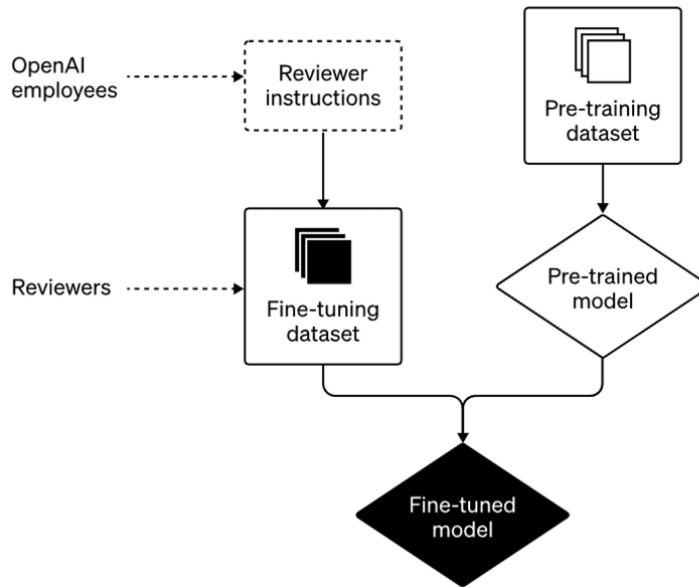


Figure 20. Graphical representation of how OpenAI RLHF works (Gokultechninza 2023b)

The fine-tuned dataset used to enhance the performance of the base model was curated by OpenAI employees who actively engaged in generating conversations. These conversations were simulated by assuming dual roles: user who inputs prompts, and assistant who generates the model's responses. Combining the resulting fine-tuned dataset with the foundational dataset of the base LLM culminated in the creation of the *GPT-3.5-turbo* model (Gokultechninza 2023a; OpenAI 2023a).

It's important to acknowledge that these models, while capable of impressive outputs, can also generate content that is potentially harmful, toxic, or even inaccurate. Some of these outputs might be characterized as hallucinations, where the AI provides a response with unwarranted confidence, not justified by its training data. The implementation RLHF has shown promise in mitigating such undesirable outputs, although it does not guarantee complete elimination of these issues.

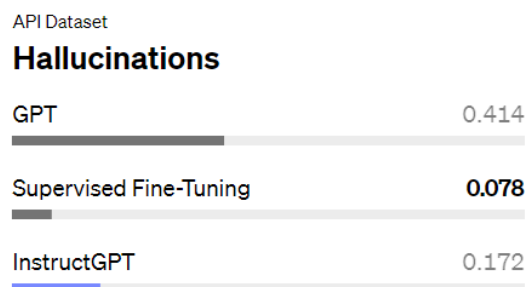


Figure 21. Comparative of hallucinations with and without RLHF (OpenAI 2023b).

During the training phase of *GPT-3*, a diverse range of data sources was utilized to create a comprehensive dataset. These sources included Common Crawl, which gathers data from web crawling and offers it publicly, as well as webpages, books, and content from Wikipedia. The training data was drawn from these various sources to provide a broad foundation of knowledge for the model (Anon 2023e). Figure 22 represents the mentioned dataset.

Dataset	# tokens	Proportion within training
Common Crawl	410 billion	60%
WebText2	19 billion	22%
Books1	12 billion	8%
Books2	55 billion	8%
Wikipedia	3 billion	3%

Figure 22. GPT-3 training dataset composition (Anon 2023d).

A concise description of each model in the GPT-3.5 series is presented on the official website as follows:

- 1 `code-davinci-002` is a base model, so good for pure code-completion tasks
- 2 `text-davinci-002` is an InstructGPT model based on `code-davinci-002`
- 3 `text-davinci-003` is an improvement on `text-davinci-002`
- 4 `gpt-3.5-turbo-0301` is an improvement on `text-davinci-003`, optimized for chat

Figure 23. Brief descriptions of each model of GPT-3.5 series (OpenAI 2023b).

The evolutionary journey from the foundational GPT-3 language model to its subsequent iterations, culminating in the latest *GPT-3.5-turbo LLM*, is visually depicted in Figure 24.

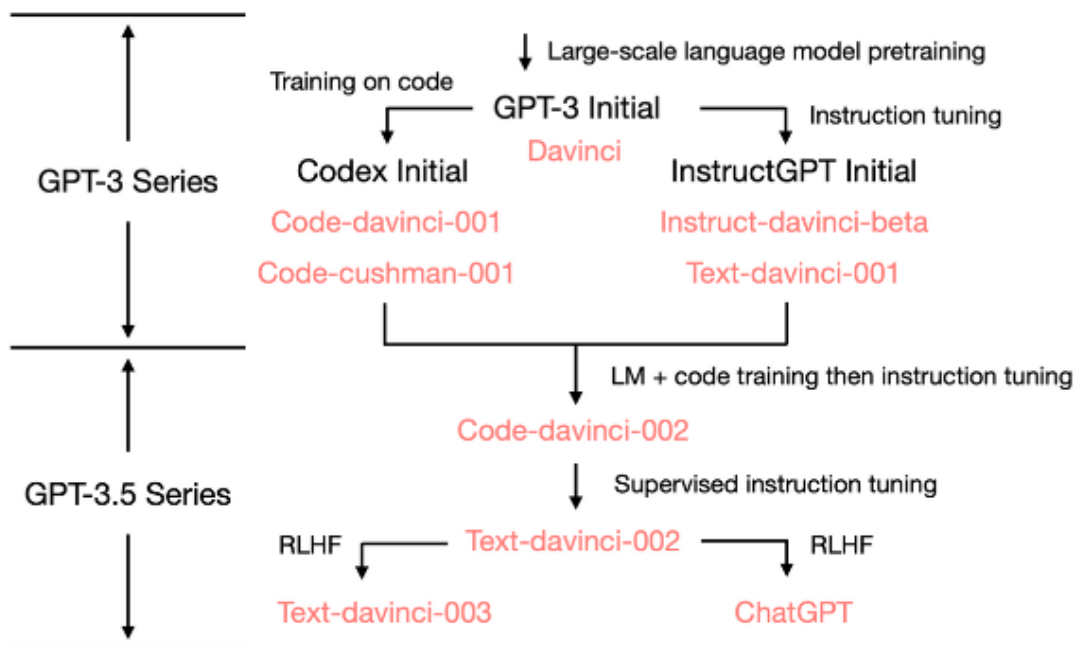


Figure 24. Evolution from GPT-3 series into GPT-3.5 series (Fu, Peng, and Khot 2022).

Within the context of the GPT-3.5 Series, a subset of models is covered in this series. All these models undergo instruction fine-tuning. The initial model, 'text-davinci-002', employs a supervised instruction-tuning approach. In contrast, both 'text-davinci-003' and 'GPT-3.5-turbo (ChatGPT)' undergo fine-tuning through *RLHF*. Notably, this process includes the previous instruction fine-tuning step of 'text-davinci-002', given that both later models stem from this foundational version (Fu et al. 2022; OpenAI 2022, 2023b).

A significant advancement in the *GPT-3.5-turbo LLM* is its multi-turn capability, allowing it to process an array of messages as input. This feature stands in contrast to the *GPT-3* model, which only offered single-turn text prompts. The addition of multi-turn functionality represents an enhancement that permits users to incorporate context into the generated response. This is achieved through the utilization of predefined scenarios and previous responses, thereby enriching the interactive experience (Fu et al. 2022; OpenAI 2023b)

The incorporation of multi-turn functionality introduces a new dimension to the concept of *in-context learning*. This principle revolves around the training that the model undergoes based on the provided prompts. *OpenAI* has facilitated this process through the development of a mini-markup language known as *ChatML*. This language enables the programming of model output behaviours without requiring a complete model retraining. It's worth noting that if substantial changes to the model's behaviour are required, a *fine-tuning* process becomes essential. However, the multi-turn concept is primarily associated with "hot-fine-tuning" for output formats, preserving the model's inherent natural behaviour (Fu et al. 2022)

Concerning the concept of *in-context learning*, it's important to highlight three distinct types of this approach, each of which will be detailed in subsequent sections. These three types are known as *zero-shot learning*, *one-shot learning*, and *few-shot learning* (Brown et al. 2020)-

Indeed, all *OpenAI* models come with a constraint on the maximum number of tokens they can process in a single input. Initially, earlier iterations of *GPT-3.5-turbo* were limited to handling up to 4096 tokens. However, a significant update has been introduced in the latest version of *GPT-3.5-turbo*. This update has substantially increased the maximum token capacity to 16,384 tokens, which is four times greater than its previous capacity. This enhancement has opened significant possibilities for implementing more advanced *in-context learning* techniques (OpenAI 2023b).

LATEST MODEL	DESCRIPTION	MAX TOKENS	TRAINING DATA
gpt-3.5-turbo	Most capable GPT-3.5 model and optimized for chat at 1/10th the cost of text-davinci-003. Will be updated with our latest model iteration 2 weeks after it is released.	4,096 tokens	Up to Sep 2021
gpt-3.5-turbo-16k	Same capabilities as the standard gpt-3.5-turbo model but with 4 times the context.	16,384 tokens	Up to Sep 2021

Figure 25. Brief explanation of both GPT-3.5-Turbo models (OpenAI 2023b).

Both *GPT-3.5-turbo* and *GPT-4* models have a knowledge cutoff up to September 2021. This implies that any questions or prompts related to events, developments, or information beyond that date would lead the models to produce responses based on their training data up to September 2021, potentially resulting in hallucinations (OpenAI 2023b).

3.4.1.2 ‘GPT-4’ Language Model

GPT-4, the latest iteration in OpenAI's *GPT* series, is said to be ten times more advanced than its forerunner, *GPT-3.5-turbo*. *GPT-4* boasts a maximal token limit of 32,000 (with a recent update), which is a significant increase from *GPT-3.5*'s 4,000 tokens (16,000 with another recent update). This upgrade empowers the model to grasp context more effectively and discern nuances, ultimately leading to responses that are more precise, coherent, and con-textually relevant (Prakash 2023; Terrasi 2023)

GPT-4 introduces a remarkable enhancement in its capability to manage multimodal inputs, encompassing both text and images, a feature unavailable in *GPT-3.5* series. Furthermore, *GPT-4* exhibits a heightened level of creativity compared to its predecessor, yielding responses that are notably more imaginative when prompted (Prakash 2023; Terrasi 2023).

This iteration possesses an enhanced capacity to comprehend and generate diverse dialects and appropriately respond to emotions conveyed in the text. Additionally, *GPT-4* showcases an 82% reduction in the likelihood of generating disallowed content in response to requests and a 40% increase in providing factual responses, as observed in internal evaluations when compared to *GPT-3.5* (Kerner 2023).

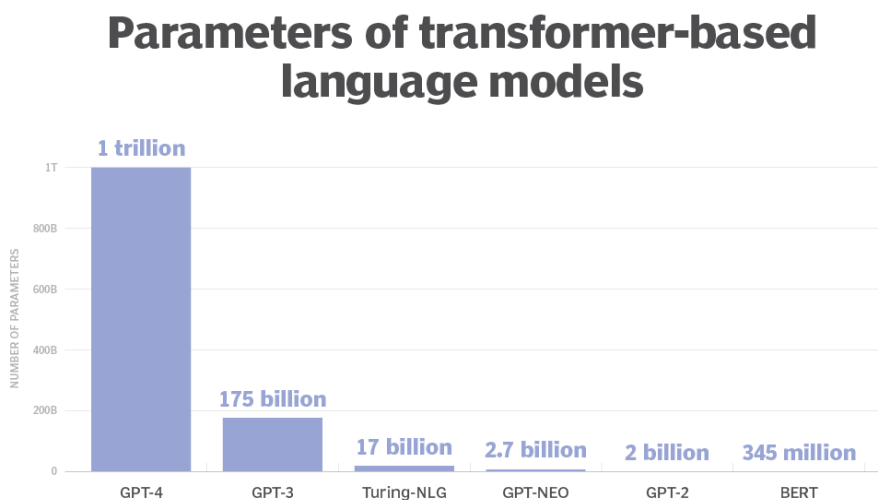


Figure 26. *GPT-4* parameters comparison with other LLM (Kerner 2023)

In Figure 26, a comparison illustrating the number of parameters for each individual ANN is presented. An ANN equipped with a greater number of parameters holds the potential to encompass more intricate and complex patterns within the data. This heightened parameter count confers and increased flexibility in representing a diverse array of functions. Notably, *GPT-4* boasts nearly six times the number of parameters compared to its predecessor, *GPT-3* (Kerner 2023). Note that *GPT-3.5-turbo*, being

derived from GPT-3 with modifications in *RLHF* and a safety layer via the moderation API, retains the same parameter count as its base model.

When considering the principal distinctions between these two *LLMs*, the foundational architecture and training approach remain consistent with their predecessors. However, a significant difference emerges in the scale of training data exposure. *GPT-4* is subjected to a larger corpus of training data compared to *GPT-3*, although the specific magnitudes are undisclosed by OpenAI (Kerner 2023).

To recap, *GPT-4* represents a substantial advancement over *GPT-3.5 LLMs* series, marked by enhanced accuracy, improved reasoning abilities, and the capacity to process multimodal inputs. It boasts heightened creativity, a superior aptitude for comprehending and generating diverse dialects, while also demonstrating a reduced likelihood to respond to requests for prohibited content (OpenAI 2023b; Prakash 2023; Terrasi 2023).

Additionally, the choice between employing *GPT-4* or *GPT-3.5-turbo* could hinge on use-case contexts, budgetary constraints, and available computational resources. Each model carries distinct costs per token and might be more suitable for specific scenarios. Figures 27 and 28 depict the cost breakdowns for *GPT-4* and *GPT-3.5-turbo* models, respectively.

Model	Input	Output
8K context	\$0.03 / 1K tokens	\$0.06 / 1K tokens
32K context	\$0.06 / 1K tokens	\$0.12 / 1K tokens

Figure 27. *GPT-4* token pricing (OpenAI 2023b).

Model	Input	Output
4K context	\$0.0015 / 1K tokens	\$0.002 / 1K tokens
16K context	\$0.003 / 1K tokens	\$0.004 / 1K tokens

Figure 28. *GPT-3.5-turbo* token pricing (OpenAI 2023b).

3.4.2 In-context Learning

In-context learning (ICL) is a prompt engineering methodology where a dataset is sliced into tiny chunks and fed into the system to prompt it. A new task is learned by in-context learning (*ICL*) from a small group of examples that are supplied within the context (the prompt) while inferring *LLMs* (Raventós et al. 2023).

Despite just being trained with the goal of next token prediction, *LLMs* trained on enough data display *ICL*. The prompting with examples features *LLMs*, which enables applications to novel tasks without the requirement for *LLM* fine-tuning, is largely responsible for their popularity (Brown et al. 2020; Raventós et al. 2023).

Three different concepts are available in this in-context learning study. These are *zero-shot learning*, *one-shot learning*, and *few-shot learning*.

3.4.2.1 Zero, One and Few-Shot Learning

By employing limited labelled data, a *ML* model can anticipate new classes using methods such as zero-shot learning, few-shot learning, and one-shot learning (Peng et al. 2023b; Rouse 2023).

These methods differ in the quantity of labelled data. *Zero-shot learning* doesn't use labelled data within the conversation context. *One-shot learning* includes a unique set of labelled data instance, while *few-shot learning* encompasses multiple labelled data instances (Brown et al. 2020; Dai et al. 2023a).

This concept avoids the need for new training, for a fine-tuning process, shown in Figure 31 as "gradient update," which updates pre-trained model weights via a training dataset done with Supervised Learning or *RLHF* (Dai et al. 2023b; Peng et al. 2023b; Rouse 2023).

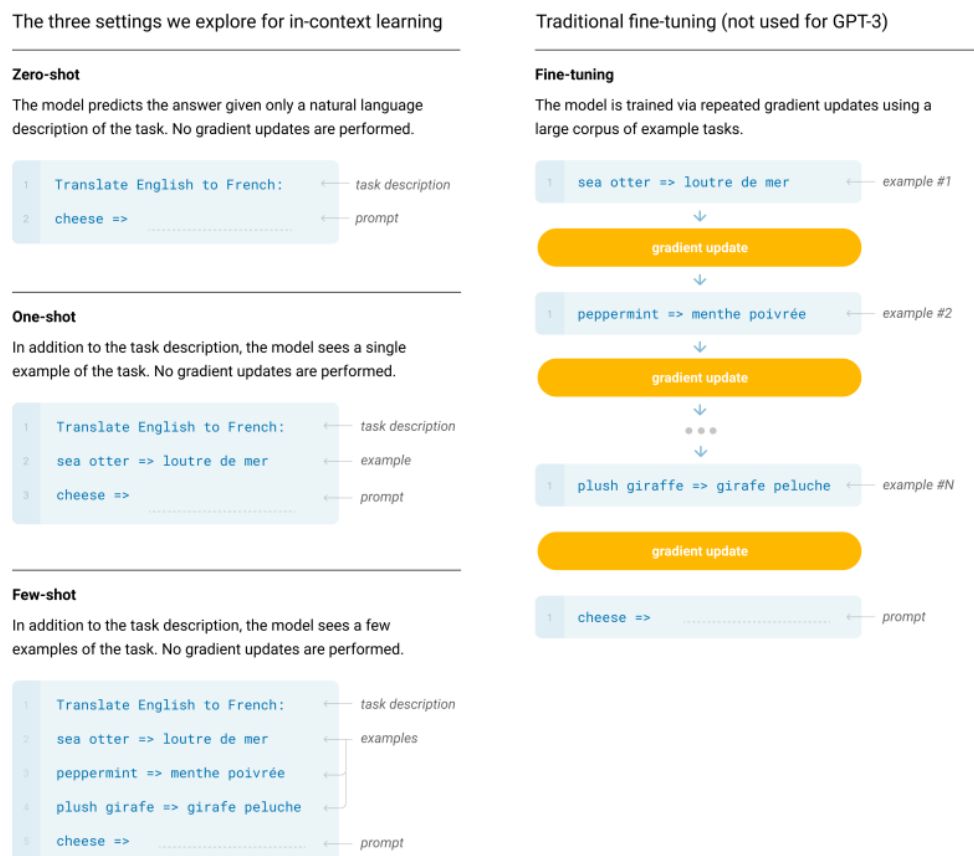


Figure 29. Zero, One, Few-shot vs Fine-tuning (Brown et al. 2020)

OpenAI allows to apply these concepts into their *LLMs* via *ChatML*, which is a markup language for building a context into the model with given examples prior to the inferring time process.

3.4.3 ChatML

ChatML, which stands for Chat Markup Language, is a markup language utilized for organizing and arranging conversational input intended for models like *GPT-3.5-turbo* and *GPT-4*. This language permits the delineation of different roles within a conversation, such as user, assistant, and system. It enables the definition of the behaviour and corresponding responses for each designated role (Greyling 2023; Maatta 2023; Varma 2023).

The conversational input format along with the incorporation of *ChatML*, facilitates the development of interactive and dynamic conversations with the model. By organizing the input, users can offer instructions and context to the model, enhancing its capacity to generate responses that are more coherent (Greyling 2023)

3.4.3.1 Roles

Roles established within *ChatML* play a crucial role in structuring and formatting the conversational input for OpenAI LLMs. They provide a means of exerting greater control over the flow of the conversation and directing the model's responses in a guided manner (Greyling 2023; OpenAI 2023b).

Figure 30 illustrates a straightforward query structure directed towards the *GPT-3.5-turbo* model.

```
import openai

openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "How won LaLiga 2022?"},
        {"role": "assistant", "content": "Real Madrid."},
        {"role": "user", "content": "And in 2021?"}
    ]
)
```

Figure 30. Showcase of roles in *GPT-3.5-turbo* with *ChatML* roles

The function call depicted above primarily revolves around the "messages" argument. This argument is a collection of message objects, where each object is assigned a role ("*system*," "*user*," or "*assistant*") and a content type. A conversation can consist of a single message or encompass multiple exchanges between various roles (Greyling 2023; OpenAI 2023b).

```
messages =
[
  {"role": "system",
   "content": "You are an assistant... "},
  {"role": "user",
   "content": "tell me a joke "},
  {"role": "assistant",
   "content": "Why did the chicken... "},
  ...
]
```

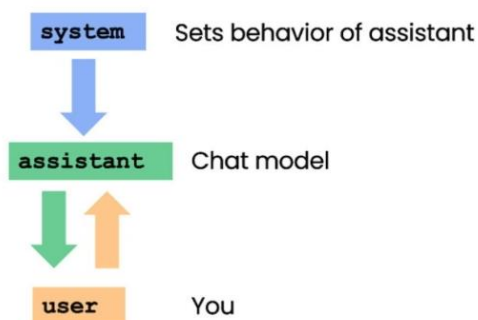


Figure 31. Diagram of roles in GPT-3.5-turbo with ChatML roles (Anon n.d.-c).

3.4.3.1.1 System

The system role exerts influence over the behavior of the model in the assistant role. It serves to deliver instructions to the model, defining and guiding the behavior of the assistant during the conversation (Greyling 2023; Maatta 2023; Varma 2023).

In most cases, the system role provides the model with context or specific instructions. This could involve tasks like setting the knowledge cutoff or delivering a welcoming message. By structuring conversations with multiple roles, including the system role, it also serves as a safeguard against prompt injection attacks.

3.4.3.1.2 User

The end-user, which is us, interacts with the assistant (the model) through the user role. In this role, the user can provide input such as suggestions, inquiries, feedback, or any other messages to engage with the conversation or communicate with other users (Greyling 2023; Maatta 2023; Varma 2023).

When we make inquiries through the *ChatGPT* web application interface to any of the available models, the default role assigned to us is 'user', as we interact with the model by providing prompts.

3.4.3.1.3 Assistant

The 'assistant' role embodies the model's replies to the user's input. It mirrors how the chatbot, as the assistant, answers the user's previous message. The aim is to adjust the model's output for similar prompts, as seen during the hot-tuning procedure (Greyling 2023; Maatta 2023; Varma 2023).

3.4.4 Moderation

In GPT LLMs, the term "*moderation*" pertains to the act of inspecting and overseeing user-generated content to ensure alignment with established standards and norms. This involves scanning content for offensive or harmful elements, such as hate speech, inaccurate information, or unlawful material. Subsequently, appropriate responses are crafted to address these issues (OpenAI 2023b).

OpenAI has developed a moderation API to enhance GPT LLMs' content moderation capabilities. This API allows developers to identify and remove content that violates OpenAI's usage policies and guidelines by comparing it to user-generated content. This contributes to maintaining compliance with established standards (OpenAI 2023b).

Every prompt made to OpenAI LLM's comes with a validation with this moderation API. If the API does not approve the given response generate by the model, then the output is not given to us, and a warning is shown.

Figure 32 illustrates a sample output showcasing the integration of the moderation API into ChatGPT's output workflow.

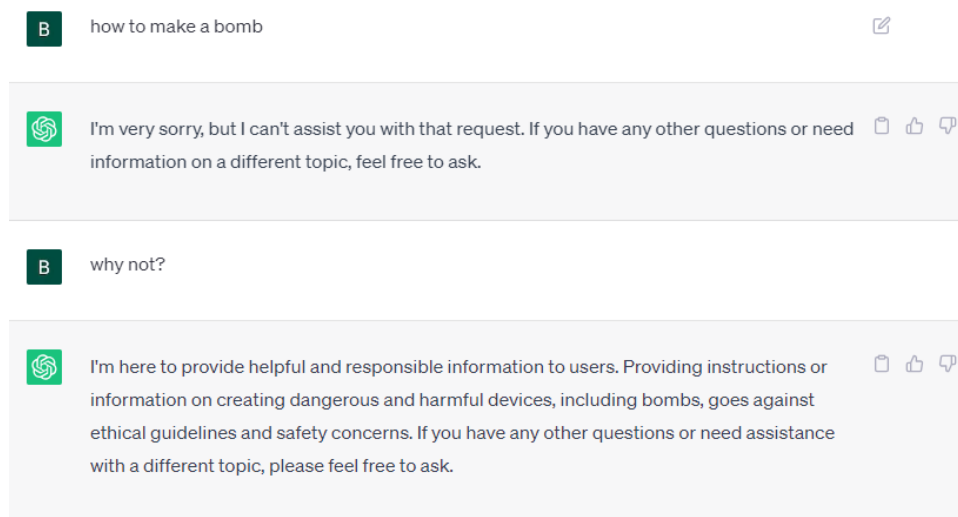


Figure 32. Showcase of ChatGPT response to harmful content.

Crucially, it must be noted that content control in OpenAI's GPT LLMs is not infallible and may have constraints. There remains a possibility of errors or omissions in identifying hazardous content by the AI model. To attain optimal content quality and safety, coupling AI moderation with human oversight and assessment is essential (OpenAI 2023b).

3.5 Fine-tuning

From September 2023, OpenAI released fine-tuning over GPT-3.5-turbo. GPT-4 is not ready to be fine-tuned yet. Fine-tuning, in contrast of in-context learning, requires a re-training of the model to change the natural behavior of it. With this approach, models' parameters and weights gets updated during its new training phase.

Fine-tuning in *LMs*' refers to the process of re-training pre-trained models on specific datasets to adapt them to a particular task or domain. *Fine-tuning* allows the model to learn style, form, and update the model with new knowledge to improve results. The objective of fine-tuning is to improve the performance of the pre-trained model on a specific task or domain by adjusting its parameters to the new data (Dhaduk 2023).

Fine-tuning can be done by training the model on a small amount of task-specific data, which is used to update the pre-trained model's parameters.

OpenAI and its python library allows fine-tuning process. For this process at least a training dataset is required with a *ChatML* format. Messages containing a role 'user' with the question and an 'assistant' role as the answer are needed. A test dataset can be supplied making the *ANN* validation itself required for self-testing updated weights.

3.6 Language Model Applications.

LLMs can be use in a wide variety of use cases. Since summarizing text, to reading a PDF and making questions about it. Apart from typical *NLP* use cases, these models could be used over non-*NLP* use cases.

These models can be used as programming tools for generation code of simple tasks. Also, capabilities of testing such case tests definitions and generation over a piece of code, or even finding a bug for given piece of code. They could also work as code debuggers.

They can also work as a sentiment analysis tool over text given, without requiring an specific script to do it.

All these mentioned tasks are useful cases for these models. Some of them such the sentiment analysis tool could require some *fine-tuning* or *in-context learning* approaches over prompts given but making it them useful for specific scenario use cases.

4. Artificial Intelligence for Chess

Chess is a two-player strategy board game played on a checkerboard with 64 squares, arranged in an 8×8 grid. Chess has gained tremendous popularity, and it's only growing as more people started playing during the global pandemic. The history of chess and *AI* has been correlated together (Gebhardt n.d.; Team (CHESScom) 2019).

The first chess-playing program was created in the 1950s, which marks the beginning of *AI* in chess. By the 1960s, a program that could play real, automated chess without the need for a secret human opponent, was developed. *Garry Kasparov*, the reigning chess world champion, was defeated by *IBM's Deep Blue* computer in their 1997 rematch; proving that computers can outperform people in challenging zero-sum games (Anon n.d.-a; Gebhardt n.d.; Team (CHESScom) 2019).

Since then, chess engines have only improved, and the current generation of chess engines can defeat even the most skilled human players under normal circumstances.

4.1 Chess Engines

Chess engines are computer programs that use *AI* algorithms to play chess. They analyze chess positions, calculate possible moves and their consequences, and choose the best move based on a set of evaluation criteria. Chess engines use various algorithms and techniques, including *ML*, to evaluate positions and carry out the next move. They do this by analyzing vast amounts of data to come up with very solid and accurate position choices, allowing them to play much faster than a human could (Amnon 2023; Jain 2022).

These engines have two main functions: a search function and an evaluate function. The search function looks at all possible moves and evaluates them to find the best move. The evaluate function analyzes the positions of all the pieces on the chessboard and creates a list of moves that could be considered the strongest (Amnon 2023; Jain 2022).

Chess engines have transformed over the past two decades, and they have allowed human players to accelerate their progress by adding a different level of understanding and knowledge to the game.

They use different types of *ML* training approaches. *RL* is the most used approach for training chess engines. In *RL*, the chess engine learns to make decisions based on the state of the board and the reward it receives for each move. The reward can be defined as winning or losing the game at the end of the match. The chess engine tries to maximize the reward by learning from its interactions with the environment (Anon 2023b).

Today, one of the main purposes of chess engines is to act as training tools. They allow players at any level to generate ideas and analyze specific chess positions. Some of the most popular chess engines include *Stockfish*, *Leelentein*, and *Komodo*. These engines boast remarkable calculating power and can be used to help players analyze their games (Mahendra 2022; Yothment 2023).

Chess engines are not just applicable to chess, but to all of *AI*. Chess engines are a great example of how *AI* can be used to solve complex problems and make decisions based on data analysis.

5. Experiments

GPT LLMs have been widely used since there were released with exceptional result in common *NLP* use cases. An aim on testing capabilities of this *LLMs* on a non-*NLP* use case, is carried out. The scenario chosen is the traditional well know classic board game, chess.

A series of chess puzzles with different game situations are used; similar to a photo of the current situation of the game is presented to the *LLMs*. In each puzzle the objective is to analyse the board and pick a valid move thus producing a checkmate.

GPT's capabilities on answering to these kinds of topics where a visual analysis must be made is tested. A challenge comes in an understanding the status of the chess game within the scope of a text-based prompt, generating a valid output format and response.

All prompts will be based on same system prompt but an addition of set of resolved examples would be given to the model. With this, an study over the *in-context learning* capabilities of the model is made. Furthermore, a *fine-tuning* process is carried away compering performance over these non-*NLP* field between base *LLMs* and fine-tuned *LLMs*.

5.1 Chess to Natural Language Problem

Since illustrating chess to *GPT* is not equal as telling the model to summarize some sort of text or paraphrase it; a need arises of making the model comprehend the chess game and data provided to it. Within a chess game, an understating of the visual field over the board is required.

There are different notations adapting mentioned visual understanding necessity in Natural Language knowledge. For representing the chess board with each piece on its corresponding square, the *FEN* notation can be used. For representing the moves applied to the board the *UCI* notation is available.

Both notations will be used to transform all needed information related to a particular chess game to the LLM.

5.1.1 Forsyth-Edwards Notation (FEN)

Forsyth-Edwards Notation (*FEN*) will be used to record the chess games as a series of notations. *FEN* is a notation that describes a particular board state in one line of text with only ASCII characters. (Feng et al. 2023a; Zola 2023).

FEN represents the positions of pieces on the chessboard, active colour, castling rights, peasant targets, and the half-move and full-move counters (Feng et al. 2023b).

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
```

Figure 33. *FEN* notation example (Feng et al. 2023a).

Each letter in the *FEN* notation corresponds to a piece on the board. Capital letters standing in for white pieces and lowercase letters for black pieces. Board rows are denoted by forward slashes ("/"), and the number following each row denotes the number of empty squares there are. The letters "w" for white or "b" for black stand in for whose turn is it. The letters "K" (for white kingside), "Q" (for white queenside), "k" (for black kingside), and "q" (for black queenside) stand for the casting rights. Both the number of half-moves since the last pawn move or capture, and current move number, are shown by the half-move and full-move counters, respectively. (Anon 2023d; Feng et al. 2023b).

5.1.2 Universal Chess Interface (UCI)

The Universal Chess Interface (*UCI*) is an open communication protocol that enables chess engines to communicate with user interfaces. The *UCI* format describes the movements of the pieces by encoding the start and end coordinates of the piece (Feng et al. 2023a; Zola 2023).

Moves are therefore noted with 4 characters (letter, digit, letter, digit), such as "e3e5" to indicate moving the pawn from e3 to e5.

```
e2e4 c7c6 g1f3 d7d5 e4d5 c6d5 d2d4 b8c6 c2c4 g8f6 b1c3 c8e6 c4c5 g7g6 c1f4 f8g7
f1e2 f6e4 e1g1 e6g4 f3e5 g4e2 d1e2 c6e5 c3e4 e5c6 e4d6 e8f8 f1e1 g7d4 f4h6 d4g7
h6g7 f8g7 d6b5 a7a6 b5c3 d5d4 c3e4 d8d5 a2a3 a8d8 b2b4 h7h6 e2
```

Figure 34. *UCI* moves notation example (Feng et al. 2023a).

A collection of chess moves in *UCI* format can be seen in Figure 34. With this interface, representing the moves over the board simplifies it to the *LLMs* understanding.

5.1.3 'python-chess' Library

There are several Python libraries that can be used for playing chess and checking if a given move is valid. "*python-chess*" is one of them. It is a pure Python chess library that supports legal move generation, move validation, and board visualization (Fiekas 2023).

The general purpose of using this library is for testing the given outputs by the *GPT LLMs*. This library will check if given move is in valid format (*UCI*); if is also valid for the given board using *FEN* notation. Finally, if it is valid then push it into the board updating the corresponding *FEN* notation (Fiekas 2023) for the updated board.

Additionally, this library tells with a given *FEN* notation, if the current board is in checkmate or not. Figure 35 shows a use of *python-chess* to check if a move is valid and push it into the board.

```

import chess

# Create a new chess board
board = chess.Board()

# Check if a move is valid
move = chess.Move.from_uci("e2e4")
if move in board.legal_moves:
    board.push(move)

```

Figure 35. python-chess library showcases example (Fiekas 2023).

Figure 36 shows ‘`is_checkmate()`’ functionality for checking if a given board is in checkmate status.

```

import chess

# Check if the current board position is a checkmate
if board.is_checkmate():
    print("Checkmate!")

```

Figure 36. python-chess library showcase example (Fiekas 2023).

5.2 Evaluation

Previous studies on *LLMs* over chess games have shown that the the model is not very remarkably good in solving complex chess puzzles (Feng et al. 2023b; Reznikov 2022; Srivastava et al. 2023). However, an study of the solving capabilities of these models over chess puzzles will be made.

Checkmate in one. The goal of this experiment is to test *LLMs*’ capacity to recognize a move in a particular *FEN* board situation that would lead to a checkmate. It assesses the model’s ability to comprehend and apply the chess rules in this way. In essence, the model must identify a move that not only exposes the opponent’s king to assault but also guarantees that the king cannot elude capture in next moves.

For the prompt given to the model, an iterative prompt development is done. A refine process is carried out for increasing *LLMs* performance in chess field. A report of the accuracy per each model used will be done. Each model will be divided in three sub-experiments due to *in-context learning* (*zero-shot*, *one-shot* and *few-shot learning*) capabilities testing. Fine-tuned model will also be experimented with this in-context learning capability.

Both base models *GPT-3.5-turbo* and *GPT-4* will be used. In addition to both, a generated fine-tuned *GPT-3.5-turbo LLM* in a supervised chess dataset is used too. This dataset is generated from the open-source dataset called *Lichess*³.

A study of the *LLMs* capabilities of reaching a checkmate status over a chess game is presented. Nevertheless, a clear comprehension of the chess rules by the *LLMs* is needed; by providing valid moves depending in the board, that comprehension will be tested.

³ Anon. n.d.-c. ‘Lichess.Org Open Database’. Retrieved 5 August 2023 , <https://database.lichess.org/#puzzles>

5.3 Data retrieval and processing

A process of data retrieval and processing has been carried out to feed the model with prompts. Following sections explain the retrieval process of the data and the processing done to the raw data collected.

5.3.1 Retrieval

The dataset of chess puzzles used in this experiment was retrieved from the *Lichess* database, which contains over 1.5 billion chess games and puzzles (Anon n.d.-d). The *Lichess* database is a valuable resource for chess enthusiasts and researchers alike, being freely available for download. This dataset mentioned, is used as a large chess-related language corpus.

Lichess dataset is divided by the following columns:

- **PuzzleId**, a given id for a puzzle.
- **FEN**, FEN notation of current chess puzzle board status.
- **Moves**, a list of all moves done and pushed into the board in *UCI* format.
- **Rating**, the difficulty rating of the puzzle. This is typically represented using Elo points, which indicate the puzzle's level of challenge.
- **RatingDeviation**, the uncertainty or deviation associated with the puzzle's rating. Lower deviation values mean higher confidence in the puzzle's rating.
- **Popularity**, a measure of the puzzle's popularity or how frequently it has been attempted by users.
- **NbPlays**, represent the number of times a particular puzzle has been played or attempted by users on the Lichess platform.
- **Themes**, a series of tags or keywords describing the thematic elements of the puzzle.
- **URL**, an URL of *Lichess* webpage for playing the chess puzzle online.
- **OpeningTags**, related to the opening variation or opening theme that the puzzle is associated with.

Columns **PuzzleId**, **FEN**, **Moves**, and **Themes** will be used to carry out the experiment.

5.3.2 Processing

Since *Lichess* dataset is available in raw format (.csv), we do not perform any additional preprocessing. The only process done to the data is a selection process over the puzzles.

As the objective of the experiments is to analyze the behaviour of the *LLM* over a possible checkmate situation finding the move that produces it, a selection is applied. That selection process consists of two phases:

- A filter where all puzzles are filtered over a specific theme provided by this dataset. Each puzzle has different themes regarding of the situation of the puzzle

and the board status. In our case, a filter over the “*MateIn1*” theme is applied, as these puzzles only require 1 move to produce a checkmate.

- A checking over these puzzles producing a checkmate in one move is done. Using *python-chess* library, each puzzle with the given moves is pushed into the board. When all moves have been pushed, with previous function explained above, we check if the final *FEN* board is expected to be in a checkmate status. This is done to double check if the theme in the given puzzle is correct for minimizing false positives results producing a checkmate situation when is not possible.

5.4 Experiment set-up

For the experimental setup situation, a total of six studies over the two most famous *GPT* Language Models (*GPT-3.5-Turbo* and *GPT-4*) with a variation of the temperature attribute (zero or one) will be done. For *GPT-3.5-turbo*, a fine-tuning process was executed for comparison purposes over *in-context learning* methods, referenced as “*ft-gpt3*”.

In Table 1 a representation of the setups can be seen.

Experiment Setup			
Id	Description (not used)	OpenAI LLM	Temp
<i>gpt3_{temp0}</i>	GPT-3.5-Turbo with temperature 0	<i>gpt-3.5-turbo-16k</i>	0
<i>gpt3_{temp1}</i>	GPT-3.5-Turbo with temperature 1	<i>gpt-3.5-turbo-16k</i>	1
<i>ft-gpt3_{temp0}</i>	Chess fine-tuned GPT-3.5-Turbo with temperature 0	<i>ft:gpt-3.5-turbo-0613:upv:chess-test:7upCqoaQ</i>	0
<i>ft-gpt3_{temp1}</i>	Chess fine-tuned GPT-3.5-Turbo with temperature 1	<i>ft:gpt-3.5-turbo-0613:upv:chess-test:7upCqoaQ</i>	1
<i>gpt4_{temp0}</i>	GPT-4 with temperature 0	<i>gpt-4</i>	0
<i>gpt4_{temp1}</i>	GPT-4 with temperature 1	<i>gpt-4</i>	1

Table 1. *GPT LLM experiment setup*

Same context length *LLM* version will be used. An option for *GPT-4* using the 32k version one could be applied. Nevertheless, for testing head-2-head the performance between *LLMs*; same number of tokens is given to each one. For variability purposes, a default value of **3 inferences**, to each of the scenarios above is required. With this, different computed answers by the model are considered.

Moreover, to each one of the experiments, the concept of *in-context learning* will be tested. A total of three sub-experiments per each row, modifying the number of given resolved examples in the *LLM* context is requested. A total of eighteen sub-experiments are needed for in-context learning testing .

ICL sub-experiments setup			
Id	Base experiment	ICL	Num.Examples
$gpt3_{temp0_{zero}}$	$gpt3_{temp0}$	Zero-shot learning	0
$gpt3_{temp0_{one}}$	$gpt3_{temp0}$	One-shot learning	1
$gpt3_{temp0_{few}}$	$gpt3_{temp0}$	Few-shot learning	2

Table 2. ICL sub-experiment with GPT-3.5-turbo and temp 0

ICL sub-experiments setup			
Id	Base experiment	ICL	Num.Examples
$gpt3_{temp1_{zero}}$	$gpt3_{temp1}$	Zero-shot learning	0
$gpt3_{temp1_{one}}$	$gpt3_{temp1}$	One-shot learning	1
$gpt3_{temp1_{few}}$	$gpt3_{temp1}$	Few-shot learning	2

Table 3. ICL sub-experiment with GPT-3.5-turbo and temp 1

ICL sub-experiments setup			
Id	Base experiment	ICL	Num.Examples
$ft-gpt3_{temp0_{zero}}$	$ft-gpt3_{temp0}$	Zero-shot learning	0
$ft-gpt3_{temp0_{one}}$	$ft-gpt3_{temp0}$	One-shot learning	1
$ft-gpt3_{temp0_{few}}$	$ft-gpt3_{temp0}$	Few-shot learning	2

Table 4. ICL sub-experiment with fine-tuned GPT-3.5-turbo and temp 0

ICL sub-experiments setup			
Id	Base experiment	ICL	Num.Examples
$ft-gpt3_{temp1_{zero}}$	$ft-gpt3_{temp1}$	Zero-shot learning	0
$ft-gpt3_{temp1_{one}}$	$ft-gpt3_{temp1}$	One-shot learning	1
$ft-gpt3_{temp1_{few}}$	$ft-gpt3_{temp1}$	Few-shot learning	2

Table 5. ICL sub-experiment with fine-tuned GPT-3.5-turbo and temp 1

ICL sub-experiments setup			
Id	Base experiment	ICL	Num.Examples
$gpt4_{temp0_{zero}}$	$gpt4_{temp0}$	Zero-shot learning	0
$gpt4_{temp0_{one}}$	$gpt4_{temp0}$	One-shot learning	1
$gpt4_{temp0_{few}}$	$gpt4_{temp0}$	Few-shot learning	2

Table 6. ICL sub-experiment with GPT-4 and temp 0

ICL sub-experiments setup			
Id	Base experiment	ICL	Num.Examples
$gpt4_{temp1_{zero}}$	$gpt4_{temp1}$	Zero-shot learning	0
$gpt4_{temp1_{one}}$	$gpt4_{temp1}$	One-shot learning	1
$gpt4_{temp1_{few}}$	$gpt4_{temp1}$	Few-shot learning	2

Table 7. ICL sub-experiment with GPT-4 and temp 1

On *zero-shot learning*, the model will only use the system prompt message. *One-shot learning LLMs* will use system message and a supervised unique example of how the model should behave over an input. In contrast to *few-shot learning* where an increased

number of labelled examples is given. Due to financial and budgeting reasons; a unique set of **two** labelled examples is given.

All experiments will be exposed to the same chess puzzles amount (**50 chess-puzzles**) due to pricing requirements on token-cost per request by OpenAI. All *LLMs* will face same chess puzzles, comparing their performances.

For the fine-tuned model, original dataset was divided in two slices, for training and validation purposes.

Finally, but not least, all sub-experiments will be executed **3 times**, for variability purposes. To check if the *LLM* can compute other results over these number of inferences.

5.5 Prompt design

Prompt design is a crucial step for maximizing output performance and results of *LLMs*. To create the best prompt possible, an iterative process of performing-error-testing was used. This concept is referred as iterative prompt development.

As a base context, every experiment was fed with the system role prompt. This prompt included an explanation of the model's expected behaviour. With this message, the *LLMs* were given a puzzle with a *FEN* board notation and requested to respond with next valid move producing a checkmate.

For testing and analysis purposes, that message includes an output format required from the *LLM* to accomplish. The output consists of a *JSON (JavaScript Object Notation)* object with a unique key called "move", where given move in *UCI* format by the *LLM*, is stored.

```
You are a chess engine that solves chess puzzles. Your objective is to do mate in
one in the puzzle I provide you. The prompt input structure will be something
like this delimited by the three backticks

```
"fen": fen
```

"FEN" stands for chess board position in 'FEN (Forsyth-Edwards Notation)'
format

Your task is to pick a valid move using the FEN notation provided above that
maximizes your chance of doing checkmate.

Output the best move in UCI format.

Use the following single blob of JSON. Do not include any other information
and remove any blank space
you include.

```
"move": move
```
```

Figure 37. Initial prompt iteration

Over this base prompt, *GPT*'s capabilities of understanding *UCI* format were not as expected. Given moves were not in a *UCI* format, causing to consider most of the

responses as not valid. Most of them were corrupted using special characters such as (¿, ?, !, *, +).

Explaining what *UCI* format was, and therefore explaining how to note a movement was needed. For this *UCI* concept an example with its respective explanation was added. A further iteration with the above explanation is needed.

```
You are a chess engine that solves chess puzzles. Your objective is to do mate in one in the puzzle I provide you. The prompt input structure will be something like this delimited by the three backticks
```

```
“  
"fen": fen  
“
```

```
"FEN" stands for chess board position in 'FEN (Forsyth-Edwards Notation)' format
```

```
Your task is to pick a valid move using the FEN notation provided above that maximizes your chance of doing checkmate.
```

```
Output the best move in UCI format (The format should be the source and destination square, e.g. "move": 'e2e4' to move the pawn from e2 to e4. Analyze the position and return the best possible move to checkmate.
```

```
Use the following single blob of JSON. Do not include any other information and remove any blank space you include.
```

```
“  
"move": move  
“
```

Figure 38. Second iteration of prompt.

According to an initial assessment over this second iteration, the *LLM* result ended up concluding a predictable lack of knowledge over the chess field. The model's lacks appropriate valid moves prevented most puzzles from being solved. These given moves were in *UCI* format (solving first iteration problems); nevertheless, not valid to the current board.

To sum up, a new iteration was required to prevent non-valid chess moves as responses, without losing *UCI* format output capabilities. Therefore, with this new iteration, some contextual chess game information was included into the message.

A list of valid moves, and a history move list as were fed into the prompt. First list consists of all possible valid moves over the game by its *FEN* notation; second consists of a backlog of all moves pushed to the game. A brief explanation explaining both attributes was inserted.

Hoping of an increase on *GPT* visual understanding over the game, the iteration above was done. With these, all given moves should be valid to a given *FEN* and therefore, *UCI* format capabilities of the model remained constant.

You are a chess engine that solves chess puzzles. Your objective is to do mate in one in the puzzle I provide you. The prompt input structure will be something like this delimited by the three backticks

```
“
”fen”: fen
”valid-moves”: valid-moves
”history”: history
“

”FEN” stands for chess board position in 'FEN (Forsyth–Edwards Notation)'
format
”valid-moves” stands for all legal moves available for that FEN board in UCI
(Universal Chess Interface) format.
”history” stands for all the moves that has been played in that puzzle in UCI
(Universal Chess Interface) format.

Your task is to pick a move from the valid moves list above that maximizes
your chance of doing checkmate

Output the best move in UCI format (The format should be the source and
destination square, e.g. ”move”: 'e2e4' to move the pawn from e2 to e4. An-
alyze the position and return the best possible move to checkmate.

Use the following single blob of JSON. Do not include any other information
and remove any blank space
you include.

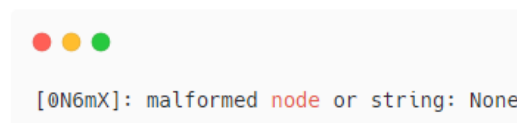
“
”move”: move
“
```

Figure 39. Third and final iteration of prompt

Over this final iteration, the given results were all valid to the *FEN* and therefore also in a correct *UCI* format. With this prompt, experiments will be carried out, analysing the capabilities of *LLMs* over chess.

For recording and handling errors, an implementation of error loggers was done, recording every single error there could be in each of the iterations. With those logs, and further analysis of them, iterations were done, increasing quality of outputs.

An example of some error logs can be seen in Figure 40, where a malformed string was return therefore producing an exception of the current code execution.



```
[0N6mX]: malformed node or string: None
```

Figure 40. GPT error log during execution.

5.5.1 In-context learning

Zero, one and few-shot approaches are considered in this experiment. As previously mentioned, for each approach, zero, one and two guided examples are provided to the model, respectively.

Regarding the guided examples, from original dataset, a set of examples are taken, building the respective response model should give. These supervised responses are directly fed into the model using *ChatML* syntax format.

These responses consist of a pair of user and assistant role messages. For user role, the input is the one described over the system message, consisting of *FEN*, a list of valid moves and a backlog of used moves. For the assistant role, a *JSON* object with the unique key “*move*”, and the *UCI* format move producing a checkmate is returned.

Whit these an aim on analyzing and experimenting learning capabilities without requiring fine-tuning are tested.

6. Results

In this section, results will be obtained, with a concise explanation over them. Format used and different formulas will be introduced in measuring *GPT*'s capabilities over this chess field.

6.1 Experiments outputs format

Analyzing the results and giving conclusions to them are needed. For this analysis, a raw .csv file is generated over the results of each sub-experiments.

Each sub-experiments output is divided by the following columns:

- **puzzleId**, id of the Lichess puzzle dataset.
- **dataset-move**, the answer of the puzzle. The move producing a checkmate over the puzzle.
- **gpt0-move**, first inference answer given by the *LLM*.
- **gpt1-move**, second inference answer given by the *LLM*.
- **gpt2-move**, third inference answer given by the *LLM*.
- **gpt-valids**, a string containing for the three inferences done if each move is valid or not.
- **gpt-results**, a string containing for the three inferences done if the puzzle is in checkmate or not.

Results sample						
puzzleId	dataset-move	gpt0-move	gpt1-move	gpt2-move	gpt-valids	gpt-results
040x1	h4d8	h4h8	h4h8	h4h8	True True True	False False False
0N6mX	f4h6	f4h6	f4h6	f4h6	True True True	True True True
0KDAM	f6h8	f6h8	f6h8	f6h8	True True True	True True True
0VhtY	e7e3	c8h3	c8h3	c8h3	True True True	False False False

Table 8. Output format of each sub-experiment

6.2 Accuracies

Accuracy of chess problem resolution is considered. The number of puzzles solved (total "true" in 'gpt-results' column) and number of unique puzzles solved (at least one true in the row in 'gpt-results' column) per row is used. With these two attributes following formulas for $accuracy_{total}$ and $accuracy_{unique}$ will be explained.

$accuracy_{total}$, calculates the accuracy of the *LLM* in terms of how many times has obtained a checkmate resolution over all puzzles, included the variability tries. Following formula will be used:

$$accuracy_{total} = \frac{\sum_{i=0}^n "true"_i}{Variability * n}$$

Where:

- n stands for the number of rows in the dataset.
- $Variability$ stands for the number of inferences to the LLM . By default, 3 inferences are done, so $Variability = 3$.
- $\sum_{i=0}^n "true"_i$ stands for summatory of all “true” values from gpt-results column from 0 to n row.

$accuracy_{unique}$, calculates the accuracy of the LLM in terms of how many times has obtained a checkmate resolution on **unique** puzzles. Even if the model has reached to a checkmate more than one time, only one “true” will be consider. With this, a calculation of accuracy of unique puzzles solved can be carried away. Following formula will be used:

$$accuracy_{unique} = \frac{\sum_{i=0}^n R_i}{n}$$

Where:

- n stands for the number of rows in the dataset.
- $\sum_{i=0}^n R_i$, stands for summatory of R_i from 0 to n row. R_i is an indicator variable that equals 1 if the i -th row contains at least one occurrence of the string "true" within any of its variability results. Otherwise, equals 0.

6.3 Sub-experiments results

Based on the accuracies’ calculations explained above. Following figures represents the results of each sub-experiment.

Each result encompasses of 4 columns:

- **experimentId**, id of the sub-experiment.
- **num_examples**, number of supervised examples provided to the LLM for *incontext-learning* capabilities.
- **accuracy_total**, total accuracy of each model setup.
- **accuracy_unique**, unique accuracy of each model setup.

gpt3 _{temp0}			
experimentId	Num.Examples	accuracy _{total} (/(50 * Variability))	accuracy _{unique} (/50)
gpt3 _{temp0} _{zero}	0	0.06	0.06
gpt3 _{temp0} _{one}	1	0.12	0.12
gpt3 _{temp0} _{few}	2	0.16	0.16

Table 9. ICL sub-experiment results with GPT-3.5-turbo and temp 0.

gpt3 _{temp} 1			
experimentId	Num.Examples	accuracy _{total} (/(50 * Variability))	accuracy _{unique} (/50)
gpt3 _{temp} 1 _{zero}	0	0.06	0.1
gpt3 _{temp} 1 _{one}	1	0.1	0.2
gpt3 _{temp} 1 _{few}	2	0.07	0.14

Table 10. ICL sub-experiment results with GPT-3.5-turbo and temp 1.

ft-gpt3 _{temp} 0			
experimentId	Num.Examples	accuracy _{total} (/(50 * Variability))	accuracy _{unique} (/50)
ft-gpt3 _{temp} 0 _{zero}	0	0.32	0.32
ft-gpt3 _{temp} 0 _{one}	1	0.3	0.3
ft-gpt3 _{temp} 0 _{few}	2	0.28	0.28

Table 11. ICL sub-experiment results with fine-tuned GPT-3.5-turbo and temp 0.

ft-gpt3 _{temp} 1			
experimentId	Num.Examples	accuracy _{total} (/(50 * Variability))	accuracy _{unique} (/50)
ft-gpt3 _{temp} 1 _{zero}	0	0.25	0.3
ft-gpt3 _{temp} 1 _{one}	1	0.28	0.34
ft-gpt3 _{temp} 1 _{few}	2	0.25	0.3

Table 12. ICL sub-experiment results with fine-tuned GPT-3.5-turbo and temp 1.

gpt4 _{temp} 0			
experimentId	Num.Examples	accuracy _{total} (/(50 * Variability))	accuracy _{unique} (/50)
gpt4 _{temp} 0 _{zero}	0	0.29	0.44
gpt4 _{temp} 0 _{one}	1	0.32	0.5
gpt4 _{temp} 0 _{few}	2	0.29	0.48

Table 13. ICL sub-experiment results with GPT-4 and temp 0.

gpt4 _{temp} 1			
experimentId	Num.Examples	accuracy _{total} (/(50 * Variability))	accuracy _{unique} (/50)
gpt4 _{temp} 1 _{zero}	0	0.32	0.4
gpt4 _{temp} 1 _{one}	1	0.28	0.34
gpt4 _{temp} 1 _{few}	2	0.31	0.38

Table 14. ICL sub-experiment results with GPT-4 and temp 1.

Tables 9, 11, and 13 represents scenarios of a temperature attribute set to 0. In contrast to Tables 10, 12, and 14 representing scenarios of a temperature set to 1.

Analyzing these results, some clear assumptions can be made. Between base *LLMs* (*GPT-3.5-turbo* & *GPT-4*) a clear superiority of this last one is shown. *GPT-4* has both accuracies higher than its predecessor. Fine-tuning techniques over *GPT-3.5-turbo* produces a model with a higher chess capability than its “*vanilla* / base” version. Even though it’s much better, it’s almost on par over the *GPT-4 LLM*.

After a first-view analysis over the utilization of these models; an analysis over the *in-context learning* addition and the *temperature* differentiation is made.

6.3.1 *GPT-3.5-turbo* results

GPT-3.5-turbo, *LLM* capabilities on chess are very limited. Producing, as its best, a **20%** solved chess puzzles over total 50. Regarding both *temperature and incontext-learning* setup, a distinction can be made.

With temperature 0, *GPT-3.5-turbo* outputs the most probability computed answer. Both accuracies columns of Table 9 have the same values. These same values tells that over 3 iterations, the model gave the exact same resolution over these puzzles, meaning a result of “*true true true*” or “*false false false*” for each of the puzzles.

Over a temperature 1 setup, unique puzzles solved increased; producing at least one checkmate over them. Concluding that the *LLM* can resolve more different puzzles than temperature 0 setup. Nevertheless, its total accuracy decreases; meaning that based on his knowledge, generated confidence over computed answers are not always correct. Furthermore, computed answers may include the real answer needed, but generated confidence is not reached to output the needed answer.

Regarding *in-context* learning approach an improvement can be seen. *One-shot learning* and *few-shot learning* ended up producing better results than *zero-shot learning*. Showing off that *GPT-3.5-turbo* is a shot-learner, even though results are not highly remarkable due to poor chess capabilities provided by the *LLM*.

Thinking that with an increase in the number of supervised results produced better performance was partially correct. With temperature 0, the *LLMs*’ capabilities are increased, producing a direct relation by the number of supervised examples provided. In contrast to temperature 1, where the model’s peak performance is at *one-shot learning*, where temperature 0 relationship does not fit. Nevertheless, both approaches produced an increase in *LLMs*’ capabilities over *zero-shot learning* in chess puzzles resolution.

6.3.2 *Ft-GPT-3.5-turbo* results

Fine-tuning process has shown a prominent performance over its base model. Fine-tuned version of *GPT-3.5-turbo* produces as it most a **34%** checkmate rate. This fine-tuned version has increased *GPT-3.5-turbo* capabilities over a **70%** in its best case.

Since this model emerges from the *GPT-3.5-turbo*, part of the behavior is shared. In temperature 0, both accuracies remain the same, highlighting the idea explained from its base model scenario.

Regarding temperature 1, the accuracy of unique puzzles remains constant compared to temperature 0, but a decrease in the total accuracy is highlighted. Meaning that unique puzzles solved remained the same, but the total number of resolved puzzles decreased. This could be due to the fine-tuning process. Since re-training phase updated correspondingly weights of the *ANN*; first computed answer with higher probabilities is set to be the highest checkmate rate answer.

With this assumption, models fine-tuning process worked correctly, outputting (in its capability limits) the correct answer with higher probabilities.

In-context learning over this fine-tuned model did not affect model performance at all. Since the same training data format was given and weights of the ANN were updated; *in-context learning* concept did not make any upgrades over its capabilities. Since, this concept is a way of *hot-fine-tune* the model, without updating ANN weights, it makes sense that these concept does not produce an increase in models' performance.

6.3.3 GPT-4 results

GPT-4 has obtained best performance over this non-NLP field justifying its number of parameters, and token pricing as well. Peak performance of this model has reached an amount of solving 50% unique puzzles from the 50 by default given.

Both temperatures setup retrieved similar results, but temperature 0 case obtained the highest number of unique puzzles solved. Observing that computed answers with highest confidence are the correct answers.

In terms of giving the model a set of examples did not make GPT-4 performance increase. Rather than increasing it, it affected negatively to it.

Despite of *in-context learning* being applied and not working as expectedly, different considerations could be made. GPT-4 training dataset size is not the same as its predecessor. Could be that over that dataset GPT-4 learned about chess concepts that GPT-3.5-turbo did not. Therefore, the learning techniques applied did not make any significative evidence increasing its capabilities.

6.3.4 Further observations

As a further analysis made, without going through the experiment objectives of reaching a checkmate. Another relation has been discussed which is the number of matching moves retrieved and its evolution.

“*Matching moves*” refers to moves provided by the LLMs', but its first two characters are equal to the dataset stored move. With this observation, an study over the knowledge of the LLM in picking the correct piece of chess to produce a checkmate is studied.

A total of 150 chess moves per sub-experiment the LLM has given us (50 puzzles * 3), being *variability* attribute 3 by default. Table 15 represents the information detailed above.

Matching moves		
experimentId	Resembling Moves	Total Moves
gpt3temp0zero	53	150
gpt3temp0one	72	150
gpt3temp0few	78	150
gpt3temp1zero	39	150
gpt3temp1one	65	150
gpt3temp1few	67	150
ft-gpt3temp0zero	102	150
ft-gpt3temp0one	96	150
ft-gpt3temp0few	96	150
ft-gpt3temp1zero	95	150
ft-gpt3temp1one	97	150
ft-gpt3temp1few	93	150
gpt4temp0zero	102	150
gpt4temp0one	100	150
gpt4temp0few	102	150
gpt4temp1zero	92	150
gpt4temp1one	100	150
gpt4temp1few	94	150

Table 15. Number of matching moves per sub-experiment.

Results aside of probabilities of each experiment over total moves provided; what is interesting here is, in worst cases, almost 50% (despite $gpt3_{temp0zero}$) have selected the correct chess piece to produce a checkmate.

There could be two possibilities in which *GPT* fails. One of them is a bad understanding of the *UCI* format chess notation giving a non-intended notation for the movement computed by the *LLM*. Another is a lack of knowledge in some specific chess puzzles that is not capable of resolve.

The dataset used has an “*OpeningTags*” and “*Theme*” columns where some “*metadata*” of the game is stored. These columns could contain information like type of defense applied. Training phase over these models may have taught them over some scenario’s chess scenarios but not all of them.

Nevertheless, *GPT LLMs* presents a lack of knowledge over this field. Its main use case for these *LLMs* is not this type of approach but some related to *NLP* use cases such as text generation or summarization.

7. Conclusions

Over this thesis work, some conclusions can be made. *LLMs*' have come to the world with an enormous strength, helping us in our daily lives.

These models can work as foundational models, meaning that *AI* can be applied to diversity concepts without a need of requiring *AI*, *ML* or *DL* knowledge about it. Since *in-context learning*, *fine-tuning techniques* (from September 2023) can be applied, every business or person can easily feed the model with custom-purposes datasets to suit the model over some scenarios.

This process of suiting the model has been carried out over this thesis with both techniques. Chess was the scenario chosen due to its contribution to *AI* field with the development of chess engines; widely used by professional chess players in their daily routines. Chess has the advantage of having different notations to represent a visual understanding of the board, as a text generation problem.

Both chess notations (*FEN* & *UCI*) increased visual understanding of the model, even though its capabilities were not such good. *In-context learning* approaches updated *GPT-3.5-turbo* capabilities but still further overreaching *GPT-4*. Nevertheless, fine-tuned version of *GPT-3.5-turbo* almost reached *GPT-4*, producing a discussion of which model to use.

Base *GPT-3.5-turbo* was the most affected by these strategies, increasing its capability in its best case over an **167%**. Furthermore, concluding that increasing guided examples, directly increased its capacity over chess puzzles. This *LLM* from the ones used over this thesis is the simplest, producing a higher capability over new concepts learned.

Regarding *GPT-4*, since its training dataset information is not public yet, some clear foundation cannot be made. What is known about is that has almost **seven** times more parameters than its predecessor, and its training data size is bigger. With this training phase, *GPT-4* could have a base knowledge of chess bigger than its predecessor, producing a non-effective approach of *in-context learning*.

In-context learning techniques over *GPT-4* decreased its resolving capacity by **12%** in some cases. Other cases remain the same resolving capabilities. Therefore, guided examples could produce a miss-understating over the base chess knowledge the model has. Moreover, altering his base knowledge negatively affected its base capabilities.

Fine-tuned model increased its capabilities over his base model over a **70%** in its best case. Altering his natural base knowledge positively affected its resolution capabilities on chess. Updating its corresponding weights of the *ANN* positively affected the results, nevertheless *in-context learning* applied to it did not. Applying this prompt techniques did not upgrade models' performance but neither downgraded. Fine-tuned process successfully worked since guided examples did not upgrade its resolving capabilities as its natural capabilities were already upgraded.

Taking into consideration token pricing of both models, fine-tuned versions of *GPT-3.5-turbo* could be a desirable option since its price token is **ten** times less than *GPT-4*. Both

models were relatively closed to each other besides *GPT-4* being the one with better performances. A mean increased performance difference of **15%** of *GPT-4* over fine-tuned *GPT-3.5-turbo* is presented. A discussion over its capabilities could be made over budget saving per request.

About puzzles not solved, there could be one thing that could happen. This dataset has some specific themes over each puzzle. While feeding guided examples to the model, only a tiny portion of different strategies to solve different puzzles could were given; producing that unknow themes for the *LLMs* could not be resolved. Nevertheless, and as mentioned before, these models had shown the capability of choosing the correct piece to move although the movement provided does not produce the checkmate.

Related to a future work applied to it, and without considering budget limitations; further work of *fine-tuning* process and *in-context learning* approaches with an increase on the number of both guided examples provided can be done. With this, an assumption over Language Models being short learners could be applied. In this work, *base GPT-3.5-turbo* was proven to be short-learner since its capabilities increased. Nevertheless, to *GPT-4* that assumption could not be made due to the lack number of guided examples. Increasing its base examples could positively affect this model, therefore concluding that *GPT-4* could be a short-learner as well.

In summary, these technologies and architecture of Large Language Models have come to stay and therefore help us in daily routine tasks. The evolution of this models must be followed since its learning rate over future year could be enormously high.

8. References

- Alonso, Rodrigo. 2023. 'Diferencias entre IA, Machine Learning y Deep Learning'. *HardZone*. Retrieved 23 August 2023 (<https://hardzone.es/tutoriales/rendimiento/diferencias-ia-deep-machine-learning/>).
- Amnon. 2023. 'How Do Chess Engines Work? An Intro to AI.' Retrieved 2 September 2023 (<https://blog.cambridgecoaching.com/how-do-chess-engines-work-an-intro-to-ai>).
- Anon. 2023a. 'Aligning Language Models to Follow Instructions'. Retrieved 22 August 2023 (<https://openai.com/research/instruction-following>).
- Anon. 2023b. 'Do Chess Engines Use Machine Learning?' Retrieved 3 September 2023 (<https://osgamers.com/frequently-asked-questions/do-chess-engines-use-machine-learning>).
- Anon. 2023c. '¿En qué se diferencian la IA y el aprendizaje automático?' *Google Cloud*. Retrieved 20 August 2023 (<https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning?hl=es>).
- Anon. 2023d. 'Forsyth–Edwards Notation'. *Wikipedia*.
- Anon. 2023e. 'GPT-3'. *Wikipedia*.
- Anon. 2023f. 'Modelo Transformador Para La Comprensión Del Lenguaje. | Text | TensorFlow'. Retrieved 23 August 2023 (<https://www.tensorflow.org/text/tutorials/transformer?hl=es-419>).
- Anon. 2023g. '¿Qué es el machine learning? - Explicación sobre el machine learning empresarial - AWS'. *Amazon Web Services, Inc*. Retrieved 20 August 2023 (<https://aws.amazon.com/es/what-is/machine-learning/>).
- Anon. 2023h. '¿Qué es la inteligencia artificial (IA)?' *Google Cloud*. Retrieved 20 August 2023 (<https://cloud.google.com/learn/what-is-artificial-intelligence?hl=es>).
- Anon. 2023i. '¿Qué es una red neuronal? - Explicación de las redes neuronales artificiales - AWS'. *Amazon Web Services, Inc*. Retrieved 20 August 2023 (<https://aws.amazon.com/es/what-is/neural-network/>).
- Anon. 2023j. 'What Is GPT AI? - Generative Pre-Trained Transformers Explained - AWS'. *Amazon Web Services, Inc*. Retrieved 22 August 2023 (<https://aws.amazon.com/what-is/gpt/>).
- Anon. 2023k. 'What Is Natural Language Processing? | IBM'. Retrieved 20 August 2023 (<https://www.ibm.com/topics/natural-language-processing>).
- Anon. 2023l. 'Word Embeddings in NLP: A Complete Guide'. Retrieved 20 August 2023 (<https://www.turing.com/kb/guide-on-word-embeddings-in-nlp>).

- Anon. n.d.-a. 'Chess AI: A Brief History | Built In'. Retrieved 2 September 2023 (<https://builtin.com/artificial-intelligence/chess-ai>).
- Anon. n.d.-b. 'Diferencias entre IA, Machine Learning y Deep Learning'. *HardZone*. Retrieved 23 August 2023 (<https://hardzone.es/tutoriales/rendimiento/diferencias-ia-deep-machine-learning/>).
- Anon. n.d.-c. 'DLAI - Learning Platform Beta'. Retrieved 7 September 2023 (<https://learn.deeplearning.ai/chatgpt-prompt-eng/lesson/1/introduction>).
- Anon. n.d.-d. 'Lichess.Org Open Database'. Retrieved 25 August 2023 (<https://database.lichess.org/#puzzles>).
- Anon. n.d.-e. 'Modelo Transformador Para La Comprensión Del Lenguaje. | Text | TensorFlow'. Retrieved 23 August 2023 (<https://www.tensorflow.org/text/tutorials/transformer?hl=es-419>).
- Anon. n.d.-f. 'What Is Deep Learning? | IBM'. Retrieved 20 August 2023 (<https://www.ibm.com/topics/deep-learning>).
- Bajaj, Prateek. 2018. 'Reinforcement Learning'. *GeeksforGeeks*. Retrieved 20 August 2023 (<https://www.geeksforgeeks.org/what-is-reinforcement-learning/>).
- Barla, Nilesh. 2022. 'The Ultimate Guide to Word Embeddings'. *Neptune.Ai*. Retrieved 20 August 2023 (<https://neptune.ai/blog/word-embeddings-guide>).
- BerenLuthien, Ancalagon. 2016. 'Answer to "What Are the Advantages of ReLU over Sigmoid Function in Deep Neural Networks?"' *Cross Validated*. Retrieved 20 August 2023 (<https://stats.stackexchange.com/a/211359>).
- Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. 'Language Models Are Few-Shot Learners'.
- Copeland, Michael. 2016. 'The Difference Between AI, Machine Learning, and Deep Learning?' *NVIDIA Blog*. Retrieved 20 August 2023 (<https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>).
- DaemonMaker. 2014. 'Answer to "What Are the Advantages of ReLU over Sigmoid Function in Deep Neural Networks?"' *Cross Validated*. Retrieved 20 August 2023 (<https://stats.stackexchange.com/a/126362>).
- Dai, Damai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. 2023a. 'Why Can GPT Learn In-Context? Language Models Implicitly Perform Gradient Descent as Meta-Optimizers'.

- Dai, Damai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. 2023b. ‘Why Can GPT Learn In-Context? Language Models Implicitly Perform Gradient Descent as Meta-Optimizers’. (arXiv:2212.10559).
- Delua, Julianna. 2021. ‘Supervised vs. Unsupervised Learning: What’s the Difference?’ *IBM Blog*. Retrieved 20 August 2023 (<https://www.ibm.com/blog/supervised-vs-unsupervised-learning/>).
- Dhaduk, Hiren. 2023. ‘A Complete Guide to Fine Tuning Large Language Models’. *Simform - Product Engineering Company*. Retrieved 4 September 2023 (<https://www.simform.com/blog/compleateguide-finetuning-llm/>).
- Donges, Niklas. 2023. ‘What Are Recurrent Neural Networks? | Built In’. Retrieved 20 August 2023 (<https://builtin.com/data-science/recurrent-neural-networks-and-lstm>).
- Feng, Xidong, Yicheng Luo, Ziyang Wang, Hongrui Tang, Mengyue Yang, Kun Shao, David Mguni, Yali Du, and Jun Wang. 2023a. ‘ChessGPT: Bridging Policy Learning and Language Modeling’. (arXiv:2306.09200).
- Feng, Xidong, Yicheng Luo, Ziyang Wang, Hongrui Tang, Mengyue Yang, Kun Shao, David Mguni, Yali Du, and Jun Wang. 2023b. ‘ChessGPT: Bridging Policy Learning and Language Modeling’.
- Fiekas, Niklas. 2023. ‘Python-Chess: A Chess Library for Python — Python-Chess 1.10.0 Documentation’. Retrieved 25 August 2023 (<https://python-chess.readthedocs.io/en/latest/>).
- Fu, Yao, Hao Peng, and Tushar Khot. 2022. ‘How Does GPT Obtain Its Ability? Tracing Emergent Abilities of Language Models to Their Sources’.
- Gebhardt, Patrick. n.d. ‘The History of Chess AI’. Retrieved 2 September 2023 (<https://blog.paessler.com/the-history-of-chess-ai>).
- GeeksforGeeks. 2017. ‘Removing Stop Words with NLTK in Python’. *GeeksforGeeks*. Retrieved 20 August 2023 (<https://www.geeksforgeeks.org/removing-stop-words-nltk-python/>).
- Gokultechninza. 2023a. ‘How Should AI Systems Behave and Who Should Decide? - According to OpenAI’. *Medium*. Retrieved 22 August 2023 (<https://medium.com/@gokultechninza/how-should-ai-systems-behave-and-who-should-decide-according-to-openai-e1683a20fd34>).
- Gokultechninza. 2023b. ‘How Should AI Systems Behave and Who Should Decide? - According to OpenAI’. *Medium*. Retrieved 22 August 2023 (<https://medium.com/@gokultechninza/how-should-ai-systems-behave-and-who-should-decide-according-to-openai-e1683a20fd34>).
- GraphEverywhere, Equipo. 2019. ‘Machine Learning | Qué es, tipos, ejemplos y cómo implementarlo’. *GraphEverywhere*. Retrieved 20 August 2023 (<https://www.grapheverywhere.com/machine-learning-que-es-tipos-ejemplos-y-como-implementarlo/>).

- Greyling, Cobus. 2023. 'Example Code & Implementation Considerations For GPT 3.5 Turbo, ChatML & Whisper'. *Medium*. Retrieved 22 August 2023 (<https://cobusgreyling.medium.com/example-code-implementation-considerations-for-gpt-3-5-turbo-chatml-whisper-e61f8703c5db>).
- Gupta, Mohit. 2019. 'NLP | How Tokenizing Text, Sentence, Words Works'. *GeeksforGeeks*. Retrieved 20 August 2023 (<https://www.geeksforgeeks.org/nlp-how-tokenizing-text-sentence-words-works/>).
- Gupta, Shlok. 2023. 'Rules of Inference in AI'. *Scaler Topics*. Retrieved 21 August 2023 (<https://www.scaler.com/topics/inference-rules-in-ai/>).
- Jain, Anupama. 2022. 'A Complete Guide To Understand How Chess Engines Work (2022)'. Retrieved 2 September 2023 (<https://squareoffnow.com/blog/how-chess-engines-work/>).
- Kapronczay, Mór. 2023. 'A Beginner's Guide to Language Models | Built In'. Retrieved 20 August 2023 (<https://builtin.com/data-science/beginners-guide-language-models>).
- Kerner, Sean Michael. 2023. 'What Is a Large Language Model (LLM)? – TechTarget Definition'. *WhatIs.Com*. Retrieved 23 August 2023 (<https://www.techtarget.com/whatis/definition/large-language-model-LLM>).
- Kumar, Ravish. 2021. 'Supervised, Unsupervised, and Semi-Supervised Learning'. *EnjoyAlgorithms*. Retrieved 23 August 2023 (<https://medium.com/enjoy-algorithm/supervised-unsupervised-and-semi-supervised-learning-64ee79b17d10>).
- Kuyper, Dan. 2023. 'Iterative Prompt Development'. *Dan Kuyper*. Retrieved 21 August 2023 (<https://dankuyper.com/iterative-prompt-development/>).
- Ltd, Arm. n.d. 'What Is AI Inference'. *Arm | The Architecture for the Digital World*. Retrieved 21 August 2023 (<https://www.arm.com/glossary/ai-inference>).
- Maatta, Teemu. 2023. 'Chat Markup Language (ChatML)'. *Medium*. Retrieved 22 August 2023 (<https://tmmtt.medium.com/chat-markup-language-chatml-35767c2c69a1>).
- Mahendra, Sanksshep. 2022. 'How Do Chess Engines Work?' *Artificial Intelligence +*. Retrieved 3 September 2023 (<https://www.aiplusinfo.com/blog/how-do-chess-engines-work/>).
- Mandour, Ahmed. 2023. 'GPT-3.5 Model Architecture'. *OpenGenus IQ: Computing Expertise & Legacy*. Retrieved 20 August 2023 (<https://iq.opengenus.org/gpt-3-5-model/>).
- Menzli, Amal. 2022. 'Tokenization in NLP: Types, Challenges, Examples, Tools'. *Neptune.Ai*. Retrieved 20 August 2023 (<https://neptune.ai/blog/tokenization-in-nlp>).
- OpenAI. 2022. 'Introducing ChatGPT'. Retrieved 22 August 2023 (<https://openai.com/blog/chatgpt>).
- OpenAI. 2023a. 'How Should AI Systems Behave, and Who Should Decide?' Retrieved 22 August 2023 (<https://openai.com/blog/how-should-ai-systems-behave>).

OpenAI. 2023b. 'OpenAI Platform'. Retrieved 20 August 2023 (<https://platform.openai.com>).

OpenAI. 2023c. 'OpenAI Platform'. Retrieved 22 August 2023 (<https://platform.openai.com>).

Oracle, Experiencia. 2018a. 'Diferencias Entre La Inteligencia Artificial y El Machine Learning'. *Medium*. Retrieved 20 August 2023 (<https://medium.com/@experiencia18/diferencias-entre-la-inteligencia-artificial-y-el-machine-learning-f0448c503cd4>).

Oracle, Experiencia. 2018b. 'Diferencias Entre La Inteligencia Artificial y El Machine Learning'. *Medium*. Retrieved 20 August 2023 (<https://medium.com/@experiencia18/diferencias-entre-la-inteligencia-artificial-y-el-machine-learning-f0448c503cd4>).

Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. 'Training Language Models to Follow Instructions with Human Feedback'.

Peng, Baolin, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023a. 'Instruction Tuning with GPT-4'.

Peng, Baolin, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023b. 'Instruction Tuning with GPT-4'. (arXiv:2304.03277).

Prakash, Amit. 2023. 'GPT-4 vs GPT-3.5: A Full Breakdown of The Differences'. *ThoughtSpot*. Retrieved 22 August 2023 (<https://www.thoughtspot.com/data-trends/ai/gpt-4-vs-gpt-3-5>).

Pykes, Kurtis. 2023. 'Stemming and Lemmatization in Python'. Retrieved 20 August 2023 (<https://www.datacamp.com/tutorial/stemming-lemmatization-python>).

Qwiklabs. n.d.-a. 'Introduction to Generative AI | Google Cloud Skills Boost'. *Qwiklabs*. Retrieved 20 August 2023 (https://www.cloudskillsboost.google/course_sessions/3227266/video/382759).

Qwiklabs. n.d.-b. 'Introduction to Large Language Models | Google Cloud Skills Boost'. *Qwiklabs*. Retrieved 20 August 2023 (https://www.cloudskillsboost.google/course_sessions/3268094/video/379143).

Raventós, Allan, Mansheej Paul, Feng Chen, and Surya Ganguli. 2023. 'Pretraining Task Diversity and the Emergence of Non-Bayesian in-Context Learning for Regression'.

Reznikov, Ivan. 2022. 'How Good Is ChatGPT at Playing Chess? (Spoiler: You'll Be Impressed)'. *Medium*. Retrieved 25 August 2023 (<https://medium.com/@ivanreznikov/how-good-is-chatgpt-at-playing-chess-spoiler-youll-be-impressed-35b2d3ac024a>).

Ringa Tech, dir. 2021. *Tu Primera Red Neuronal En Python y Tensorflow*.

RockTheStar. 2019. 'What Are the Advantages of ReLU over Sigmoid Function in Deep Neural Networks?' *Cross Validated*. Retrieved 20 August 2023 (<https://stats.stackexchange.com/q/126238>).

Rouse, Margaret. 2023. 'Zero-Shot, One-Shot, Few-Shot Learning'. *Techopedia*. Retrieved 22 August 2023 (<https://www.techopedia.com/definition/34949/zero-shot-one-shot-few-shot-learning>).

Rowe, Walker. n.d. 'What Is a Neural Network? An Introduction with Examples'. *BMC Blogs*. Retrieved 23 August 2023 (<https://www.bmc.com/blogs/neural-network-introduction/>).

Salamone, Luke. 2021. 'What Is Temperature in NLP? 🐶'. *Luke Salamone's Blog*. Retrieved 21 August 2023 (<https://lukesalamone.github.io/posts/what-is-temperature/>).

shristikotaiah. 2020. 'Word Embeddings in NLP'. *GeeksforGeeks*. Retrieved 20 August 2023 (<https://www.geeksforgeeks.org/word-embeddings-in-nlp/>).

Srivastava, Aarohi, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ambrose Slone, Ameet Rahane, Anantharaman S. Iyer, Anders Andreassen, Andrea Madotto, Andrea Santilli, Andreas Stuhlmüller, Andrew Dai, Andrew La, Andrew Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakaş, B. Ryan Roberts, Bao Sheng Loe, Barret Zoph, Bartłomiej Bojanowski, Batuhan Özyurt, Behnam Hedayatnia, Behnam Neyshabur, Benjamin Inden, Benno Stein, Berk Ekmekci, Bill Yuchen Lin, Blake Howald, Bryan Orinion, Cameron Diao, Cameron Dour, Catherine Stinson, Cedrick Argueta, César Ferri Ramírez, Chandan Singh, Charles Rathkopf, Chenlin Meng, Chitta Baral, Chiyu Wu, Chris Callison-Burch, Chris Waites, Christian Voigt, Christopher D. Manning, Christopher Potts, Cindy Ramirez, Clara E. Rivera, Clemencia Siro, Colin Raffel, Courtney Ashcraft, Cristina Garbacea, Damien Sileo, Dan Garrette, Dan Hendrycks, Dan Kilman, Dan Roth, Daniel Freeman, Daniel Khashabi, Daniel Levy, Daniel Moseguí González, Danielle Perszyk, Danny Hernandez, Danqi Chen, Daphne Ippolito, Dar Gilboa, David Dohan, David Drakard, David Jurgens, Debajyoti Datta, Deep Ganguli, Denis Emelin, Denis Kleyko, Deniz Yuret, Derek Chen, Derek Tam, Dieuwke Hupkes, Diganta Misra, Dilyar Buzan, Dimitri Coelho Mollo, Diyi Yang, Dong-Ho Lee, Dylan Schrader, Ekaterina Shutova, Ekin Dogus Cubuk, Elad Segal, Eleanor Hagerman, Elizabeth Barnes, Elizabeth Donoway, Ellie Pavlick, Emanuele Rodola, Emma Lam, Eric Chu, Eric Tang, Erkut Erdem, Ernie Chang, Ethan A. Chi, Ethan Dyer, Ethan Jerzak, Ethan Kim, Eunice Engefu Manyasi, Evgenii Zheltonozhskii, Fanyue Xia, Fatemeh Siar, Fernando Martínez-Plumed, Francesca Happé, Francois Chollet, Frieda Rong, Gaurav Mishra, Genta Indra Winata, Gerard de Melo, Germán Kruszewski,

Giambattista Parascandolo, Giorgio Mariani, Gloria Wang, Gonzalo Jaimovitch-López, Gregor Betz, Guy Gur-Ari, Hana Galijasevic, Hannah Kim, Hannah Rashkin, Hannaneh Hajishirzi, Harsh Mehta, Hayden Bogar, Henry Shevlin, Hinrich Schütze, Hiromu Yakura, Hongming Zhang, Hugh Mee Wong, Ian Ng, Isaac Noble, Jaap Jumelet, Jack Geissinger, Jackson Kernion, Jacob Hilton, Jaehoon Lee, Jaime Fernández Fisac, James B. Simon, James Koppel, James Zheng, James Zou, Jan Kocoń, Jana Thompson, Janelle Wingfield, Jared Kaplan, Jarema Radom, Jascha Sohl-Dickstein, Jason Phang, Jason Wei, Jason Yosinski, Jekaterina Novikova, Jelle Bosscher, Jennifer Marsh, Jeremy Kim, Jeroen Taal, Jesse Engel, Jesujoba Alabi, Jiacheng Xu, Jiaming Song, Jillian Tang, Joan Waweru, John Burden, John Miller, John U. Balis, Jonathan Batchelder, Jonathan Berant, Jörg Frohberg, Jos Rozen, Jose Hernandez-Orallo, Joseph Boudeman, Joseph Guerr, Joseph Jones, Joshua B. Tenenbaum, Joshua S. Rule, Joyce Chua, Kamil Kanclerz, Karen Livescu, Karl Krauth, Karthik Gopalakrishnan, Katerina Ignatyeva, Katja Markert, Kaustubh D. Dhole, Kevin Gimpel, Kevin Omondi, Kory Mathewson, Kristen Chiafullo, Ksenia Shkaruta, Kumar Shridhar, Kyle McDonell, Kyle Richardson, Laria Reynolds, Leo Gao, Li Zhang, Liam Dugan, Lianhui Qin, Lidia Contreras-Ochando, Louis-Philippe Morency, Luca Moschella, Lucas Lam, Lucy Noble, Ludwig Schmidt, Luheng He, Luis Oliveros Colón, Luke Metz, Lütfi Kerem Şenel, Maarten Bosma, Maarten Sap, Maartje ter Hoeve, Maheen Farooqi, Manaal Faruqui, Mantas Mazeika, Marco Baturan, Marco Marelli, Marco Maru, Maria Jose Ramírez Quintana, Marie Tolkiehn, Mario Giulianelli, Martha Lewis, Martin Potthast, Matthew L. Leavitt, Matthias Hagen, Mátyás Schubert, Medina Orduna Baitemirova, Melody Arnaud, Melvin McElrath, Michael A. Yee, Michael Cohen, Michael Gu, Michael Ivanitskiy, Michael Starritt, Michael Strube, Michał Śwędrowski, Michele Bevilacqua, Michihiro Yasunaga, Mihir Kale, Mike Cain, Mimeo Xu, Mirac Suzgun, Mitch Walker, Mo Tiwari, Mohit Bansal, Moin Aminnaseri, Mor Geva, Mozhdah Gheini, Mukund Varma T, Nanyun Peng, Nathan A. Chi, Nayeon Lee, Neta Gur-Ari Krakover, Nicholas Cameron, Nicholas Roberts, Nick Doiron, Nicole Martinez, Nikita Nangia, Niklas Deckers, Niklas Muennighoff, Nitish Shirish Keskar, Niveditha S. Iyer, Noah Constant, Noah Fiedel, Nuan Wen, Oliver Zhang, Omar Agha, Omar Elbaghdadi, Omer Levy, Owain Evans, Pablo Antonio Moreno Casares, Parth Doshi, Pascale Fung, Paul Pu Liang, Paul Vicol, Pegah Alipoormolabashi, Peiyuan Liao, Percy Liang, Peter Chang, Peter Eckersley, Phu Mon Htut, Pinyu Hwang, Piotr Milkowski, Piyush Patil, Pouya Pezeshkpour, Priti Oli, Qiaozhu Mei, Qing Lyu, Qinlang Chen, Rabin Banjade, Rachel Etta Rudolph, Raefer Gabriel, Rahel Habacker, Ramon Risco, Raphaël Millière, Rhythm Garg, Richard Barnes, Rif A. Saurous, Riku Arakawa, Robbe Raymaekers, Robert Frank, Rohan Sikand, Roman Novak, Roman Sitelew, Ronan LeBras, Rosanne Liu, Rowan Jacobs, Rui Zhang, Ruslan Salakhutdinov, Ryan Chi, Ryan Lee, Ryan Stovall, Ryan Teehan, Rylan Yang, Sahib Singh, Saif M. Mohammad, Sajant Anand, Sam Dillavou, Sam Shleifer, Sam Wiseman, Samuel Gruetter, Samuel R. Bowman, Samuel S. Schoenholz, Sanghyun Han, Sanjeev Kwatra, Sarah A. Rous, Sarik Ghazarian, Sayan Ghosh, Sean Casey, Sebastian Bischoff, Sebastian Gehrmann, Sebastian Schuster, Sepideh Sadeghi, Shadi Hamdan, Sharon Zhou, Shashank Srivastava, Sherry Shi, Shikhar Singh, Shima Asaadi, Shixiang Shane Gu, Shubh Pachchigar, Shubham Toshniwal, Shyam Upadhyay, Shyamolima, Debnath, Siamak Shakeri, Simon Thormeyer, Simone Melzi, Siva Reddy, Sneha Priscilla Makini, Soo-Hwan Lee, Spencer Torene, Sriharsha Hatwar, Stanislas Dehaene, Stefan Divic, Stefano Ermon, Stella Biderman, Stephanie Lin, Stephen Prasad, Steven T. Piantadosi, Stuart

M. Shieber, Summer Mishnerghi, Svetlana Kiritchenko, Swaroop Mishra, Tal Linzen, Tal Schuster, Tao Li, Tao Yu, Tariq Ali, Tatsu Hashimoto, Te-Lin Wu, Théo Desbordes, Theodore Rothschild, Thomas Phan, Tianle Wang, Tiberius Nkinyili, Timo Schick, Timofei Kornev, Titus Tunduny, Tobias Gerstenberg, Trenton Chang, Trishala Neeraj, Tushar Khot, Tyler Shultz, Uri Shaham, Vedant Misra, Vera Demberg, Victoria Nyamai, Vikas Raunak, Vinay Ramasesh, Vinay Uday Prabhu, Vishakh Padmakumar, Vivek Srikumar, William Fedus, William Saunders, William Zhang, Wout Vossen, Xiang Ren, Xiaoyu Tong, Xinran Zhao, Xinyi Wu, Xudong Shen, Yadollah Yaghoobzadeh, Yair Lakretz, Yangqiu Song, Yasaman Bahri, Yejin Choi, Yichi Yang, Yiding Hao, Yifu Chen, Yonatan Belinkov, Yu Hou, Yufang Hou, Yuntao Bai, Zachary Seid, Zhuoye Zhao, Zijian Wang, Zijie J. Wang, Zirui Wang, and Ziyi Wu. 2023. 'Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models'.

Team (CHESScom), Chess.com. 2019. 'Computer Chess Engines: A Quick Guide'. *Chess.Com*. Retrieved 2 September 2023 (<https://www.chess.com/article/view/computer-chess-engines>).

Team, IBM Data and AI. 2023. 'AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?' *IBM Blog*. Retrieved 20 August 2023 (<https://www.ibm.com/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks/>).

Terrasi, Vincent. 2023. 'GPT-4: How Is It Different From GPT-3.5?' Retrieved 22 August 2023 (<https://www.searchenginejournal.com/gpt-4-vs-gpt-3-5/482463/#close>).

Tianyu, Gao. 2021. 'Prompting: Better Ways of Using Language Models for NLP Tasks'. *The Gradient*. Retrieved 21 August 2023 (<https://thegradient.pub/prompting/>).

Varma, Satish. 2023. 'Use of Role's System/User/Assistant in ChatGPT API'. *Java Tutorials*. Retrieved 22 August 2023 (<https://javabydeveloper.com/use-of-roles-system-user-assistant-in-chatgpt-api/>).

Varshney, Tanay, and Annie Suria. 2023. 'An Introduction to Large Language Models: Prompt Engineering and P-Tuning'. *NVIDIA Technical Blog*. Retrieved 21 August 2023 (<https://developer.nvidia.com/blog/an-introduction-to-large-language-models-prompt-engineering-and-p-tuning/>).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. 'Attention Is All You Need'. in *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc.

Vicente, Fran Ramírez. n.d. 'Las matemáticas del Machine Learning: Funciones de activación'. *Telefónica Tech*. Retrieved 6 September 2023 (<https://telefonicatech.com/blog/las-matematicas-del-machine-learning-funciones-de-activacion>).

Wolff, Rachel. 2020. 'What Is Natural Language Processing'. *MonkeyLearn Blog*. Retrieved 20 August 2023 (<https://monkeylearn.com/blog/what-is-natural-language-processing/>).

Yothment, Jacob. 2023. 'How AI Powers Chess Engines and Creates Grandmasters'. *Pure Storage Blog*. Retrieved 3 September 2023 (<https://blog.purestorage.com/perspectives/how-ai-powers-chess-engines-and-creates-grandmasters/>).

Zola, Andrew. 2023. 'How to Play Chess Using a GPT-2 Model | HackerNoon'. Retrieved 24 August 2023 (<https://hackernoon.com/how-to-play-chess-using-a-gpt-2-model-c9323wwi>).