The technical concepts and training approaches that power Generative Pre-trained Transformer (GPT) models -- some preliminary observations

Arnold F. Shapiro[1]

Pennsylvania State University, 361 Business Administration Bldg., University Park, PA 16802, USA

Abstract

Generative Pre-trained Transformer (GPT) technology is increasingly becoming a part of the insurtech dialogues, where GPT models have the potential to significantly enhance the efficiency and accuracy of actuarial work. See, for example, Balona (2024), Brown et al (2023), Carlin and Mathys (2024), Jones and McLeod (2023), and du Preez et al (2024). In spite of such studies, empirical evidence seems to indicate that researchers who have been exposed to GPT models like Chat GPT, or one of the popular apps, Copilot (Microsoft) or Gemini (Google), usually do not know how and why these GPT models work.

Assuming this to generally be the case, the goal of this research is to help alleviate this situation. Based on my understanding of the GPT reports, articles and presentations that I have reviewed, I am conducting a study that investigates and documents the technical concepts and training approaches that power GPT models.

This article presents some preliminary observations from that study. The specific topics discussed are itemize in the keywords section.

The article concludes with a commentary.

Keywords: Generative Pre-trained Transformer (GPT), data collection, preprocessing, pre-training, fine-tuning, reinforcement learning, human-in-the-loop, tokenization, attention, positional encoding, vector representation, embedding, residual connection

---

[1] The support of the Risk Management Research Center at the Penn State University is gratefully acknowledged.

1. Introduction

This article begins with a simple query to GPT involving the formula for the present value of a whole life insurance policy for a life aged x. The GPT response sets the stage and the context for the rest of the article, where we focus on preliminary observations regarding three main topics: the GPT model, how GPT processes a prompt, and the GPT transformer/decoder.

The subtopics covered under the "GPT model" include: accessing a large dataset of existing text, preprocess the text data, choose a machine learning algorithm, pre-training, fine-tuning the trained model, testing the model, and refining the model.

The subtopics covered under "how GPT processes a prompt" include: tokenization, context processing, knowledge retrieval & pattern matching, predicting the next token, response refinement, output formatting, and the output.

Finally, topics covered under "the GPT transformer/decoder" include: input embedding, positional encoding, simple visualization of the embedding-positional encoding process, residual connection, multi-head attention, attention, the hidden states of GPT, the query (Q), key (K) and value (V) matrices, the relationship between Q and K, the attention mechanism, multi-head attention representation, masked multi-head attention, add & norm, feed forward neural networks, and output probabilities.

The article ends with a commentary.

2. A GPT Example

Prompt: What is the formula for the present value of a whole life policy for a life aged x

Copilot's response:[2]

> "The present value of a whole life insurance policy for a life aged x can be calculated using actuarial notation.
>
> The formula [using linear notation] for the actuarial present value (APV) of a [continuous] whole life insurance policy is given by:

[2] In its response, Copilot cited: https://hartman.byu.edu/docs/344Files/Stat344_Chapter4.pdf and https://users.stat.ufl.edu/~rrandles/sta4930/4930lectures/chapter4/chapter4R.pdf

A_x = \int_0^\infty v^t \cdot _tp_x \cdot \mu_{x+t} \, dt

where:

A_x is the actuarial present value of the whole life insurance policy for a life aged x.
v is the discount factor, which is v = \frac{1}{1+i}, where i is the effective annual interest rate.

_tp_x is the probability that a life aged x will survive to age x+t.

\mu_{x+t} is the force of mortality at age x+t."

3. How did GPT do that?

How did GPT come up with that response?
How did GPT respond with what appears to be a human-like response?
How did GPT do it so quickly?

For that matter:

What is GPT?
How does GPT work?

These are the questions that will be addressed in this article.

Our underlying focus will be on 3 main topics:

1. The GPT model
2. How GPT processes a prompt
3. The GPT Decoder

4. What are GPTs?[3]

GPTs, or Generative Pre-trained Transformers, are a family of neural network models that are based on the transformer architecture (see §7) introduced by the Google researchers, Vaswani et al (2017), and implemented by the Open AI researchers, Radford et al (2018).

[3] References for this section include Brodigan and O'Dwyer (2023) and Howarth (2025).

GPT is a large language model (LLM) that can interpret and generate human-like text based on patterns learned from vast amounts of data.

Their distinguishing characteristic is that the GPT model, in contrast to previous models, is better able to interpret and apply the context in which words and expressions are used and selectively focus on different portions of the input. In particular, it can focus on relevant words or phrases that it perceives as more important to the outcome.

5. The GPT model[4]

This section provides an overview of the GPT model. Figure 1 is a flowchart of the various steps in that process.

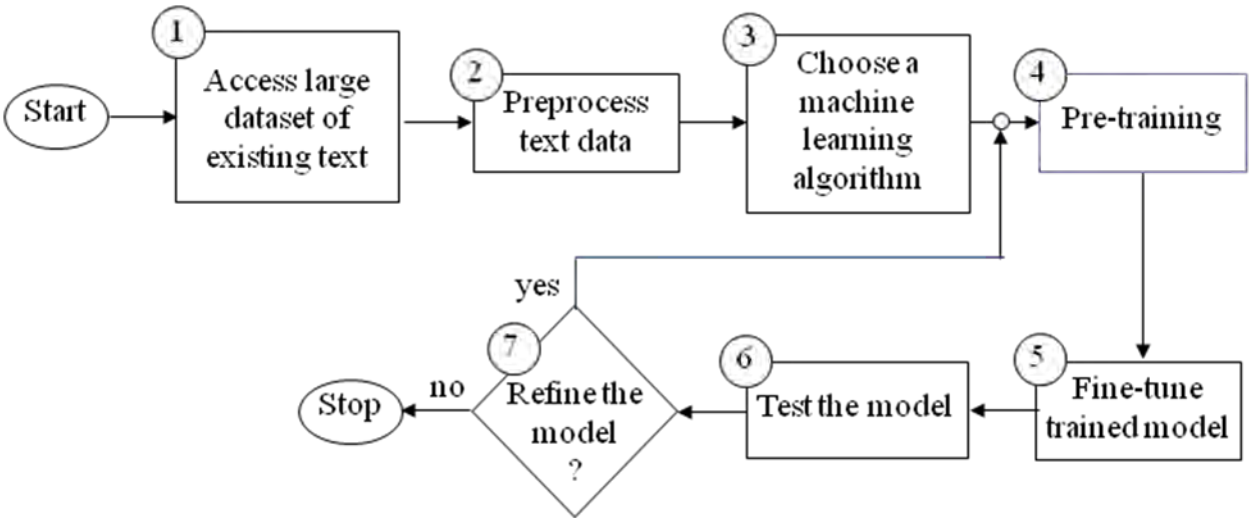


Figure 1: Flowchart of the GPT model[5]

The rest of this section elaborates on each of these steps.

Step 1: Access a large dataset of relevant existing text

GPTs are trained on massive amounts of publicly available text, books, articles, and websites.

[4] References for this section include Barkovska et al (2024) and Huang et al (2023).

[5] Adapted from Huang et al (2023:50-52).

However, unlike search engines or relational databases, GPT models are not dependent on traditional databases, and do not retain exact copies of its training data. Instead, as discussed in the following pages, they learn patterns, relationships, and linguistic structures.

Insofar as an actuarial perspective, to create a large language model that can respond to actuarial prompts, the dataset might include actuarial books, reports, articles and regulatory constraints.

Step 2: Preprocess the text data[6]

In order for the text data to be used for training a machine learning algorithm, it needs to be preprocessed. Among the sub-steps normally involved are:

Tokenization, under which text is broken down into smaller units called tokens. This allows GPT to interpret the structure of the text and to predict the next token based on patterns it has learned. More on this later.

Normalization, which involves converting all text to lowercase letters and removing characters and punctuation that do not convey important information.

Removal of stop words, which includes the elimination of common words such as "the" or "and" from the text. They do not convey much additional meaning and can slow down the training.

The goal of these sub-steps is to help provide a structured, noise-free dataset for the subsequent stages of analysis and classification.

Step 3: Choose a ML algorithm[7]

A number of machine learning (ML) algorithms can be used to create a language model; three of the popular ones are:

Recurrent neural network (RNN), which are a type of neural network designed to process sequential data, by recursively applying the same set of weights to each element of the sequence.

RNNs suffer the vanishing gradient problem (VGP). [8]

---

[6] References for this section include Barkovska et al (2024) and Huang et al (2023).

[7] References for this section include Huang et al (2023) and Sengupta (2023).

[8] Gradient descent is an optimization algorithm for finding the minimum of a function by iteratively adjusting the parameters in the direction of steepest descent. The vanishing gradient problem, which is a common issue with respect to deep neural networks, where gradients become extremely small as they are propagated back through the

Long Short-Term Memory (LSTM), which are a type of recurrent neural network that can learn to process and generate sequences of data.

LSTM Networks resolve the VGP, but are relatively slow.

Transformers, which are a neural network architecture designed to handle sequential data, like text, using a mechanisms that weighs the importance of different parts of the input.

Transformers resolves both the foregoing issues.

Our focus is on Transformers, as in Generative Pre-trained Transformers (GPT).

Step 4: Pre-training[9]

Once the machine learning algorithm is chosen, it can train on the preprocessed text data. During training, the algorithm first learns to recognize patterns in the text and then uses these patterns to generate new text. For subsequent GPT applications, the algorithm will be trained to predict the next word in a sequence of words given the previous words.

More on this topic later.

Pre-training limitations

Wu et al (2024) cautions that although models like GPT-4 have been trained on immense corpora of public text, their knowledge is frozen at the time of training. As a result, they lack awareness of current events, proprietary company data, and user-specific contexts.

Core Pre-training Concepts

Before continuing, we stop to consider three core pre-training concepts: tokenization, embedding, and embedding matrix.

---

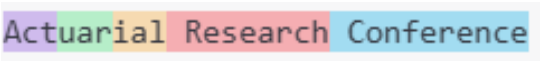
layers during training. This results in slow convergence and poor performance, particularly in the initial layers of the network.

[9] References for this section include Huang et al (2023).

Tokenization

In the context of GPT models, tokens are the basic units of text that the model processes. They can represent words, subwords, or special characters. Each token is assigned a unique identifier (ID), while phrases are tokenized into multiple tokens, each with its own unique identifier.

For example, for the phrase “Actuarial Research Conference”, the 5 GPT-4 tokens are:[10]



and the 5 GPT-4 token IDs are:

“Act" → 2471
“uar" → 19253
“ial" → 532
“Research” → 8483
“Conference" → 15217

These token IDs are used internally by the model to process and generate text.

Embedding[11]

Embeddings allow one to manipulate language data using a mathematical approach.

Specifically, embeddings refer to the representation of words, phrases, or sentences as vectors in a continuous high-dimensional space, and derived from the underlying semantic or contextual information of the text. As such, they capture the meaning, relationships, and semantic similarity between words or textual elements.

In GPT models, embeddings nearest to a given token, like "actuarial," are determined by the similarity of their high-dimensional vectors. These vectors capture the semantic meaning of the tokens, which allows the model to interpret and process text more effectively. To find the nearest embeddings, we normally use a measure like cosine similarity, which calculates the angle between two vectors in the embedding space.

[10] See https://platform.openai.com/tokenizer

[11] References for this subsection include Balona (2024), Barkovska et al (2024), Sanderson (2024), and Wolfram (2024).

For example, tokens with respect to "actuarial analysis"[12] and "financial engineering" would have embeddings close to one another because they share similar semantic meanings. As depicted in Figure 2, these embeddings are clustered together in the high-dimensional space, allowing the model to interpret and generate text based on these relationships.

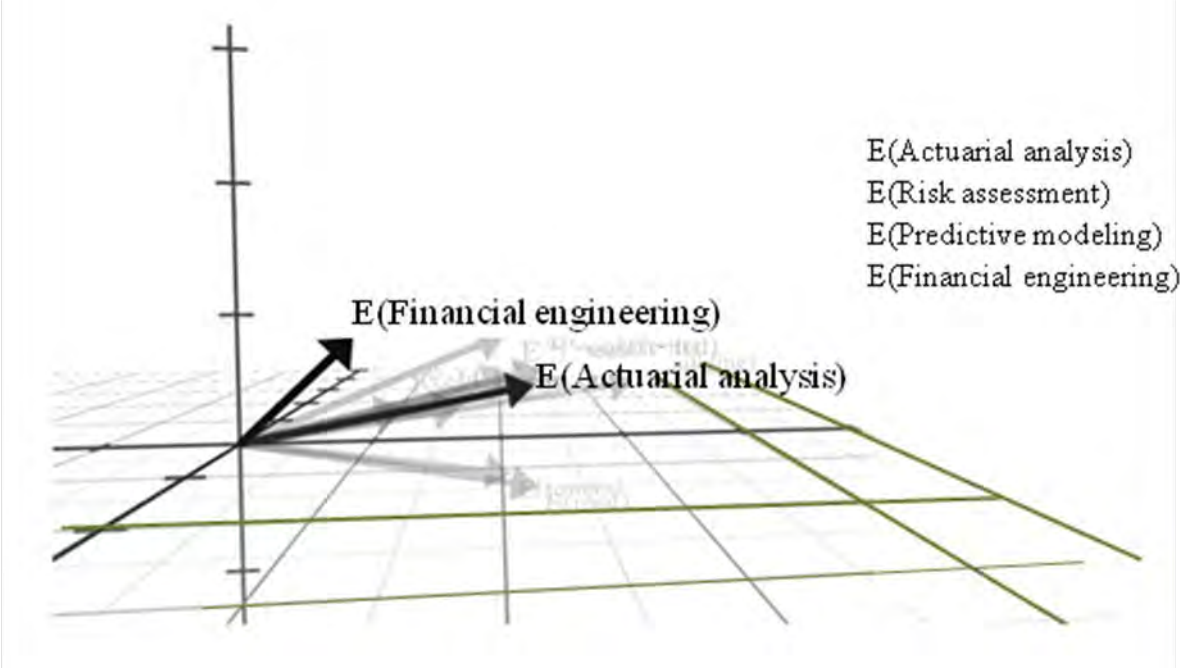


Figure 2: Embedding in the vicinity of E(actuarial analysis)[13]

Wolfram (2024: 42) had an interesting take on this issue:

> Rather than directly trying to characterize "what image is near what other image", we instead consider a well-defined task (in this case digit recognition) for which we can get explicit training data—then use the fact that in doing this task the neural net implicitly has to make what amount to "nearness decisions". So instead of us ever explicitly having to talk about "nearness of images" we're just talking about the concrete question of what digit an image represents, and then we're "leaving it to the neural net" to implicitly determine what that implies about "nearness of images".

[12] For illustrative purposes, "actuarial analysis" and "financial engineering" are referred to as tokens. In practice, of course, they each might be split into two or more tokens, depending on the tokenizer.

[13] Adapted from Sanderson (2024).

An Embedding matrix

An embedding matrix helps convert tokens into numerical vectors that the model can understand and work with. An example, of which, is shown in Figure 3. As indicated, using the GPT-3 numbers, the vocabulary size, which refers to the number of unique tokens that the model can recognize and generate, is 50,257, while its embedding dimension, which represents each token as a vectors in a continuous high-dimensional space, has a fixes dimension of 12,288. Thus, the embedding matrix size of GPT-3 is 617,558,016 weights.

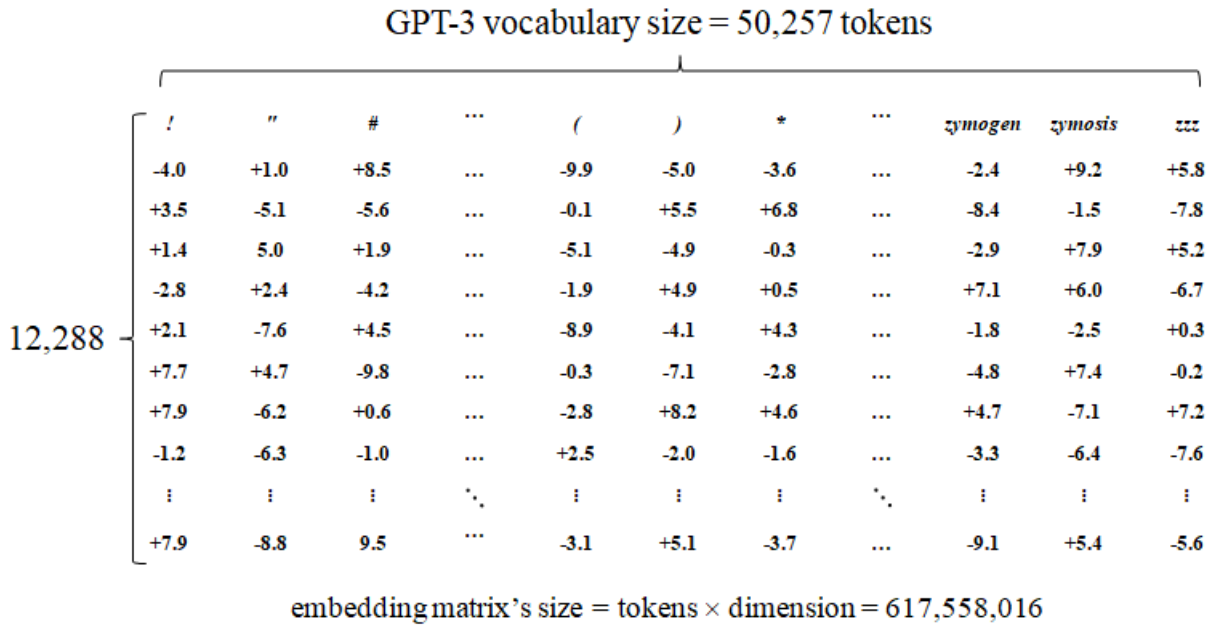


Figure 3: Embedding matrix[14]

Step 5: Fine-tune the pre-trained model

After pre-training the machine learning algorithm, the trained model likely will need to be fine-tuned to improve its performance. This can involve several sub-steps, including:

Adjusting the model's hyperparameters,[15] such as the learning rate, acceleration rate, or the number of hidden layers.

Adding additional training data on task-specific or domain-specific datasets.

[14] Adapted from Sanderson (2024).

[15] Hyperparameters are settings that control how the algorithm is trained.

Regularization, which helps prevent over-fitting.[16] Common approaches are:

Adding penalty terms to the loss function, and

Dropout, which involves setting a subset of neurons to zero during a forward pass.

Fine-tuning limitations

Soudani et al (2024) cautions that, while effective in enhancing domain alignment, fine-tuning is both computationally expensive and inflexible. It requires large volumes of labeled data, extensive computing resources, as well as expertise and time.

Step 6: Test the model[17]

Once the model is fine-tuned, it needs to be tested and evaluated. One way of doing this is by generating sample text and comparing it to the original training data. The test is whether the generated text is of high quality and matches the desired language or style.

Step 7: Refine the model?

If the generated text does not meet the desired standards in terms of quality and language/style, steps 4 and 5 need to be iterated. This can involve reinforcement learning with human feedback (RLHF), where human expertise is used to train the model further.[18]

[16] Over-fitting occurs when the model memorizes the training data instead of learning to generate new text.

[17] References for this section include Huang et al (2023).

[18] 60 Minutes (2024-11-24) discussed RLHF in India, where wages were $2/day.

6. How GPT processes a prompt

This section provides an overview of how GPT processes a prompt. When a user inputs a prompt to GPT, whether a question, command, or statement, GPT processes it through the series of steps shown in Figure 4.

At the core of GPT is a transformer architecture (see §7) that converts the prompt into tokens. These tokens are then mapped into high-dimensional vectors, which allows the model to interpret their meaning in context.

GPT weighs the relevance of each token relative to others, enabling it to capture nuanced relationships and dependencies across the input. It then generates a response by predicting the next most likely token, one at a time, based on the learned patterns from its training data (see pre-training and fine-tuning). This prediction is guided by probabilities and refined through layers of neural computation, resulting in coherent and contextually appropriate output.

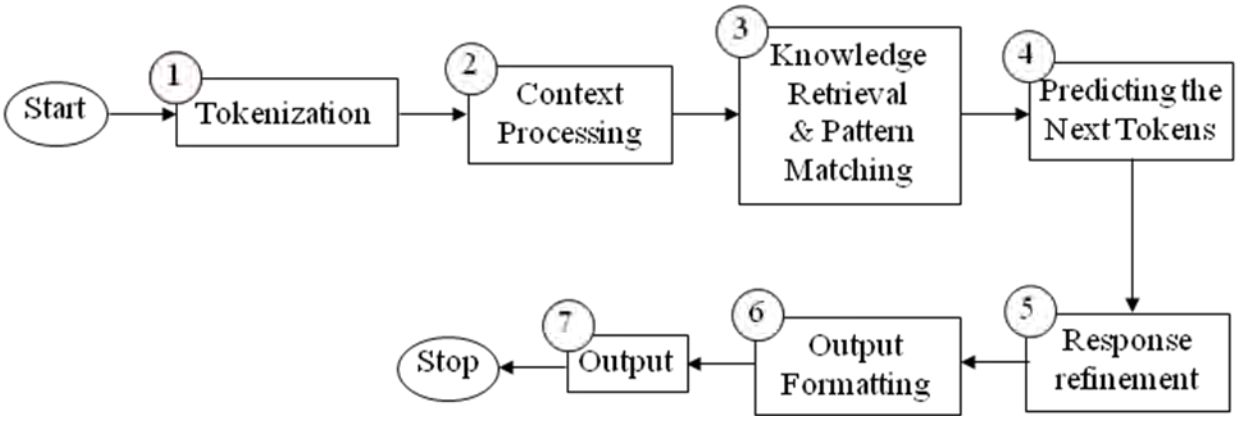


Figure 4: Flowchart of how GPT processes a prompt

An overview of these steps follows.

Step 1: Tokenization

Given a prompt, the first step is for GPT to convert the text to tokens. See page 7. Then the embedding matrix is used to map each token to a unique numerical representation that can be processed by the model.

Step 2: Context Processing

If the prompt is a follow-up from a conversation, GPT maintains relevance and coherence by incorporating the previous interactions. These previous conversations are incorporated as components of the input prompt. This helps promote continuity with previous exchanges.

Step 3: Knowledge Retrieval & Pattern Matching

GPT does not browse the internet in real-time in responding to a query. It relies on its extensive training dataset for knowledge retrieval and pattern matching. To this end, GPT incorporates the most relevant concepts, facts, or explanations, given its stored knowledge.

Step 4: Predicting the Next Tokens

As mentioned previously, GPT generates responses based on probabilities. Thus, it does not retrieve answers directly from stored information. Instead, GPT predicts the (most) likely next token given the previous tokens in the sequence. Moreover, it builds sentences incrementally, refining logical consistency, relevance, and flow, as it goes along.

Step 5: Response refinement

In order that responses be fluent and readable, the model strives for logical consistency. For example, repetitive phrases and/or contradictory statements are removed.

Step 6: Output Formatting

When appropriate for clarity, GPT uses bullet points, paragraphs, or numbered lists to configure responses. Also, in response to the wording of users, the tone or level of detail may be adjusted.

Step 7: Output

GPT presents its generated response, which allows for user interaction, and follow-up questions by the user are answered dynamically. Since GPT responses reflect the probability of their occurrence, they may vary slightly, even with the same query, depending on phrasing, context, and prior messages.

7. Transformers[19]

As mentioned previously, GPT uses a deep learning model called a transformer that processes words in relation to each other, which allows it to interpret context and generate coherent responses.

[19] Merritt (2022) gives some interesting insights regarding the reception transformers received when they were first introduced.

Now, the basic structure of the original transformer architecture was made up of two modules, an encoder and a decoder, where the encoder processes input sequences and the decoder generating output sequences. However, GPT focuses on the decoder portion, since it is designed for uni-directional language modeling tasks, which means it intends to predict the next word in a sequence based on the context provided by the preceding words. As a result, GPT does not require an encoder to process input text.

8. The GPT Decoder[20]

The GPT decoder is an autoregressive text generator, since it predicts the next word based on previous words. To accomplish this, GPT relies on a stack of decoder layers, which use feed-forward neural networks and self-attention mechanisms, where:

> Neural networks are a type of machine learning algorithm that consist of layers of interconnected nodes that can learn and make predictions based on input data.
>
> Self-attention is a type of attention mechanism that allows neural networks to focus on different parts of an input sequence without the need of explicit alignment information.

GPT's Utilization of Decoder Layers

Figure 5 shows a representation of a GPT decoder. The rest of this section discusses each of its components.

Component 1: Input

Component 1 of the decoder is the input. If, for example, the prompt is the question

> "What is an actuary?"

GPT begins by rewriting the text as tokens. In this case, assuming GPT-4, the 6 GPT tokens are

and the 6 GPT-4 token IDs are:

> [3923, 374, 459, 1180, 3620, 30],

which would be the input.

---

[20] References for this section include Ferrer (2024), Huang et al (2023), Kalra (2023), and Sengupta (2023).

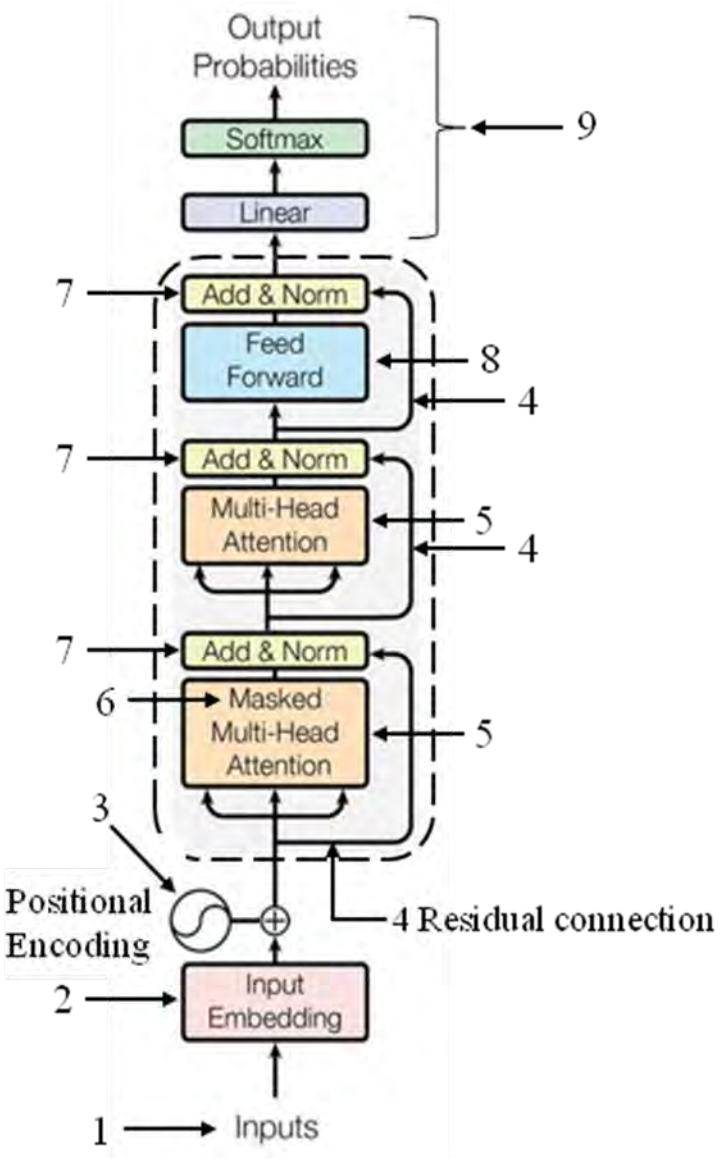


Figure 5: The GPT Decoder[21]

Component 2: Input Embedding

Earlier on, during the pre-testing section, we discussed the embedding matrix. We noted that this embedding space is like an open space where words of similar meanings are grouped together. Here, each input token is mapped to a high-dimensional vector that meaningfully represents its underlying word.

[21] Adapted from Vaswani et al (2017).

Component 3: Positional encoding[22]

Since transformers do not process language sequentially, like traditional models, they do not inherently track the order of tokens. Positional encoding resolves this issue by injecting information about a word's position into the input embedding.

We follow Vaswani et al (2017), and use a combination of sine and cosine functions to create positional vectors. These are then added to the input embeddings so as to provide information about the position of each token in the sequence.

Hence, the formula for positional encoding is as follows:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10,000^{2i/d}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10,000^{2i/d_{model}}}\right)$$

where: (see Figure 6)

pos ≡ the position of the token in the sequence.

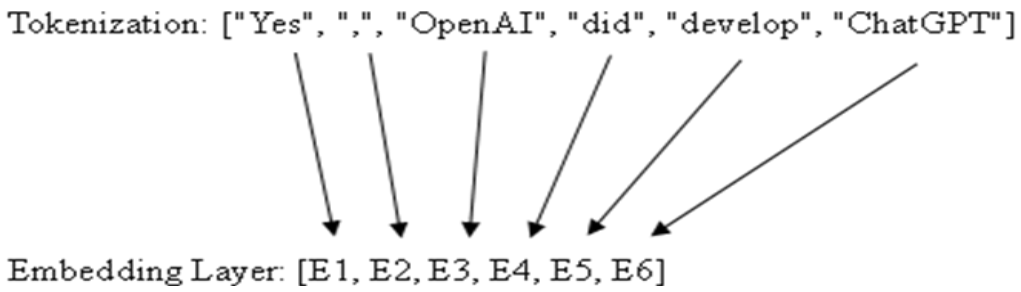
i ≡ dimension index.

d ≡ the dimensionality of the model.

These sine and cosine functions of different frequencies help the model to distinguish between different positions in the sequence.

A Simple visualization of the embedding-positional encoding process

Suppose the input sequence is: "Yes, OpenAI did develop ChatGPT". Then,



[22] References for this section include Haider (2020) and Vaswani et al (2017).

Assuming the Positional Encoding is [P1, P2, P3, P4, P5, P6]

Then, the combined embeddings is [E1+P1, E2+P2, E3+P3, E4+P4, E5+P5, E6+P6]

And the input to model is [E1+P1, E2+P2, E3+P3, E4+P4, E5+P5, E6+P6]

Wolfram (2024: 49) comments "Why does one just add the token-value and token-position embedding vectors together? I don’t think there’s any particular science to this. It’s just that various different things have been tried, and this is one that seems to work."

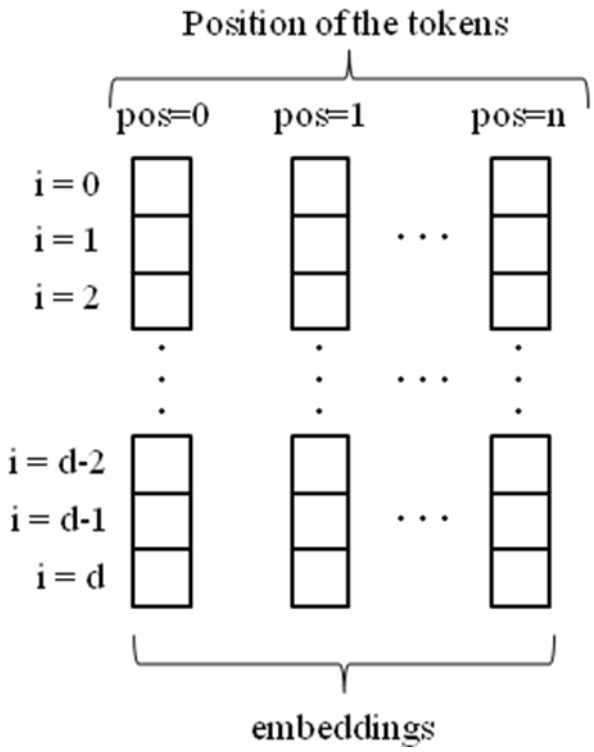


Figure 6: Parameters of Vaswami's positional encoding formula

Component 4: Residual connection[23]

Residual connections provide shortcuts that skip layers or blocks of computations. The primary reason for doing this is to mitigate the problem of vanishing gradients. As a result of the residual connection, the output of a layer is added to its input, which allows the gradient to pass through the layers without losing much information.

Component 5: Multi-Head Attention[24]

The goal of the attention layer is to capture the contextual relationships that exists between different words in the input sentence. This step ends up generating an attention vector for each word. As we are using multiple attention vectors (see Figure 9), this process is called a multi-head attention block.

Attention[25]

A major reason for GPT's success is its transformer architecture, which relies on self-attention mechanisms to process and generate text. This self-attention mechanism enables GPT to capture long-range dependencies and context within input sequences, which allows it to produce more coherent and contextually relevant outputs.

The focus on self-attention not only overcomes the limitations of earlier NLP models like recurrent neural networks (RNNs), but also enables highly segmentable computation, resulting in faster and more efficient training.

The Hidden States of GPT[26]

As mentioned previously, each input token is first converted from a discrete symbol into a continuous, high-dimensional vector through an embedding process. These embeddings contain the basic semantic properties of the tokens. Then, as the embedded tokens pass through the multiple transformer layers, each token's representation gets refined. These representations are commonly referred to as **hidden states**.

By the time these hidden states are ready for the attention mechanism, they are rich in contextual information, capturing both local details (like word order) and Global dependencies (related to earlier tokens in the sequence).

---

[23]References for this subsection include Sengupta (2023) and Tutorialspoint (2024).

[24] References for this section include Ankit & Whitfield (2024), Kalra (2023) and Keita (2022).

[25] References for this subsection include Sengupta (2023).

[26] References for this subsection include Yoshida and Gimpel (2021).

The Query (Q), Key (K) and Value (V) matrices[27]

Query-key-value is a procedure used in self-attention layers of transformer models, including GPT, where each input token is transformed into three vectors:

A query vector, which is used to compute a score for how well each token in the sequence matches the current token, based on their similarity.

A key vector, which is used to capture information about other tokens in the sequence, and

A value vector, which represents the actual content or information the token carries.

Let:

H represents the collection of hidden states[28], and

W_Q, W_K, and W_V be weight matrices developed during pre-training.

Linear Projection for Q, K, and V:

For the self-attention operation, the hidden states undergo three separate linear transformations using distinct learned weight matrices:

Query (Q): Computed as: $Q = H \times W_Q$

Key (K): Computed as: $K = H \times W_K$

Value (V): Computed as: $V = H \times W_V$

Analogies:

Query (Q): the current word

Key (K): other words in the sequence, and

Value (V): stores the encoded information of each word

[27] References for this section include Huang et al (2023).

[28] The collection of hidden states refers to the set of vector representations produced at each layer of the model for each token in the input sequence.

The relationship between Q and K

As far as the relationship between the Q's and K's, consider Figure 7, where the input text is

"The total exposure was underestimated."

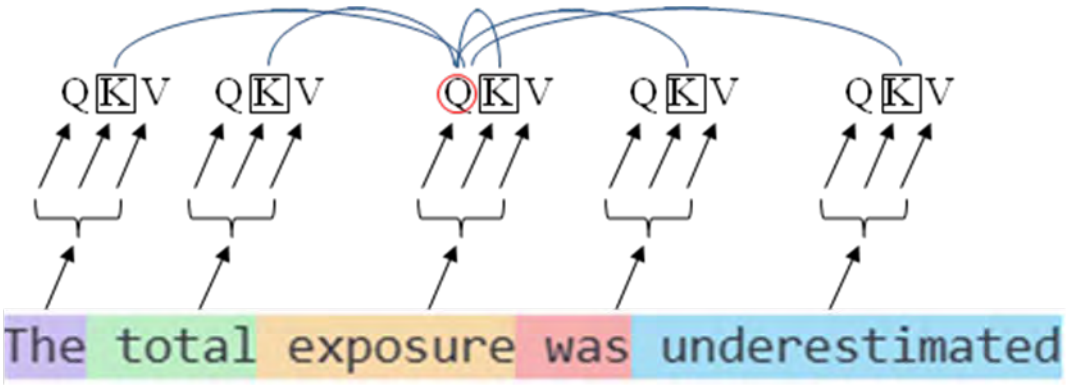


Figure 7: Relationship between Q's and K's[29]

Focusing on the word "exposure", the procedure will be to take its vector Q, and combine it, via a dot product, with each of the K vectors, for all the words, including for the vector K of the word "exposure".

Although these are separate operations, they will occur in parallel.

The extent of their relationship is indicated by their dot product.[30]

The Attention Mechanism[31]

The attention mechanism is depicted in Figure 8 and is implemented as follows:.

Step 1: Input Tokenization

The input sentence is split into tokens.

Each token is mapped to a vector using the embedding matrix.

---

[29] Adapted from Superdatascience (2024).

[30] A dot product in the vicinity of 1 indicates a high relevance between Q and K, while a dot product in the vicinity of 0, suggests little or no relevance between Q and K .

[31] References for this subsection include Sengupta (2023).

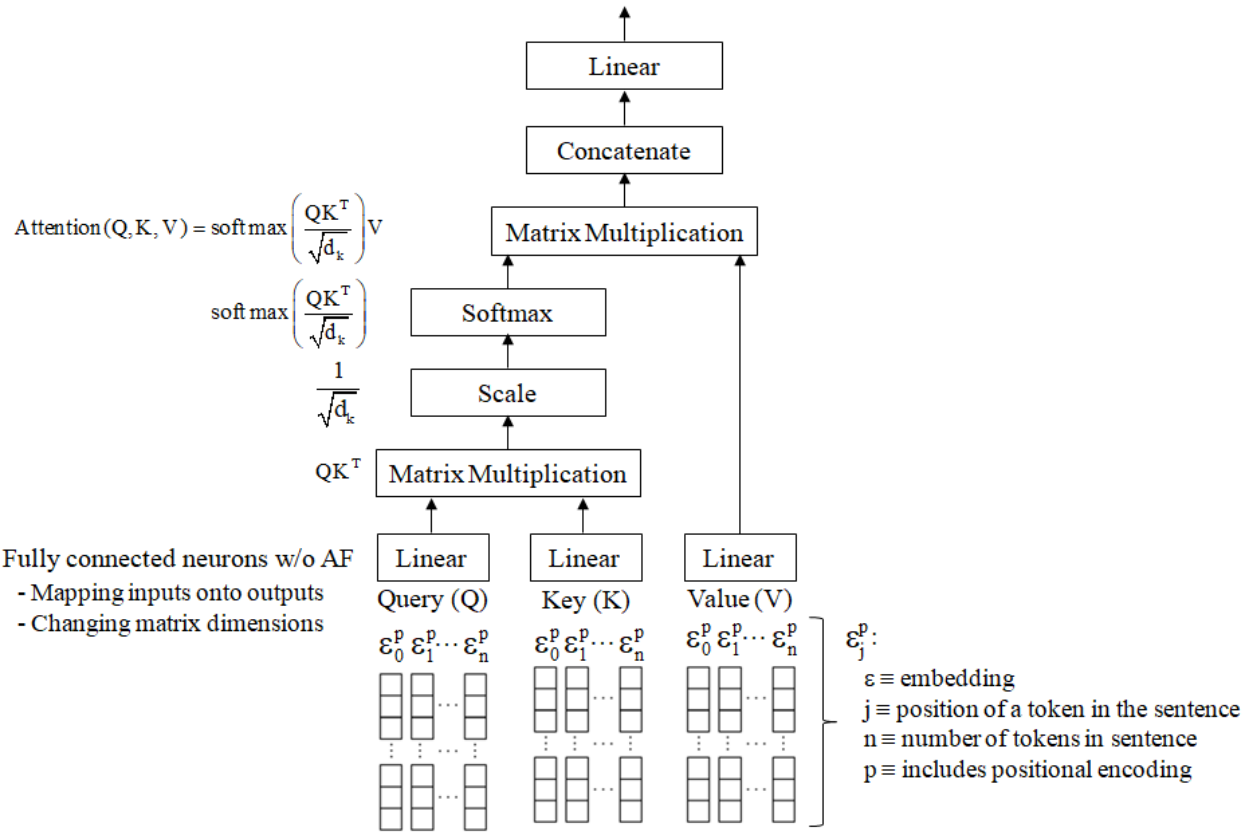


Figure 8: The Attention Mechanism[32]

Step 2: Create Query, Key, and Value Vectors

For each token, the model creates three vectors Query (Q), Key (K), Value (V)

Step 3: Compute Attention Scores

The model compares each Query to all Keys using dot products.

This results in a score matrix showing how much each token is related to every other token.

Step 4: Normalize using matrix multiplication

Step 5: Apply Softmax

The scores are normalized using softmax, turning them into probabilities.

These probabilities determine how much weight each token gets in the final output.

---

[32] Adapted from Haider (2020) and Dahami (2024), and Sengupta (2023).

Step 6: Weighted Sum of Values

Each token's output is a weighted sum of all Value vectors, using the attention scores as weights.

This allows the model to focus more on relevant tokens.

Step 7: Output to Next Layer

The attention output is passed to the next layer (or used to generate the next token in GPT).

This process is repeated across multiple layers in the transformer.

Multi-head Attention Representation

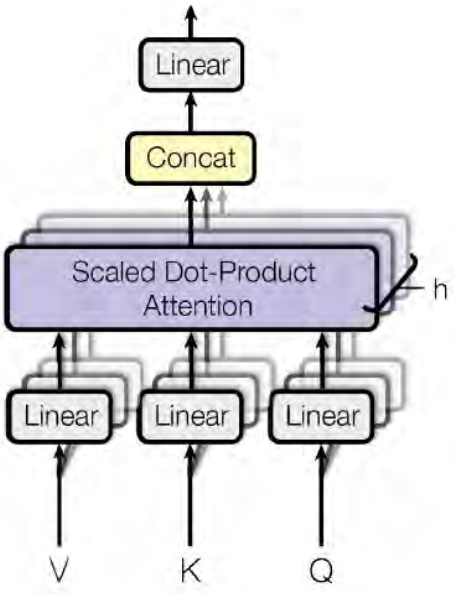


Figure 9: Multi-head Attention Representation[33]

As reported by Vaswani et al (2023: 4-5), "Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to dk, dk and dv dimensions, respectively. … These are concatenated and once again projected, resulting in the final values, as depicted in [Figure 9]. ... Multi-head attention allows the model to jointly attend to (focus on) information from different representation subspaces at different positions."

[33] Adopted from Vaswani et al (2023: 4-5).

Component 6: Masked Multi-head attention[34]

In GPT, when generating text, the decoder works autoregressively, in that it predicts one token at a time, based on previous tokens. Masking ensures that the model only considers previous words when making each prediction, preventing it from seeing future tokens in the sequence before they are generated.

Component 7: Add & Norm[35]

An "add & norm" denotes a residual connection immediately followed by a layer normalization. Its purpose is to boost the performance of the decoder by avoiding vanishing gradients and stabilizing the training process. The "Add" portion of the operation refers to the addition of a residual connection, which allows information to flow more easily through the layers of the model during training. The "Norm" part refers to layer normalization, which standardizes the values of the input to a mean of 0 and a standard deviation of 1. This helps to stabilize the learning process and improve the model's ability to generalize to new data.

Component 8: Feed forward NN[36]

A simple feed-forward neural network is applied to every attention vector to transform it into a form that is acceptable to the next decoder layer.

The feed-forward network incorporates the information from the previous steps to find potential output words.

Given the word and the context it is in, the network predicts the next most probable word.

Component 9: Output probabilities[37]

The final component of the decoder's process involves:

A linear layer, serving as a token classifier,

Topped off with a softmax function to calculate the probabilities of different words.

---

[34] References for this subsection include Kalra (2023).

[35] References for this subsection include Tutorialspoint (2024).

[36] References for this subsection include Ankit &Whitfield (2024) and Kalra (2023).

[37] References for this subsection include Kalra (2023) and Ferrer (2024).

Commentary

The purpose of this article was to present some preliminary observations from a study that investigated and documented the technical concepts and training approaches that power GPT models.

The topics covered included:

GPT model development,
How GPT processes a prompt,
Core pre-training concepts, including tokenization and embedding,
Core GPT concepts, including self-attention and positional encoding,
Transformers, and
the GPT decoder.

This being a preliminary version notwithstanding, it is hoped that this article helps to explain how and why GPT models work. To the extent it does so, it will have served its purpose.

References

Ankit, A. Whitfield, B. (2024). Transformer Neural Networks: A Step-by-Step Breakdown. https://builtin.com/artificial-intelligence/transformer-neural-network

Balona, C. (2024). ActuaryGPT: applications of large language models to insurance and actuarial work, British Actuarial Journal, 29, e15, 1–42.

Barkovska, O. (2022). Performance study of the text analysis module in the proposed model of automatic speaker's speech annotation. Computer systems and information technologies, 4, 13-19. DOI: 10.31891/csit-2022-4-2.

Barkovska, O., Havrashenko, A., Serdechnyi, V., Kholiev, V., Rusnak, P. (2024). Analysis of the impact of the contextual embeddings usage on the text classification accuracy, Intelligent information technologies, **3(111),** 67-78. DOI: 10.32620/reks.2024.3.05

Barkovska, O., Kholiev, V., Havrashenko, A., Mohylevskyi, D. & Kovalenko, A. (2023). A Conceptual Text Classification Model Based on Two-Factor Selection of Significant Words, 7th International Conference on Computational Linguistics and Intelligent Systems, COLINS, 244-255.

Barkovska, O., Khomych, V., & Nastenko, O. (2022). Research of the text processing methods in organization of electronic storages of information objects. Innovative technologies and scientific solutions for industries,1(19), 5-12. DOI: 10.30837/ITSSI.2022.19.005.

Brodigan, D., O'Dwyer, D. (2023). Glossary of AI terms: Understanding GPT, neural networks, and more. https://www.intercom.com/blog/ai-glossary/

Brown, O., Latendresse, H., Tan, S. (2023). GPT-3 and the actuarial landscape: An Overview of Large Language Models and Applications, CAS RPM Seminar, 1-46.

Carlin, S., Mathys, S. (2024). A Primer on Generative AI for Actuaries, Society of Actuaries Research Institute, 1-46.

Dascalescu, D., Hasan, Z. (2023). Vector Embeddings Explained https://weaviate.io/blog/vector-embeddings-explained

Dahami, Y. (2024). Understanding How ChatGPT Uses the Decoder-Only Transformer Architecture. https://medium.com/@dahami/understanding-how-chatgpt-uses-the-decoder-only-transformer-architecture-c247c872754a

du Preez, V., Bennet, S., Byrne, M., Couloumy, A., Das, A., Dessain, J., Galbraith, R., King, P., Mutanga, V., Schiller, F., Zaaiman, S., Moehrke, P., van Heerden, L. (2024). From bias to black boxes: understanding and managing the risks of AI – an actuarial perspective, British Actuarial Journal 29, e6, 1–52.

Ferrer, J. (2024). How Transformers Work: A Detailed Exploration of Transformer Architecture https://www.datacamp.com/tutorial/how-transformers-work

Gautam, H. (2020). Word Embedding: Basics https://medium.com/@hari4om/word-embedding-d816f643140

Giray, L. (2023). Prompt Engineering with ChatGPT: A Guide for Academic Writers. Annals of Biomedical Engineering, 51, 2629–2633. https://doi.org/10.1007/s10439-023-03272-4

Google. (2025). Gemini [Large language model]. https://gemini.google.com/

Haider, B. (2020). Visual Guide to Transformer NNs - 1 Position Embeddings. https://www.youtube.com/watch?v=dichIcUZfOw

Hinton, J. (2023) Godfather of AI - 60 Minutes Interview. https://www.youtube.com/watch?v=qrvK_KuIeJk

Howarth, J. (2025) When Will ChatGPT-5 Be Released (July 2025Update)

Huang, K., Wang, Y., Zhu, F., Chen, X., Xing, C. (2023). Beyond AI, Springer Nature, Switzerland.

Jones, H., McLeod, A. (04-03-2023). The Future Is Now: How GPT-4 Will Revolutionize the Actuarial Profession, Canadian Institute of Actuaries.

Kalra, J. (2023). Explaining GPT-4's Secret Sauce: Transformers. https://janvikalra.substack.com/p/explaining-gpt-4s-secret-sauce-transformers

Kansal, A. (2024) Building Generative AI-Powered Apps, Springer Science + Business Media New York

Merritt, R. (2022). What Is a Transformer Model?

https://blogs.nvidia.com/blog/what-is-a-transformer-model/

Microsoft. (2025). Copilot (GPT-4) [Large language model]. https://Copilot.microsoft.com/

OpenAI. (2025, March 27). ChatGPT (Mar 27 version) [Large language model]. https://chat.openai.com/

Penn State University (2024) Generative AI: Chat GPT and Beyond. https://guides.libraries.psu.edu/c.php?g=1338692&p=9866318

Po, L. M. (2025). A Brief History of LLMs. https://medium.com/@lmpo/a-brief-history-of-lmms-from-transformers-2017-to-deepseek-r1-2025-dae75dd3f59a

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training, OpenAI.

Sanderson, G. (2024). Transformers, the tech behind LLMs | Deep Learning Chapter 5. https://www.youtube.com/watch?v=wjZofJX0v4M&t=437s

Sariagac (2025). Exploring the feasibility of retrieval-augmented generation with integrating private data into AI-driven BPM tool in company Promatis software. Master Thesis, School of Economics and Business, University of Ljubljana.

Sengupta, S. (2023). A Deep Dive into GPT's Transformer Architecture: Understanding Self-Attention Mechanisms, https://www.gptfrontier.com/a-deep-dive-into-gpts-transformer-architecture-understanding-self-attention-mechanisms/

Soudani, H., Kanoulas, E., Hasibi, F. (2024). Fine Tuning vs. Retrieval Augmented Generation for Less Popular Knowledge. In Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region, 12-22. https://doi.org/ 10.1145/3673791.3698415

Superdatascience (2024). Why Self-Attention Powers AI Models: Understanding Self-Attention in Transformers. https://www.youtube.com/watch?v=1qgeA0Fl_HY

Tutorialspoint (2024). Normalization and Residual Connections. https://market.tutorialspoint.com/

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30. https://arxiv.org/abs/1706.03762

Wolfram, S. (2023). What Is ChatGPT Doing … and Why Does It Work? Wolfram Media, Inc.

Wolfram, S. (2024). Chat GPT - What Is It and Why Does It Work. https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/

Yoshida, D., Gimpel, K. (2021). Reconsidering the Past: Optimizing Hidden States in Language Models, 4099-4105