Large Language Model Architectures and Training Techniques: A Literature Review

KATHLEEN KELLY, Colorado School of Mines, USA

Large language models (LLMs) represent the forefront of artificial intelligence in natural language processing. To effectively contribute to this area of research, it is important to understand the underlying fundamentals and architectures of LLMs. This literature review aims to present a history of LLMs, some fundamental concepts of the LLM architecture, and then compare and contrast several well-known models as well as more recent advances in the field. Reviewing many architectures together, pros and cons of the varying design decisions will be shown with the goal of increasing understanding of the field to help further research.

Additional Key Words and Phrases: large language model,neural network,machine learning,GPT,BERT,BART,Claude,Llama,RNN,encoder,decoder,feed-forward, transfer learning,attention,pre-training,fine-tuning,layer,Gemini

ACM Reference Format:

Kathleen Kelly. 2025. Large Language Model Architectures and Training Techniques: A Literature Review. 1, 1 (June 2025), 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Large language models represent the forefront of artificial intelligence in natural language processing. To effectively contribute to this area of research, it is important to understand the underlying fundamentals and architectures of LLMs. Models have nuances in their designs and training methodologies, challenges in their development life cycles and deployments, and strengths and weaknesses in their model decisions.

Natural language processing and neural networks became especially popular in the 1990s, leading to deep learning and then the novel Transformer architecture from which most LLMs are based today. Understanding the layers of this architecture is a crucial first step in understanding the most recent developments in LLMs, including how they are trained, fine-tuned, deployed, and used.

This literature review begins with some previous work and fundamentals of LLMs, describes the most common steps in the development life cycle, then focuses on reviewing the architectures of 14 large language models, comparing their design decisions. The rest of the paper is organized as follows: 2) History and Fundamentals, 3) Development Life Cycle, 4) Architectures, 5) Deployment, 6) Use Cases and Challenges, and 7) Discussion and Future Work.

Author's address: Kathleen Kelly, kathleenmariekelly@mines.edu, Colorado School of Mines, Golden, CO, USA.

XXXX-XXXX/2025/6-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 HISTORY AND FUNDAMENTALS

2.1 History of LLMs

Large language models (LLMs) are based on artificial neural networks and have grown in use and improvement for a hundred years. LLMs can be traced back to the 1800s when languages and semantics were studied by Michel Breal, a French philologist [26]. Languages as systems were studied in the early 1900s at the University of Geneva, leading to the publication of Language as a Science in 1916. In 1959, Arthur Samuel at IBM created a computer game of checkers that was described as "machine learning". The first artificial neural network was called the Mark 1 Perceptron in 1958 [26] and laid the groundwork for standardizing software across different computers.

The first natural language programming (NLP) is credited by Joseph Weizenbaum in 1966 at MIT, a program called ELIZA. It was able to take input and respond with a programmed answer based on pre-defined rules [2]. Due to small amounts of data storage and very slow processing speeds, development of language models did not grow in the 1970s, but by the 1980s IBM developed a first small language model that could predict the next word in a sentence using statistics. This statistical work along with increased computational power, improved machine algorithms, and the large amounts of data available on the internet led to a boom in NLP in the 1990s.

The Recurrent Neural Network (RNN) was introduced in 1986 [70], a novel approach to neural networks that could handle input and output of varying sizes because of a recursive processing unit and hidden state that remembers the past. The basic RNN could easily become overloaded due to trying to remember too much information. A variant of RNN, Long Short-Term Memory (LSTM) was introduced in 1997, leading to deep learning where machine learning was combined with neural networks and many more layers. LSTM was able to remember more information with its input, output, and forget gates along with a sigmoid function. Deep learning became widespread in the 2010s, used in everyday households with Apple's Siri. In 2010, Stanford introduced The CoreNLP suite, and in 2011, Google introduced Google Brain, both of which helped researchers solve much more complex machine learning problems.

In 2017 the Transformer architecture was introduced [48], the most common architecture used in generative AI today. Google researchers released BERT in 2019, a bidirectional encoder representation from the Transformer architecture [18]. Then in 2022, OpenAI released ChatGPT that brought generative AI into a new dimension, allowing for entire conversations as well as generation of resumes, speeches,

Table 1. Literature Review Papers

Author	Year	Title
Cho et al. [19]	2014	Learning Phrase Representations using RNN Encoder-Decoder...
Vaswani et al. [67]	2017	Attention Is All You Need
Devlin et al. [23]	2019	BERT: Pre-training of Deep Bidirectional Transformers...
Liu et al. [44]	2019	RoBERTa: A Robustly Optimized BERT Pretraining Approach
Lewis et al. [42]	2019	BART: Denoising Sequence-to-Sequence Pre-training...
Touvron et al. [66]	2023	LLaMA: Open and Efficient Foundation Language Models
Hal Koss [39]	2023	What is Claude AI...
Fayyazi et al. [57]	2024	Advancing TTP Analysis...
Zhang et al. [73]	2024	TinyLlama: An Open-Source Small Language Model
Reid et al. [56]	2024	Gemini 1.5: Unlocking multimodal understanding...
Pan et al. [53]	2024	Unifying Large Language Models and Knowledge Graphs: A Roadmap
Liu et al. [43]	2024	From LLM to Conversational Agent...
Zhan et al. [72]	2024	AnyGPT: Unified Multimodal LLM...
Cong et al. [21]	2024	AttentionLego: An Open-Source Building Block...
Park et al. [55]	2024	Any-Precision LLM: Low-Cost Deployment...
Xuchen Suo [65]	2024	Signed-Prompt: A New Approach...

images, and more. See Figure 1 for a brief outline of the history of LLMs [2].

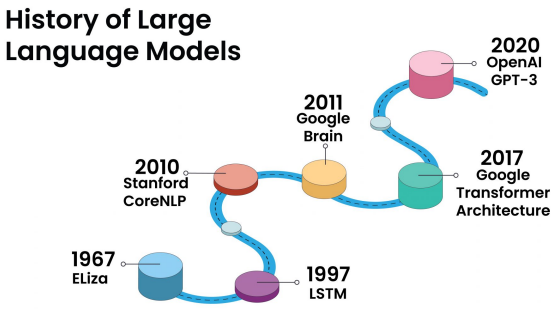

Fig. 1. History of Large Language Models from [2]

2.2 Fundamentals of LLMs

At the most basic level, a machine learning model is a computer program that can recognize patterns in input data, then make predictions from those patterns to generate output data [11]. These models use machine learning algorithms to perform the prediction tasks. The most common machine learning algorithms are supervised learning, unsupervised learning, and reinforcement learning. With supervised learning, the input data is labeled so the algorithm can learn from the labels, where in unsupervised learning, there are no labels so the algorithm must learn from patterns in the data. Some algorithms use semi-supervised learning, a combination of both supervised and unsupervised learning. Reinforcement learning assigns positive and negative values to the algorithm results, trying to encourage certain results while discouraging others, a trial-and-error learning process. In all these cases, the algorithms are trained with input data by the machine learning engineers.

Training an algorithm involves first setting hyperparameters, for example the number of clusters to use in a pattern-matching clustering algorithm, or the number of branches to use in a decision tree algorithm. These parameters guide the algorithm and lead to output parameters, such as weights and biases. LLMs today are training with billions or even trillions of parameters allowing the models to learn natural language processing as well as context, semantics, and nuances [20]. Training on a few hyperparameters is relatively straightforward, while training LLMs with billions of parameters presents challenges in computational resources and cost [68].

Two main types of machine learning models are those used for regression and those used for classification or prediction. The underlying algorithms used are regression algorithms and classification algorithms [61]. In a regression algorithm, a prediction is made based on past input and output using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). In this case, the target variable is continuous. Using these metrics, there are different types of regression models that are often used, including linear regression, ridge regression, and lasso regression.

Classification algorithms are used to predict a "discrete outcome" based on classes or categories [61]. Classifications can be binary or multi-class depending on the possible outcomes. Classification algorithms are measured for their accuracy, precision, and recall. Common classification models include logistic regression and K-nearest neighbors. Tree-based models are also commonly used for both classification and regression problems. The trees can be decision trees, breaking a problem down into branches of decisions, and random forests that are decision trees with less overfitting. The models described are supervised machine learning models, while there are also popular unsupervised clustering algorithms. K-Means Clustering is a common model that tries to group similar unlabeled objects together into K clusters.

The training of a LLM is the process of teaching the model to make decisions or predictions. To train a model, there must be data, algorithms, adjustable parameters for fitting the data, a loss function to measure the difference between the model predictions and actuality, and optimizations [10]. By training a model, it is able to better generalize its findings on new data, it is better able to recognize patterns in data, and it will be better able to be automated. To properly train a

model, there must be well-prepared data, the appropriate algorithm for classification, regression, or clustering, evaluation, parameter tuning, and optimization, all in a very delicate balance.

There are challenges associated with LLMs, both with the models and the data sets. To train a LLM uses significant computational power and very advanced hardware that is hard to obtain and expensive, as well as adding an "environmental toll" with the massive resources needed [59]. During training, it is common that a model can suffer from overfitting or underfitting [49]. Overfitting is when a model is too complex and is able to memorize the training data instead of actually learning data patterns. This model will appear to be very precise with the training data, but will perform poorly with new data. Similarly, underfitting is when a model is too simple to understand data patterns and performs poorly. There are ethical considerations around the training data, whereby if the model is trained on biased data, it will output non-diverse responses [59]. Similarly, there are data privacy concerns with the data the model is trained on [59]. It is crucial that the creators of LLMs are aware of the challenges and address them with transparency and responsibility.

3 DEVELOPMENT LIFE CYCLE

LLMs are built from several components that all work together to understand the human language [64]. These key components can be seen in Figure 2.

Fig. 2. LLM Building Blocks from [64]

3.1 Data collection and preprocessing

Training large language models involves data collection and preprocessing, a series of steps for cleaning and tokenizing the data. There is also model configuration of the hyperparameters.

Data is collected from massive datasets, web pages, books, and more to provide the model with a large range of concepts and languages, also known as a corpus [47]. The corpus is then cleaned to remove invalid or incorrect characters and duplicates, as well as removing personally identifiable information. The data preparation step is crucial so the model is trained on the most accurate and complete data, leading to better output results [9]. However, as data is cleaned, there may be missing values or outliers in the values that need to be corrected or removed.

Once the data has been cleaned, it is transformed into a format the algorithm can understand, usually some type of data normalization. Often times this format is feature scaling or feature encoding. In these processes, values that are on different scales or in varying categories need to be normalized into one universal scale for the algorithm to be able to comprehend.

Data tokenization involves splitting text into small individual tokens. The value of a token is mapped to some part of the plaintext data, and all the token mappings are stored in a database. There are many algorithms for tokenizing data, and the challenge is finding the best location for the tokenization split to provide meaning and context to each token. Word level tokenization splits a sentence into each word, as opposed to character level tokenization that splits each sentence into every character in every word. Another common algorithm is Byte Pair Encoding (BPE) which merges commonly occurring characters by splitting the word into sub-words, such as the beginning and end of the word [33]. Tokenization methods choose varying token sizes, where smaller tokens use less memory and are more flexible but are not able to understand context. Larger tokens increase efficiency and allow for understanding of context, but are more memory-intensive [32].

The preprocessing pipeline can be seen in Figure 3.

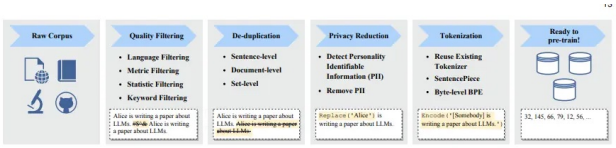

Fig. 3. Data Preprocessing Pipeline from [47]

3.2 Data embedding

Data embedding is the process of converting words into vectors of numbers. The length of the vector of numbers is variable and can be tuned as a hyperparameter of the model. The columns of the vector can represent features of the words, such as size, color, etc. One issue with embeddings is that multiple meanings of a word will be represented as the same embedding, but the attention layer is helpful for these cases [38].

Input embeddings convert words into vectors of numbers to be used by the model. The simplest form of embedding is the one-hot encoding method, where every word in the input data is represented as a vector of all zeros and a single one at the position of that word in the vocabulary. Depending on the vocabulary size N, every word would be represented as a vector of N-1 zeros and a single one. This method is simple but does not retain context of words, resulting in

words with multiple meanings to have the same vector embedding [38].

Term frequency-inverse document frequency (TF-IDF) is an embedding strategy where words are assigned a weight based on a word's frequency in a document and its inverse document frequency. A TF-IDF score is high when a word appears frequently in one document but rarely in an entire corpus of documents, showing its importance in one document yet rarity overall. TF-IDF helps capture context, but is ineffective in capturing the semantics of words.

For semantic capturing, one popular technique is Word2Vec [38]. This technique uses a neural network to be able to predict the surrounding words of an input word, learning word associations. Another word embedding technique is GloVe, which uses a "co-occurrence matrix", keeping track of how often two words appear together, thus learning word associations. Both these techniques, however, are unable to encode words that are not in the vocabulary, and both techniques struggle with sentence context [37]. A sentence embedding technique is embedding from language models (ELMo) [38]. ELMo is able to learn the meaning of words and their context, assigning the same word different embeddings depending on the sentence in which it appears. ElMo uses a bi-directional LSTM network to produce its vector representations, achieving "state of the art results" but also being computationally expensive [7].

In one novel approach named Landmark Embedding, Kun et al. [45] used a "chunking-free embedding method" where long sentences could be kept intact instead of tokenized. Landmark tokens were used at the end of each sentence to keep together semantics of that sentence, while also holding together the related sentences. The landmark tokens also helped with retrieval of information, being able to label the exact boundaries of information.

Regardless of the embedding algorithm chosen, the tokenized input data into a LLM is converted into numerical vectors that are then stored in a vector database for quick responses. These vectors encode the tokens and their relationships so the model can understand context. See Figure 4.

3.3 Data normalization

In the data normalization step, the data is transformed so that its mean is close to zero and its standard deviation is close to one, also known as standardization of the data. Common normalization techniques include batch normalization and layer normalization [6]. Batch normalization uses batch statistics to process data in batches, but for small batch sizes, the batch mean and standard deviation values will not be meaningful [17]. Batch normalization works well for mini-batch training and can help with the problem of overfitting, but can also cause slowness in training [29]. Layer normalization calculates normalization for each layer per data point instead of per batch. This normalization technique can be used with any batch size and is simpler to implement, but may not outperform batch normalization for large batch sizes [29]. In batch normalization, there is a dependency on batch size, but not so for layer normalization. Batch normalization is able to normalize each feature across each small batch, while layer normalization works on the inputs in the batch "independently across all features" [17].

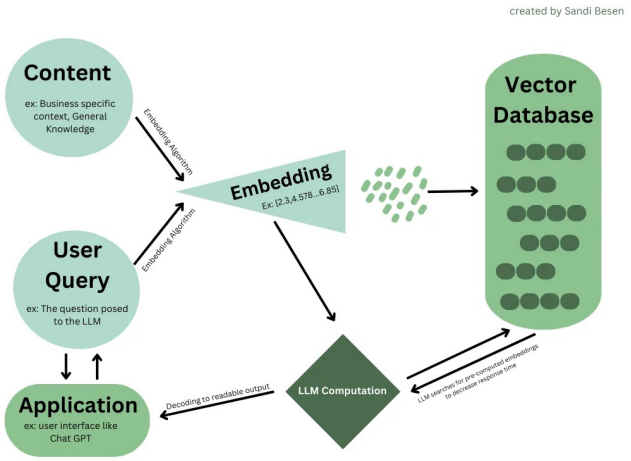

Fig. 4. Flow of Data in LLM from [15]

3.4 Attention

The attention layer of a LLM is used to discover relationships between tokens and find "long-range dependencies" [64], deciding what information in the input is the most important and help with ambiguity of words. This layer is usually highly parallelized for efficiency. This is a crucial layer of the LLM as it looks at the tokens in relation to each other to discover context, dependencies, and relationships. There are different types of attention strategies, such as self-attention, cross attention, sparse attention, and flash attention [50]. Specifically, the self-attention mechanism allows the model to focus on different parts of the input at the same time instead of step-by-step [28]. By using weighted sums of the values it receives from the previous layer, this layer is able to pay attention to certain parts of the input using query, key, and value representations. Key, query, and value matrices are matrices of values that are populated during training. As embedding vectors come into the attention layer, the values within the vectors are updated as new weights are calculated based on context, then the vectors can be sent back through the model.

Similarity between words or tokens are measured using similarity scores. A dot product calculation is a way to measure similarity, multiplying embedding vectors of feature data. Tokens sharing similar features will yield higher values in corresponding vector indices, resulting in a higher dot product score and causing them to cluster together in the model. Cosine similarity is another way to measure similarity, measuring the angle between two embeddings with a result of a number between 1 and -1. Additionally, the scaled dot product is a widely used technique for measuring similarity that does the dot product divided by the length of the embedding vector, creating manageable measurements for very large vectors.

As mentioned, the matrices in a similarity computation are key and query matrices which are used together to make a linear transformation matrix. This linear transformation is used in the dot product of the input tokens in question, looking for the highest similarity score, updating the embedding to one that is "better" [4]. A value matrix is used to find the next best word in the sentence. The values matrix is multiplied by the resulting similarity scores to then choose the next word based on context. The similarity embedding knows the features of words while the word embedding knows the sentence context. A Softmax function is used in the attention layer to convert the weights to normalized positive values, passing the values on to the next step in the layer. See Figure 5 for the steps in the attention layer.

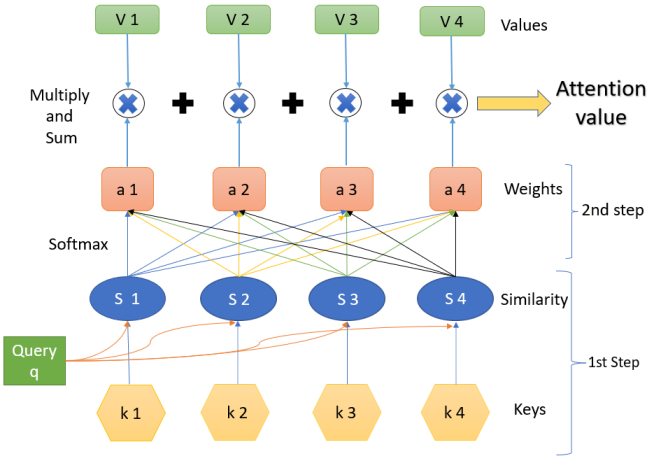

Fig. 5. Attention Layer from [58]

In multi-head attention layers, the attention module is repeated many times in parallel to give the Transformer the ability to encode relationships and nuances for each word. In this technique, an embedding matrix is transformed using linear transformations to create modified versions of the matrix, then each matrix is given a score. The higher scoring embedding will be the embedding that is chosen by the model. Multi-head attention uses many key and query matrices to form multiple attention embeddings, then multiple value matrices to transform similarity embeddings to context embeddings. These results are concatenated into a high-dimensional embedding, then transformed into a lower-dimensional embedding [4]. By splitting the input embeddings across multiple attention heads, the model is able to learn different aspects of the meanings of words.

3.5 Feed-Forward

After the attention layer, the attention vectors go through a feed-forward layer one at a time so they can be parallelized [54]. Feed-forward layers are unidirectional, only looking forward, not back in time. The model is able to capture features and patterns in this layer.

3.6 Pre-training

Pre-training is a step that involves sending the prepared data as input into the model repeatedly, teaching the model to choose more wisely as its loss function improves. This step is unsupervised, whereby the model is given unlabelled data. A loss function, or error function, is an algorithm that quantifies the error between what the model outputs and what is expected. This loss should be as small as possible for the model to output the most correctly. One common loss function is the mean square error function (MSE), where for every data point input into the model and every output value, a distance is calculated between them. The error is this distance squared. The average of all the distance squared values is the MSE, and linear regression is the process of trying to minimize this MSE [5]. A loss function attempts to maximize a likelihood, namely that the model will choose the correct next token of all the possibilities. Input text goes through the layers of the model, then output text is the probability distribution of all the possible next tokens at that location in the sequence. The logarithm of that probability is calculated and used in a sum for the loss function [24]. See Figure 6.

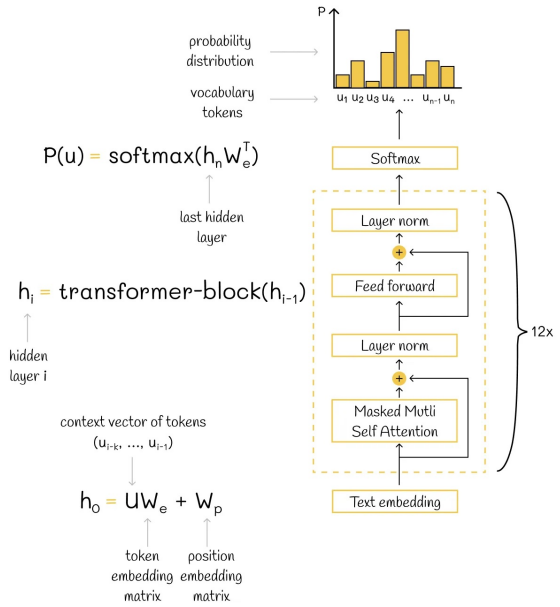

Fig. 6. Pre-training in GPT from [24]

3.7 Fine-tuning

Fine-tuning is a step that occurs further downstream in the model on labelled supervised data. This layer also uses a loss function, but the loss being computed is between the expected label and the actual label from the model. This step trains the model on a specific task so it can adapt what it learned in pre-training to that specific task [36]. The model is able to retain what it already learned, but by adjusting parameters, can specialize in the target domain. Fine-tuning is accomplished in various ways, such as supervised, few-shot learning, transfer learning, and domain-specific fine-tuning. In each

case, the model is further trained on data, examples, or domain-specific target data. The quality of the fine-tuning data is critical, as well as the tuning of hyperparameters [25].

3.8 Transfer learning

Because of the fine-tuning step, LLMs are able to perform transfer learning, being able to transfer knowledge from one source to a different target. Performing training on models is time consuming and expensive, but with fine-tuning and transfer learning, models can reuse training data across many different contexts [13]. Sentiment Analysis is a common application of transfer learning, the process of a model learning the sentiment or emotion of the text. Named Entity Recognition (NER) is another common application of transfer learning, where the model is able to categorize named entities due to the model parameters being adapted to the specific categorization task [13]. Other transfer learning applications include text classification, question answering systems, and language translation.

As with other layers of the LLM, there are concerns of bias in the pre-training data which would then propagate through the fine-tuning and into the transfer learning layers. Similarly, privacy and security risks in the data would also propagate through the model. Challenges in transfer learning include differences in the domains and the data between them, leading to additional steps to adapt the data between domains. There also needs to be plenty of data to train against as well as large computational resources.

4 ARCHITECTURES

The architecture of LLMs involves multiple neural network layers, including the embedding layers, the feed-forward layers, and the recurrent layers [69] that all work together. The embedding layer generates the input data embeddings so the model can hold semantic and syntactic meanings. The feed-forward layer does transformation on the embeddings to help the model understand "user intent in inputs". The recurrent layer works on interpretation of the input data, creating data associations.

4.1 History

LLMs started as predetermined rules and heuristics. For example, Eliza, developed by Joseph Weizenbaum at MIT, would take the user's statement and rephrase it into a question, making it seem conversational. Next came more statistical models that tried to predict the most likely output words given input. Neural networks followed, using earlier models and artificial neurons that would activate based on the output of other nodes [46].

A Recurrent Neural Network (RNN) is a type of neural network that passes output from the previous step as input into the current step, as opposed to the inputs and outputs being independent of each other. The RNN contains a Hidden Layer that keeps a "hidden" or "memory" state, remembering information about previous inputs. In an RNN, the weight across the network stays the same, different from deep neural networks where the weights change. At each unit of the RNN, there is a current state that is calculated based on the input state and the previous state. To train a RNN, the current state is calculated which then becomes the input to the next step, going as many steps as is necessary for the problem, leading to an output. This output is compared to the actual target output and an error is generated, then back-propagated to update the weights in the network and train again [1].

Variations of the RNN include the Bidirectional Neural Network (BiNN) and the Long Short-Term Memory (LSTM). LSTMs improve on the RNN by being better able to handle long-term dependencies. RNNs and LSTMs are still in use but tend to be used in conjunction with the Transformer architecture. A Gated Recurrent Unit (GRU) is a variation on the LSTM that tries to solve the "vanishing gradient problem" [40] and was introduced by Kyunghyun Cho et al. in 2014 [19]. GRU uses update and reset gates that manage what information is passed to the output. The update gate is used to determine how much of past information should be passed forward in the model, while the reset gate determines how much of past information should be dropped.

In 2016, the Transformer architecture was founded, moving from sequential to self-attention mechanisms. This architecture allowed for the model to analyze all the words in a sequence at the same time instead of one at a time. This is still the primary architecture used today [22]. The Transformer has three key components: an encoder, a decoder, and a self-attention mechanism. The novel self-attention mechanism is what allows the model to keep long-range dependencies. Most of the LLMs in use today are based on the Transformer architecture but with small modifications to the encoder or decoder.

In a revolutionary paper, Vaswani et al. [67] proposed the Transformer architecture that is based on the attention mechanism and does away with the recurrence model. As seen in Figure 7, there is an encoder with some number of identical layers, each made up of a multi-head self-attention mechanism and a feed-forward network. There is also a decoder with some number of identical layers, each made up of a masked multi-head attention layer, a multi-head attention layer, and a feed-forward network. Attention layers are used in three different ways in the Transformer: 1) self-attention layers in the encoder that use keys, values, and queries from the previous encoder layer, 2) self-attention layers in the decoder that work similarly, and 3) "encoder-decoder attention" layers that use keys and values from the encoder and queries from the decoder. According to Vaswani et al., the Transformer can be trained much faster than RNNs.

4.2 Transformers

4.2.1 GPT. From the Transformer architecture was born all the LLMs of the present day. Generative Pre-trained Transformer (GPT) is a LLM that is based on the Transformer architecture, founded by OpenAI in 2018. It has had iterations of GPT-1, GPT-2, GPT-3, and GPT-4, each an improvement of its predecessor. GPT-3 improved GPT-2 with its number of training parameters, going from about 1.5 billion parameters to 175 billion, as well as being able to generate computer programs. GPT-4 is also able to parse image inputs [60]. GPT differs from other Transformer models in that it undergoes both pre-training and fine-tuning before a final task-specific fine-tuning step. It also uses a multi-head self-attention mechanism so

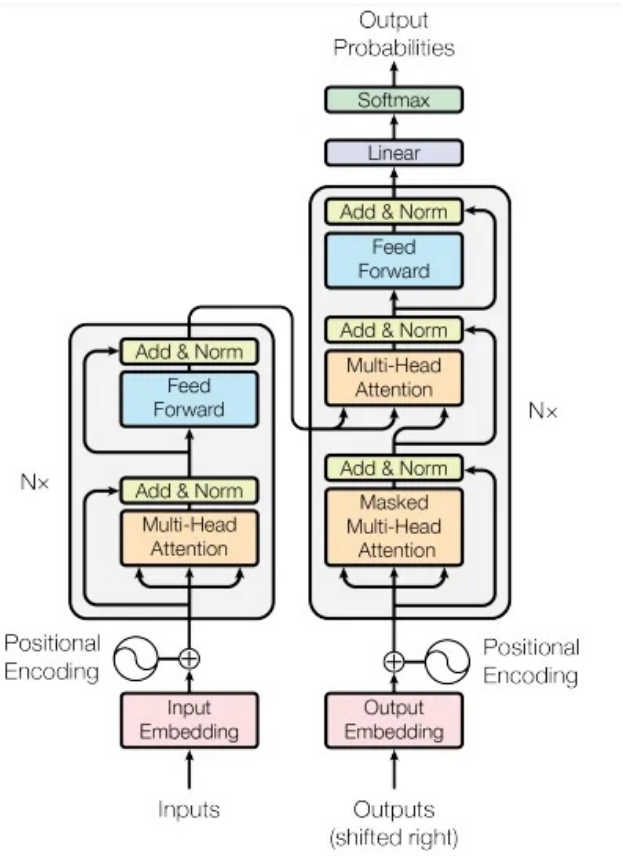

Figure 1: The Transformer - model architecture.

Fig. 7. Transformer architecture from [67]

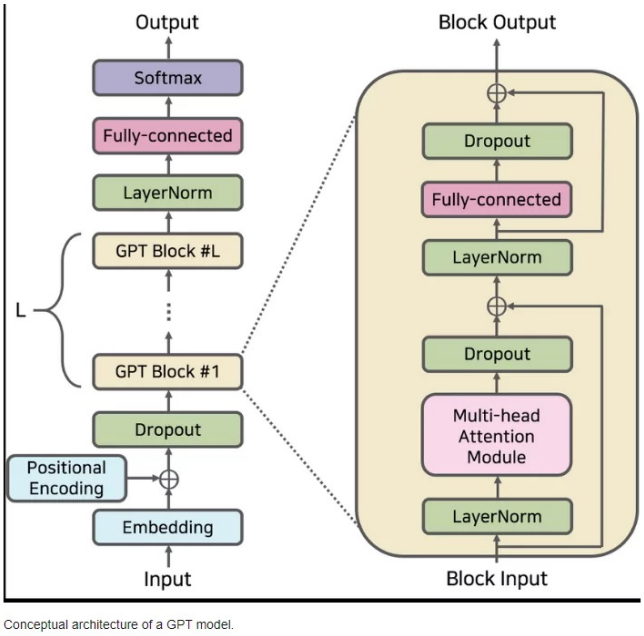

Fig. 8. GPT architecture from [41]

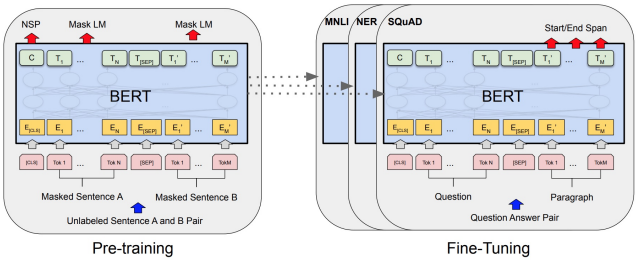

Fig. 9. BERT architecture from [23]

it can focus on different parts of the input simultaneously [27]. To overcome a Transformer limitation of the order of input elements, GPT uses positional encodings. GPT also has a "layer-wise structure", a stack of layers that each has its own parameters, as shown in Figure 8.

4.2.2 BERT. BERT, Bidirectional Encoder Representations from Transformers, was introduced in a paper in 2019 by Devlin et. al [23]. This novel approach uses bidirectional training of text by looking at the full input from both the left and the right directions. BERT uses a "masked language model" (MLM) to mask some of the input tokens, then predict the masked token based on the context. BERT is made up of two major steps: pre-training and fine-tuning. Pre-training involves unlabeled data where fine-tuning uses the pre-trained parameters and the output labeled data. Both these steps use a unified architecture, a "multi-layer bidirectional Transformer encoder". BERT is an encoder-only model, predicting tokens based on the input data and the context from both sides of that data, as shown in Figure 9.

4.2.3 RoBERTa. Robustly Optimized BERT Approach (RoBERTa) is an advanced version of BERT where the model was trained for a longer period of time and on longer sequences, the next sentence prediction objective was removed, and the masking pattern applied was changed [44]. Liu et al. [44] described the masking pattern used as "dynamic" as compared with the static pattern used by BERT, generating the masking pattern for every input sequence. The authors also described the Next Sentence Prediction (NSP) loss used in BERT where the model predicts if input segments are from the same or different documents. Liu et al. proposed that NSP loss was not as crucial as claimed, doing experiments on segments with and without NSP loss. Results showed that the best performance occurs with input sequences from a single document and without NSP loss. The authors concluded that RoBERTa outperforms BERT with its use of longer training time, bigger training batches over larger training sets, removing the NSP loss, and using dynamic masking.

4.2.4 BART. In 2019, Lewis et al. [42] introduced BART, a LLM that combines an encoder and a decoder into the architecture, combining BERT and an auto-regressive Transformer model. BART uses a

noising function to randomly shuffle the original text order as well as masking arbitrary lengths of the input text. It uses a bidirectional encoder similar to BERT, then a left-to-right decoder similar to GPT. BART is trained by corrupting documents, then reconstructing them and examining the difference to produce the loss function. The text corruption could be token masking or deletion, sentence permutation or rotation, or text infilling. Results showed that token masking was the most successful form of pre-training, as well as left-to-right pre-training.

4.2.5 Llama. In 2023, Meta AI came out with a novel LLM called Llama, with a couple iterations of Llama-1 and Llama-2 [66]. These models range in parameter sizes of 7 to 70 billion, train on more data and longer context length, and use an optimized Transformer architecture. As shown in Figure 10, Llama uses a Root Mean Square Layer Normalization (RMSNorm) instead of a traditional normalization layer, making it computationally simpler and more efficient. Llama also does the normalization step before other major layers instead of after, adding to training stability. Another architectural difference in the Llama model is the use of a grouped-query attention method (GQA) instead of a multi-head or multi-query method during the attention layer. GQA produces similar results to multi-head attention but is similar in speed to multi-query attention.

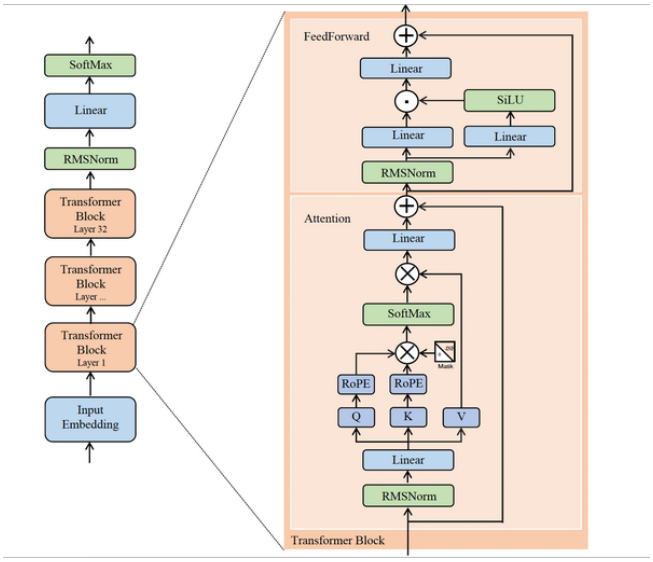

Fig. 10. Llama-2 architecture from [3]

4.2.6 Claude. Claude is a LLM developed by Anthropic in 2023 that is based on the Transformer architecture, but also uses a novel Constitutional AI to train the model using ethical principles as its guide [39]. It uses unsupervised learning as well as reinforcement learning with human feedback (RLHF), then adds in the additional constitutional AI layer. The constitutional layer is used during the supervised learning phase as well as during the reinforcement learning phase, as shown in Figure 11.

4.2.7 Gemini. Gemini, formerly known as Bard, is a LLM developed by Google. Reid et al. presented this LLM in a paper, calling it

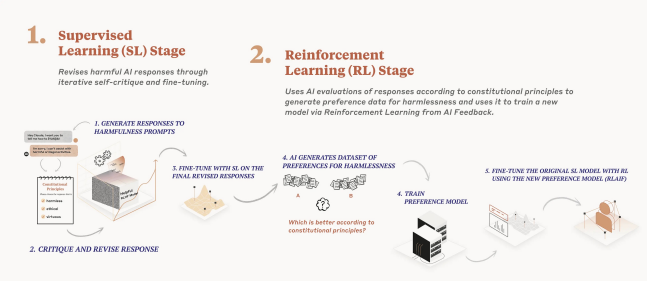

Fig. 11. Claude architecture from [8]

a "multimodal mixture-of-experts model" [56]. With millions of tokens, Gemini (Generalized Multimodal Intelligence Network [51]) is able to parse input from text, video, images, audio, graphs, and 3D sources, encoding the various inputs into a format the model can understand agnostically. The decoder is able to decode into the modality of choice by the user. The multimodal encoder and decoder allow the model to eliminate a fine-tuning stage. See Figure 12.

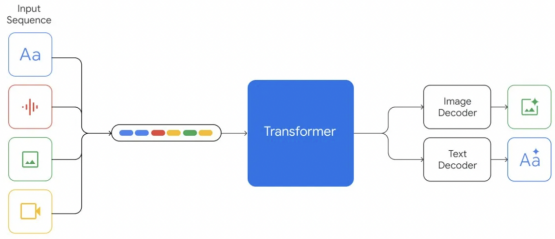

Fig. 12. Gemini high level architecture from [12]

4.3 Recent advances

In more recent work, advances are being made in LLM research to improve performance, dependability, and content. This section describes new LLM models and techniques of Retrieval Augmented Generation, TinyLlama, knowledge graphs, RAISE, AnyGPT, and AttentionLego.

4.3.1 RAG. In a paper by Fayyazi et al. [57], encoder-only and decoder-only models were used in conjunction with Retrieval Augmented Generation (RAG) to study the accuracy of models in the cybersecurity field and their analysis of "cyberattack procedure descriptions". The authors discussed the tendency of LLMs to hallucinate and output incorrect data, so RAG techniques were used to give the model the most relevant documents as further input from a vector database. The research involved testing output on encoder-only LLMs, decoder-only LLMs, and using RAG with decoder-only LLMs.

For encoder-only LLMs the authors used RoBERTa and SecureBERT, fine-tuning them with curated cybersecurity data. For decoder-only analysis, the authors used GPT-3.5-turbo-1106 and a generic prompt: *"You are a cybersecurity expert. Knowing that «procedure», what*

MITRE ATTCK tactics will a cyber adversary achieve with this technique? Please only respond with the MITRE ATTCK tactics you are certain about."

When adding RAG, the authors inserted into the prompt three procedures similar to the procedure under question. The analysis done involved both "recall" and "precision", how well the model was able to map the input to the correct output while keeping a low hallucination rate. Results showed that SecureBERT performed better of the encoder-only models, while decoder-only plus RAG outperformed decoder-only. The authors found that the encoder-only model performed better for precision, while the decoder-only model performed better for recall, showing that hallucinations tend to come from the decoding stage. Fayyazi et al. also found that RAG, while helpful, can also be a distraction in the model's output, causing it to focus too much on the extra content provided. Overall the authors concluded that the decoder-only LLMs with RAG performed the best, but noted that encoder-only LLMs required fewer resources. It was also proposed that LLMs of the future should "improve precision without compromising recall", allowing for less hallucinations while correctly interpreting cybersecurity procedures.

4.3.2 TinyLlama. Zhang et al. [73] created an open-source model called TinyLlama, based on the Llama 2 model but small in size. This work focused on training a smaller model on a large number of tokens, specifically 1.1B parameters and 3 trillion tokens on a Transformer decoder-only model. TinyLlama was pre-trained on 950 billion tokens with a mixture of natural language and code data. The architecture was similar to Llama 2 while including a grouped-query attention layer so key and value data could be shared across multiple heads. There were also speed and attention optimizations made in the architecture. The authors found that this smaller model performed as well as competitors, but with the smaller architecture would be an excellent use for mobile devices or for testing new LLM ideas.

4.3.3 Knowledge Graphs. In another work, Pan et. al [53] reviewed the idea of LLMs and Knowledge Graphs (KGs) as a way of adding more knowledge to the LLM. The authors explained that LLMs tend to be "black-box models" while KGs are difficult to evolve, so it was proposed to combine the two ideas into one, a KG-enhanced LLM. Due to the lack of "factual knowledge", LLMs can struggle with recalling facts and therefore generate hallucinations instead. The authors explained that KGs can be created with domain-specific content to help the LLM be more dependable. The pros and cons of both LLMs and KGs are shown in Figure 13.

In the paper by Pan et. al [53], the authors described three variations of combining LLMs and KGs: a KG-enhanced LLM, a LLM-augmented KG, and a synergized LLM plus KG, as shown in Figure 14. KGs are able to store structured knowledge in the form of triples that represent the entities and their relations. KGs are often filled with encyclopedic content, domain-specific content, common sense content, or multi-modal content. A KG-enhanced LLM could utilize a KG in the pre-training or the inference stages of a LLM, depending on the application. The authors also researched involving a KG in the process of trying to understand the inner workings of the LLM. LLMs could complement KGs by adding encoding and embedding

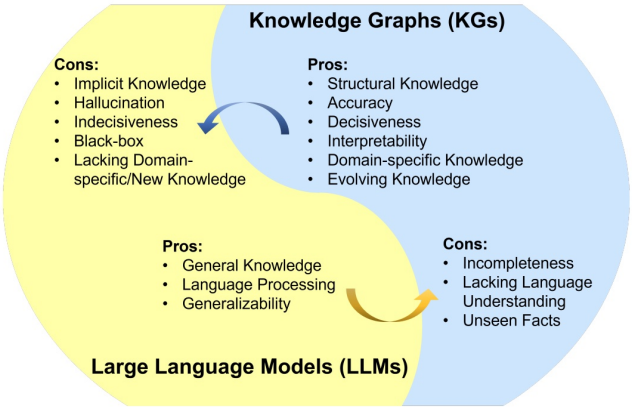

Fig. 13. LLMs versus KGs from [53]

functionality to the KG. LLMs could also be used in the various stages of creating a KG. Lastly, the authors researched a synergized LLM plus KG approach with LLMs understanding natural language and KGs being the knowledge base. One approach researched used a T-encoder to encode the input, then a K-encoder to encode the KG, then fused the encodings together for a "synergized knowledge representation". Another approach researched was "synergized reasoning", where the model could reason using both LLMs and KGs. The architecture could include a separate encoder for the LLM and the KG, then an additional reasoning module. The architecture could alternatively include the LLM as an "agent for reasoning" on the KG.

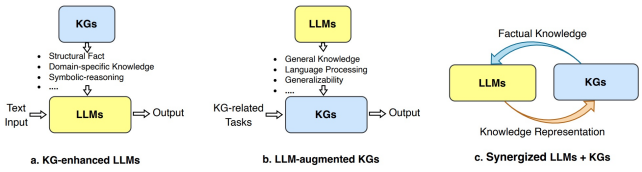

Fig. 14. LLMs with KGs from [53]

The authors concluded with some future direction in the area of unifying LLMs and KGs. KGs could be used to help detect hallucinations in LLMs. KGs could also help with editing the internal knowledge inside LLMs. LLMs could be used to add multi-modal support to KGs. A final note made by the authors was to truly integrate LLMs and KGs by developing a KG structure that can be directly input into the LLM. Combining the understanding of natural language with that of structured knowledge representation could benefit many diverse applications.

4.3.4 RAISE. Liu et al. [43] introduced an architecture to help with the integration of LLMs and conversational agents called RAISE, Reasoning and Acting through Scratchpad and Examples. RAISE was based on the existing ReACT framework [71] but added a "dual-component memory system" to help with short-term and long-term memory abilities for more continuous dialogue. The authors

performed their analysis in the real estate domain but concluded that RAISE could be used in any context. Results showed that adding examples and scratchpads to the existing ReACT framework were effective, and that fine-tuning was a better approach than prompting. The architecture of RAISE is shown in Figure 15.

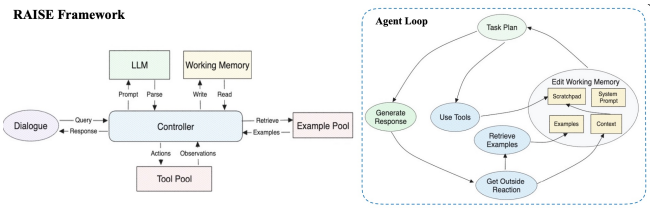

Fig. 15. RAISE architecture from [43]

4.3.5 AnyGPT. Zhan et al. [72] proposed a multi-modal LLM solution called AnyGPT that could generate any modality from any modality. This idea would allow for input speech, text, image, and music to each be encoded and decoded in discrete ways, while leaving the existing LLM architecture unmodified. The authors used training data made up of 108k conversations that included multiple modalities, conversations that included text, speech, images, and music. AnyGPT was not fine-tuned or pre-trained for its experiments, using a zero-shot model and showing more general results. By using discrete representations of all the modalities, the authors showed that adding new modalities would be seamless. Results showed that AnyGPT worked well in all the cross-modal experiments performed, and that using discrete representations of all the data was an effective way to unify all the modalities. The AnyGPT architecture is shown in Figure 16. The authors described limitations of the work, including its novelty and therefore a lack of benchmarks in the area. AnyGPT had a higher loss compared to models that train on a single modality and could be trained on longer content and more enhanced tokenizers.

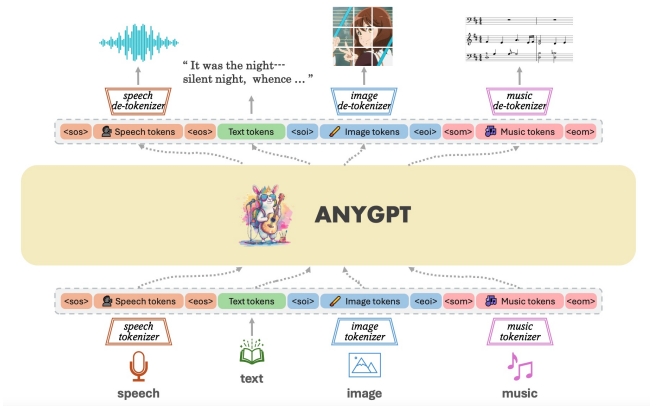

Fig. 16. AnyGPT architecture from [72]

4.3.6 AttentionLego. Cong et al. [21] described a novel self-attention accelerator called AttentionLego that uses Processing-In-Memory (PIM) technology to help with the I/O bandwidth demand that LLMs enforce and helping with efficiency. The authors proposed AttentionLego to be a building block for "spatially expandable LLM processors". PIM technology allows for the processing units and the memory to be located on the same chip, therefore eliminating the transfer of data between them. AttentionLego is comprised of five parts: the input process module, the score module, the Softmax module, the Direct Memory Access (DMA) module, and the top controller. The architecture of AttentionLego can be seen in Figure 17.

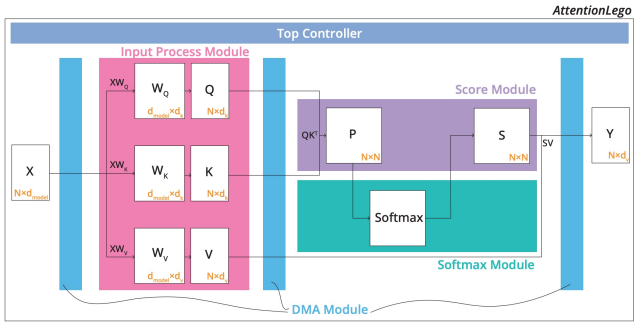

Fig. 17. AttentionLego architecture from [21]

5 DEPLOYMENT

Once a LLM is built, there are deployment strategies to consider to minimize latency, cost, resources, and privacy concerns. The actual hardware on which the model runs as well as the parameters of the model will affect its cost and speed. The model should have enough capacity to run as well as be scalable for different use cases. Data must be unbiased and secure [16].

When considering deployment challenges, there are several aspects of the LLM. Generating a response from the LLM can have a high latency depending on the requested task. The length of the input prompt can also affect the amount of work the LLM has to do since the longer the prompt, the more input tokens have to be created and sent into the model. Some models keep a key/value cache of previously generated tokens, but this requires more memory. Two main challenges to keep in mind are the LLM memory requirement and choosing the best scheduling strategy for the LLM to respond optimally [14].

In a typical LLM, memory is needed for the model parameters, caching, retrieval augmented generation requirements, and input tokens. To help reduce memory requirements, model compression can be done with distillation, pruning, or quantization. In quantization, the model weights are represented with lower precision values, such as using one byte instead of four. Attention layers can also be optimized to reduce the amount of data movement. Scheduling of requests can also be optimized for the best user experience, for example by doing batch level scheduling, batching requests and handling them at the same time. Iteration level scheduling is a process of scheduling tokens one at a time, adjusting batch sizes dynamically depending on the request [14].

Memory bandwidth is also a concern since constant data movement across GPUs will affect performance. Data parallelism can help the model handle more requests as they are sent in parallel, but the model must fit on a single GPU. For multiple GPUs, tensor parallelism can be used to split the model such that each GPU computes part of the result, then all the partial results are synchronized in a final step. Pipeline parallelism is another splitting approach that does not require synchronization but instead the GPUs work as a pipeline, passing output from one GPU to the next GPU as input. There can be a decrease in throughput, though, since the GPUs have to work serially [14].

In a novel paper by Park et al. [55], the authors introduce Any-Precision LLM to reduce the deployment costs of multiple LLMs of different sizes. The authors proposed a solution of overlaying varying LLMs into one memory footprint of the largest bit-width LLM. The authors noted that to achieve this LLM, Any-Precision quantization needs to be improved and a GPU that can do quantized matrix-vector multiplication needs to be developed. A novel approach was proposed that the authors called "incremental upscaling", shown in Figure 18.

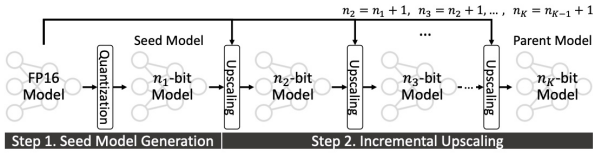

Fig. 18. Incremental Upscaling from [55]

The authors also proposed a bitplane-based representation of the model's quantized weights, as opposed to a more commonly used bitpacking-based representation, as shown in Figure 19. With this variant representation, Any-Precision LLM deployment can be achieved more efficiently since any runtime request would load only the specified amount of bits needed. The authors concluded that Any-Precision LLM would be both memory-efficient and cost-saving in the deployment of multiple varying-sized LLMs.

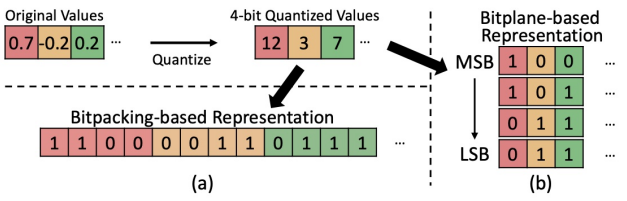

Fig. 19. Bitpacking versus bitplane from [55]

Overall, the deployment of a LLM must consider the complexities of the model, the cost, the performance, and security and privacy concerns. Before deployment, security assessments should be performed to identify any vulnerabilities in the LLM [62].

6 USE CASES AND CHALLENGES

Artificial neural networks are in use more today than ever before; research into their architectures is ongoing. In this section, some of the architectural decisions will be compared and contrasted looking at the pros and cons of various models as well as their use cases.

In general, LLMs have been hailed for being able to deliver general knowledge and perform language processing. LLMs enable the automation of data generation as well as the ability to develop an understanding of language context. LLM parameter and pre-training datasets continue to grow, helping the overall model performance and accuracy. On the flip side, LLMs are computationally expensive, causing a large carbon footprint and an expense that smaller companies cannot afford. As the model architectures become more complicated, so does the understanding, making the model more of a black box. Because LLMs tend to learn generalized knowledge, they also lack domain-specific or new and evolving knowledge. The models can also have hallucinations, producing inaccurate or offensive results. There are ethical and privacy considerations that need to be considered, too [34]. For models that add a fine-tuning layer for domain-specific tasks, there is much more data required for training which can lead to overfitting of the model to the narrow fine-tuned data.

6.1 RNN

Recurrent neural networks (RNNs) are best suited for sequential data analysis, for example doing translations of text from one language to another or doing time-series prediction. Data in a RNN from the previous step is passed into the next step, so the network can remember all the information through time. RNNS, though, are slow to train, cannot handle long sequences, and have an "exploding gradient" problem where the slope of the gradient grows exponentially because the model weights get too large [1]. The Long Short-Term Memory (LSTM) variation of the RNN is able to overcome the exploding gradient problem because it works to "read-write-and-forget", reading the input data and only saving the most useful information for predicting the output. It is able to keep memory over longer sequences, but it is still slow to train and is more complex with additional gates. Both RNNs and LSTM networks are sequential and therefore do not make use of the parallel computation of modern GPUs.

6.2 Transformer

The Transformer architecture is a neural network architecture that overcomes slow and sequential formats of the RNN. The Transformer is faster because it can pass input into its encoder and decoder in parallel. It does not have the concept of a time step as it determines word embeddings and context simultaneously. Different from the previously mentioned neural network architecture, the Transformer does not use recurrent layers but instead the attention layer as its fundamental layer. This layer allows the Transformer to pay more attention to the input data that is the most meaningful. This layer allows the model to preserve the overall context of the input, unlike RNNs and LSTM. However, Transformers are also more expensive and use more memory. They are more complicated,

making it very difficult to gain insight into how the model produces its output [52]. Transformers need to be pre-trained on massive amounts of labeled data to be able to learn effectively.

6.2.1 BERT. BERT uses the Transformer model but with encoders only, pre-training the model with data, then fine-tuning it for a specific task. Fine-tuning uses supervised learning where the input is both a question and a passage with the answer, the output is only the answer. This fine-tuning step makes BERT a useful model for handling specific tasks. BERT excels in tasks such as text classification because of its bidirectional nature. It also was a forerunner in transfer learning methods with its fine-tuning step. A weakness of BERT is how it accomplishes tokenization, splitting words into sub-tokens which could cause a loss of context. BERT also is very computationally expensive, especially during the fine-tuning stage [35]. BERT is used for text categorization, sentiment analysis, question and answering, and language modeling.

6.2.2 RoBERTa. RoBERTa is a "robustly optimized" version of BERT, training with more data and for a longer training duration. This helps with contextual understanding and generalization of the model. RoBERTa, however, still has the sub-token weakness of BERT, and is also resource intensive [35]. Like BERT, RoBERTa is used for text categorization, sentiment analysis, question and answering, and language modeling.

6.2.3 GPT-3. GPT-3 uses the Transformer architecture, but with a decoder-only model working in an auto-regressive way such that only the tokens of the past can be seen at any one time. It prepares the model with 175 billion parameters, allowing for capabilities in language translation, question answering, and text generation [35]. It is able to understand context as well as generalize. Given some input, the model generates output, then calculates the error between the actual and expected outputs, updating the model's parameters and weights with this error difference. GPT-3 has a maximum input length of 2048 tokens, each one represented as a vector that travels through 96 layers of Transformer decoders processed sequentially in smaller chunks. It performs both pre-training and fine-tuning, allowing it to have knowledge and work well on specific tasks. GPT-3 also utilizes 0-shot, 1-shot, and few-shot learning where the input prompt includes zero, one, or more than one example with the prompt. Weaknesses of GPT-3 are related to its enormous size which makes it expensive to train and to deploy. GPT-3 also relies on patterns in the data so can be factually incorrect in its responses. This powerful model also still needs to consider ethics, privacy, and harmfulness in its responses. GPT-3 has been used as a chatbot, for personal recommendations, for text and code generation, for language translation, for searches, and much more.

6.2.4 BART. BART is a combination of BERT and GPT, a Transformer architecture with both an encoder and a decoder. BART was shown to perform similarly to RoBERTa, but needs more exploration in how it corrupts the input documents during pre-training. BART has been used for document summarizations and creating grammatically correct output from noisy input [63].

6.2.5 Llama. The Llama architecture follows a decoder-only Transformer model while including a pre-normalization step, an extra activation function, and a "Rotary Position Embedding (ROPE)" to give it better performance [30]. The newest Llama-2 model has up to 70 billion parameters, trains on 40% more data and more context length than Llama-1, and adds a fine-tuning step. Llama is also cognisant of the carbon footprint, reducing the CO2 by 100 tons. It is being shown to outperform GPT-3 with less parameters and a smaller architecture but more training data. Like GPT-3, though, it is large in size and therefore expensive to run, possibly making it not as accessible to some users. Llama-2 is used for generating safe and non-harmful text as well as doing text summarization.

6.2.6 Claude. Claude is an architecture that is based on a Constitutional AI paper and is completely self-supervised with no human involvement. It is fed the rules of the constitution, then performs reinforcement learning. It can process up to 200,000 pieces of information at a time, making it able to hold more context and hold longer conversations as a chat agent. Claude 3 is not able to understand code or generate images, and it also puts limitations on persona modeling to ensure that it is following the constitution [31]. Like other models, Claude is used for content generation, classification, and search.

6.2.7 Gemini. Gemini uses a novel multi-modal encoder and decoder as part of its architecture, allowing for varying types of output no matter what the input type. This facilitates communication across the modalities throughout the process. For example, Gemini can take an image as input and output a textual description of that image. It is able to process "multiple millions of tokens" [56], allowing it to respond accurately and precisely. Gemini is used for text processing, generation, and translation, but also for image and video understanding, and multi-modal reasoning.

6.2.8 Most recently. In more recent models reviewed, suggestions were made by authors that would increase model accuracy, generalizability, and performance. Zhan et al. [72] described AnyGPT as being capable of going from any modality input to any modality output, making it more generalizable. However, their performance data was so novel that there were not other comparable benchmarks to truly measure against. Cong et al. [21] described an advanced attention layer, AttentionLego, that would co-locate the processing units and the memory on the same chip, reducing the data bandwidth and increasing performance. This Processing-In-Memory technique is both power and energy saving but needs more quantitative analysis. Zhang et al. [73] proposed a smaller scale Llama architecture dubbed TinyLlama, making speed and attention optimizations to the Llama architecture. This model was found to perform well but would be better suited for smaller devices such as mobile.

To help improve model accuracy, Pan et al. [53] suggested the intermingling of a LLM and knowledge graphs that are easier to comprehend. With the use of KGs, it would be easier to detect hallucinations and inaccuracies in the model, then easier to update the internal knowledge. The authors suggested a model that could directly interpret a KG as part of its training data. Liu et al. [43] proposed adding examples and scratchpads to the model training as well as to the fine-tuning stage to increase model accuracy. This work was specifically done in the real estate domain, but the authors called for its use in other domains, too. Fayyazi et al. [57]

proposed adding a vector database of relevant information in addition to training data, known as Retrieval Augmented Generation, to improve the model's accuracy. This work was done in the cybersecurity domain, enhancing the model with a database of cybersecurity procedures.

In a novel paper by Xuchen Suo [65], the issue of prompt injection attacks in LLMs was reviewed, suggesting a "Signed-Prompt" method to encrypt the important commands of the input prompt so it cannot be attacked. This model would include an encoder module to sign the user instructions as well as LLM modifications to be able to understand the signed instructions, thus differentiating between legitimate users and external attackers (see Figure 20). The authors proposed more research in this defense strategy area of LLMs, enhancing existing strategies as well as improving the performance of "Signed-Prompt".

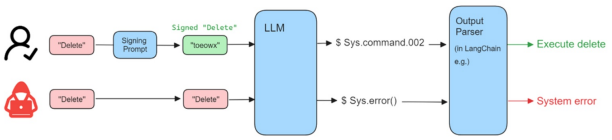

Fig. 20. Signed-Prompt Example from [65]

7 DISCUSSION AND FUTURE WORK

In most of this literature review, we have looked at papers surrounding various LLM architectures from past years as well as recent work. Each architecture has advantages and disadvantages, and each new LLM release attempts to address disadvantages while introducing new benefits in a novel way.

Starting with the recurrent neural network (RNN) with an encoder, decoder, and hidden layers [19], this architecture contains a sequential format and the ability to remember or forget information, thus capturing both long-term and short-term dependencies. This architecture works well for translation tasks, but its sequential nature poses limitations as the input lengths grow and require batching due to memory constraints.

The novel Transformer architecture [67] offers a solution that removes the recurrence of RNNs and adds attention mechanisms. This architecture is parallelizable and takes less time to train. The constant number of sequential operations make the attention layers faster than recurrent layers that have variable sequential operations.

All the models researched in this paper are then based on the Transformer, each adding a unique spin to the model. Models select the components that are best suited for a particular use case, such as a decoder-only Transformer, an encoder-only Transformer, or a combination. Models use varying numbers of parameters, train on varying amounts of data, and train on varying context sizes and lengths of time. Each of these choices affects the model complexity, cost, efficiency, and ethical considerations. Some models are open source and train on publicly available data while others are more of a black box. The data quality used in pretraining is crucial as it could introduce biased or harmful output from the model. Due to the black box nature of LLMs, the output can be difficult to explain. Adding more visibility into how a model is generating its output could be a major factor in reducing harmful and biased results.

Related to the data, a common limitation of the researched LLMs is the difficulty in updating the model with the most current knowledge. When a model needs to be updated with new knowledge, it requires more pretraining and more fine-tuning. Fayyazi et al. [57] suggested a Retrieval Augmented Generation (RAG) system where new knowledge could be retrieved from a database at various stages of the architecture before the final output response is generated. Perhaps RAG could be added to both the prompt and the fine-tuning stages for optimal recall and precision. Pan et at. [53] explored the idea of enhancing a LLM with knowledge graphs that could hold domain-specific up-to-date knowledge, allowing the LLM to access the latest knowledge without retraining. Liu et al. [43] enhanced a conversational LLM agent with external tools to guide the LLM in its reasoning.

Another common limitation of LLMs is the tendency for hallucinations, where the LLM outputs contradictions or non-sensical information. This can be mitigated with clear and specific prompts as well as with adjusting LLM parameters that control the randomness of the output. The work of Fayyazi et al. [57] and of Pan et al. [53] offer possible solutions with their domain-specific knowledge injections into the prompt or the fine-tuning stages. More studies could be done in the area of prompt engineering, researching the LLM prompts in an effort to produce the most accurate output. Prompts can include zero or more examples, too, offering the model guides to imitate with an effort to reduce hallucinations.

Detecting hallucinations is a manual process that can no longer scale with the amount of context LLMs are producing. Models like Gemini [12] and AnyGPT [72] are multi-modal, allowing for the generation of text, audio, video, music, and more. As LLMs continue to expand, these hallucinations need further consideration to ensure safe, accurate, and unbiased content is being generated. One thought would be to create LLMs that work in an adversarial way, where one LLM generates content and the other validates it until there are no more hallucinations.

In this literature review, we looked at papers surrounding various LLM architectures from past years as well as recent work. To present these papers, we first explained the history and fundamentals of LLMs, followed by a deep dive into the development life cycle of a LLM. We were then able to better present some well-known Transformer architectures, followed by very recent advances being made in LLM architectures. Deployment strategies were discussed, followed by the pros and cons of the various models. In reviewing over ten LLMs in one paper, we were able to better understand the differing architectures and nuances of each model as well as the challenges faced. With this better understanding, we hope to continue innovation in the field of LLMs.

8 ACKNOWLEDGMENTS

The author thanks Dr. Bo Wu for advice on the direction of this literature review, resulting in a deep exploration and study of how LLMs work and the challenges faced. She also thanks Dr. Christine Liebe for advice on how to research a topic of interest, how to read a paper, and how to write a literature review. She also thanks fellow PhD students Ben Wagley and Xiangyu Li for their help in clarifying some of the nuances in LLMs and for being a sounding board.

REFERENCES

[1] [n. d.]. Introduction to Recurrent Neural Network. *GeeksForGeeks* ([n. d.]). https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/
[2] [n. d.]. Large Language Models 101: History, Evolution and Future. *scribbleData* ([n. d.]). https://www.scribbledata.io/blog/large-language-models-history-evolutions-and-future/
[3] [n. d.]. Llama 2 detailed explanation. *Zhihu* ([n. d.]). https://zhuanlan.zhihu.com/p/649756898
[4] [n. d.]. The math behind Attention: Keys, Queries, and Values matrices. https://www.youtube.com/watch?v=UPtG_38Oq8o
[5] [n. d.]. What are loss functions in machine learning? https://video.search.yahoo.com/search/video?fr=mcafee&p=what+is+a+loss+function+machine+learning&type=E210US91217G0#id=3&vid=ff14c4ae56089bfc30b1097d6d2b7322&action=click
[6] [n. d.]. What is Layer Normalization? https://www.youtube.com/watch?v=2V3Uduw1zwQ
[7] 2020. ELMo Embeddings. *John Snow LABS* (2020). https://sparknlp.org/2020/01/31/elmo.html#:~:text=The%20complex%20architecture%20achieves%20state%20of%20the%20art,word%20embedding%20modules%20that%20only%20perform%20embedding%20lookups.
[8] 2023. Claude's Constitution. https://www.anthropic.com/news/claudes-constitution
[9] 2023. Data Preparation for Machine Learning: The Ultimate Guide to Doing It Right. *Pecan* (2023). https://www.pecan.ai/blog/data-preparation-for-machine-learning/#:~:text=You%20start%20with%20collecting%20data%20from%20various%20sources,to%20make%20it%20compatible%20with%20machine%20learning%20algorithms.
[10] 2023. How to Train a Machine Learning Model: The Complete Guide. *ProjectPro* (2023). https://www.projectpro.io/article/training-a-machine-learning-model/936
[11] 2023. Machine Learning Models: What THey Are and How to Build Them. *Coursera* (2023). https://www.coursera.org/articles/machine-learning-models
[12] 2024. Gemini Pro vs GPT-4: An in-depth comparison of LLM models. *Bind AI* (2024). https://blog.getbind.co/2024/01/07/google-gemini-pro-vs-gpt-4-llm-models/
[13] 2024. Transfer Learning in Large Language Models: A Game Changer in AI. *Expand My Business* (2024). https://blog.emb.global/transfer-learning-in-large-language-models/
[14] Sherif Akoush. 2023. Deploying Large Language Models in Production: LLM Deployment Challenges. *Seldom* (2023). https://www.seldon.io/deploying-large-language-models-in-production-llm-deployment-challenges
[15] Sandi Besen. 2023. LLM Embeddings - Explained Simply. *aimind* (2023). https://pub.aimind.so/llm-embeddings-explained-simply-f7536d3d0e4b
[16] Viktor Bezdek. 2023. A Guide to Deploying Large Language Models (LLMs). *medium.com* (2023). https://medium.com/@viktorbezdek/a-guide-to-deploying-large-language-models-llms-8f172e211f02
[17] Bala Priya C. 2023. Build Better Deep Learning Models with Batch and Layer Normalization. *Pinecone* (2023). https://www.pinecone.io/learn/batch-layer-normalization/
[18] Matt Casey. 2023. Large language models: their history, capabilities and limitations. *Snorkel* (2023). https://snorkel.ai/large-language-models-llms/
[19] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv:1406.1078 [cs.CL]
[20] Paul Christiano. 2023. Large Language Model Training in 2023: A Practical Guide. *Expert Beacon* (2023). https://expertbeacon.com/large-language-model-training/
[21] Rongqing Cong, Wenyang He, Mingxuan Li, Bangning Luo, Zebin Yang, Yuchao Yang, Ru Huang, and Bonan Yan. 2024. AttentionLego: An Open-Source Building Block For Spatially-Scalable Large Language Model Accelerator With Processing-In-Memory Technology. arXiv:2401.11459 [cs.AR]
[22] Kiel Dang. 2023. Language Model History - Before and After Transformer: The AI Revolution. *medium.com* (2023). https://medium.com/@kirudang/language-model-history-before-and-after-transformer-the-ai-revolution-bedc7948a130
[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]
[24] Vyacheslav Efimov. 2024. Large Language Models, GPT-1 - Generative Pre-Trained Transformer. *Towards Data Science* (2024). https://towardsdatascience.com/large-language-models-gpt-1-generative-pre-trained-transformer-7b895f296d3b
[25] Josep Ferrer. 2024. An Introductory Guide to Fine-Tuning LLMs. *datacamp* (2024). https://www.datacamp.com/tutorial/fine-tuning-large-language-models
[26] Keith D. Foote. 2023. A Brief History of Large Language Models. *Dataversity* (2023). https://www.dataversity.net/a-brief-history-of-large-language-models/
[27] Olga Green. 2023. Understanding the GPT Architecture: A Deep Dive into Generative Pre-trained Transformers. *Stackademic* (2023). https://blog.stackademic.com/understanding-the-gpt-architecture-a-deep-dive-into-generative-pre-trained-transformers-b4a9d2f3f5c0
[28] Abhishek Gupta. 2023. Transformers and Attention Mechanism: The Backbone of LLMs. (2023). https://blog.acethecloud.com/transformers-and-attention-mechanism-the-backbone-of-llms-blog-3-10-bfba00fcded6
[29] Minhajull Hoque. 2023. Demystifying Neural Network Normalization Techniques. *medium.com* (2023). https://medium.com/@minh.hoque/demystifying-neural-network-normalization-techniques-4a21d35b14f8
[30] Julian Horsey. 2023. Llama 1 vs Llama 2 AI architecture compared and tested. *Geeky Gadgets* (2023). https://www.geeky-gadgets.com/llama-1-vs-llama-2/
[31] Julian Horsey. 2024. Pros Cons of Claude 3 AI compared to ChatGPT. *Geeky Gadgets* (2024). https://www.geeky-gadgets.com/claude-3-vs-chatgpt/
[32] Michael Humor. 2023. Understanding "tokens" and tokenization in large language models. (2023). https://blog.devgenius.io/understanding-tokens-and-tokenization-in-large-language-models-1058cd24b944
[33] Gwyneth Iredale. 2021. Tokenization Algorithms in NLP. *101 Blockchains* (2021). https://101blockchains.com/tokenization-nlp/
[34] Alex Ivankov. 2023. Advantages and Disadvantages of Large Language Models. *Profolus* (2023). https://www.profolus.com/topics/advantages-disadvantages-of-large-language-models/
[35] Liva Jorge. 2023. GPT-3, BERT, and RoBERTa | AI Model Analysis Comparison. *medium.com* (2023). https://medium.com/@livajorge7/gpt-3-bert-and-roberta-ai-model-analysis-comparison-7dfab049367d
[36] Bhuvi Kathpalia. [n. d.]. Introduction to Large Language Models (LLMs). *Leena AI* ([n. d.]). https://leena.ai/blog/large-language-models-llms-guide/
[37] Kashyap Kathrani. 2020. All about Embeddings. *medium.com* (2020). https://medium.com/@kashyapkathrani/all-about-embeddings-829c8ff0bf5b
[38] Ruhma Khawaja. 2023. Demystifying embeddings 101 - The foundation of large language models. *DataScienceDojo* (2023). https://datasciencedojo.com/blog/embeddings-and-llm/#
[39] Hal Koss. 2024. What Is Claude AI and How Does It Compare to ChatGPT? *built-in* (2024). https://builtin.com/articles/claude-ai
[40] Simeon Kostadinov. 2017. Understanding GRU Networks. *Towards Data Science* (2017).
[41] Minhyeok Lee. 2023. A Mathematical Investigation of Hallucination and Creativity in GPT Models. *Mathematics* 11 (05 2023), 2320. https://doi.org/10.3390/math11102320
[42] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. arXiv:1910.13461 [cs.CL]
[43] Na Liu, Liangyu Chen, Xiaoyu Tian, Wei Zou, Kaijiang Chen, and Ming Cui. 2024. From LLM to Conversational Agent: A Memory Enhanced Architecture with Fine-Tuning of Large Language Models. arXiv:2401.02777 [cs.CL]
[44] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL]
[45] Kun Luo, Zheng Liu, Shitao Xiao, and Kang Liu. 2024. BGE Landmark Embedding: A Chunking-Free Embedding Method For Retrieval Augmented Long-Context Large Language Models. arXiv:2402.11573 [cs.CL]
[46] Michael McDonough. 2024. large language model. *Britannica* (2024). https://www.britannica.com/topic/large-language-model#ref373414
[47] Akshit Mehra. 2023. Data Collection and Preprocessing for Large Language Models. *Labellerr* (2023). https://www.labellerr.com/blog/data-collection-and-preprocessing-for-large-language-models/
[48] Cristian Munoz. 2023. Overview of Large Language Models: From Transformer Architecture to Prompt Engineering. *Holistic AI* (2023). https://www.holisticai.com/blog/from-transformer-architecture-to-prompt-engineering
[49] Samarpit Nasa. 2023. Addressing Overfitting and Underfitting in LLM Training. *appypie* (2023). https://www.appypie.com/blog/overfitting-and-underfitting-llm-training

[50] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2024. A Comprehensive Overview of Large Language Models. arXiv:2307.06435 [cs.CL]

[51] Nomadev. 2023. Google's Gemini: The Next Big Thing in AI Revolution. *Nomadev* (2023). https://dev.to/thenomadevel/googles-gemini-the-next-big-thing-in-ai-revolution-17a4

[52] Jean Nyandwi. 2023. The Transformer Blueprint: A Holistic Guide to the Transformer Neural Network Architecture. *AI Research Blog* (2023). https://deeprevision.github.io/posts/001-transformer/

[53] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering* (2024), 1–20. https://doi.org/10.1109/TKDE.2024.3352100

[54] Harika Panuganty. 2023. From Words to VEctors: Inside the LLM Transformer Architecture. *medium.com* (2023). https://medium.com/@harikapanuganty/from-words-to-vectors-inside-the-llm-transformer-architecture-50275c354bc4

[55] Yeonhong Park, Jake Hyun, SangLyul Cho, Bonggeun Sim, and Jae W. Lee. 2024. Any-Precision LLM: Low-Cost Deployment of Multiple, Different-Sized LLMs. arXiv:2402.10517 [cs.LG]

[56] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, Ioannis Antonoglou, Rohan Anil, Sebastian Borgeaud, Andrew Dai, Katie Millican, Ethan Dyer, Mia Glaese, Thibault Sottiaux, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, James Molloy, Jilin Chen, Michael Isard, Paul Barham, Tom Hennigan, Ross McIlroy, Melvin Johnson, Johan Schalkwyk, Eli Collins, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, Clemens Meyer, Gregory Thornton, Zhen Yang, Henryk Michalewski, Zaheer Abbas, Nathan Schucher, Ankesh Anand, Richard Ives, James Keeling, Karel Lenc, Salem Haykal, Siamak Shakeri, Pranav Shyam, Aakanksha Chowdhery, Roman Ring, Stephen Spencer, Eren Sezener, Luke Vilnis, Oscar Chang, Nobuyuki Morioka, George Tucker, Ce Zheng, Oliver Woodman, Nithya Attaluri, Tomas Kocisky, Evgenii Eltyshev, Xi Chen, Timothy Chung, Vittorio Selo, Siddhartha Brahma, Petko Georgiev, Ambrose Slone, Zhenkai Zhu, James Lottes, Siyuan Qiao, Ben Caine, Sebastian Riedel, Alex Tomala, Martin Chadwick, Juliette Love, Peter Choy, Sid Mittal, Neil Houlsby, Yunhao Tang, Matthew Lamm, Libin Bai, Qiao Zhang, Luheng He, Yong Cheng, Peter Humphreys, Yujia Li, Sergey Brin, Albin Cassirer, Yingjie Miao, Lukas Zilka, Taylor Tobin, Kelvin Xu, Lev Proleev, Daniel Sohn, Alberto Magni, Lisa Anne Hendricks, Isabel Gao, Santiago Ontañón, Oskar Bunyan, Nathan Byrd, Abhanshu Sharma, Biao Zhang, Mario Pinto, Rishika Sinha, Harsh Mehta, Dawei Jia, Sergi Caelles, Albert Webson, Alex Morris, Becca Roelofs, Yifan Ding, Robin Strudel, Xuehan Xiong, Marvin Ritter, Mostafa Dehghani, Rahma Chaabouni, Abhijit Karmarkar, Guangda Lai, Fabian Mentzer, Bibo Xu, YaGuang Li, Yujing Zhang, Tom Le Paine, Alex Goldin, Behnam Neyshabur, Kate Baumli, Anselm Levskaya, Michael Laskin, Wenhao Jia, Jack W. Rae, Kefan Xiao, Antoine He, Skye Giordano, Lakshman Yagati, Jean-Baptiste Lespiau, Paul Natsev, Sanjay Ganapathy, Fangyu Liu, Danilo Martins, Nanxin Chen, Yunhan Xu, Megan Barnes, Rhys May, Arpi Vezer, Junhyuk Oh, Ken Franko, Sophie Bridgers, Ruizhe Zhao, Boxi Wu, Basil Mustafa, Sean Sechrist, Emilio Parisotto, Thanumalayan Sankaranarayana Pillai, Chris Larkin, Chenjie Gu, Christina Sorokin, Maxim Krikun, Alexey Guseynov, Jessica Landon, Romina Datta, Alexander Pritzel, Phoebe Thacker, Fan Yang, Kevin Hui, Anja Hauth, Chih-Kuan Yeh, David Barker, Justin Mao-Jones, Sophia Austin, Hannah Sheahan, Parker Schuh, James Svensson, Rohan Jain, Vinay Ramasesh, Anton Briukhov, Da-Woon Chung, Tamara von Glehn, Christina Butterfield, Priya Jhakra, Matthew Wiethoff, Justin Frye, Jordan Grimstad, Beer Changpinyo, Charline Le Lan, Anna Bortsova, Yonghui Wu, Paul Voigtlaender, Tara Sainath, Charlotte Smith, Will Hawkins, Kris Cao, James Besley, Srivatsan Srinivasan, Mark Omernick, Colin Gaffney, Gabriela Surita, Ryan Burnell, Bogdan Damoc, Junwhan Ahn, Andrew Brock, Mantas Pajarskas, Anastasia Petrushkina, Seb Noury, Lorenzo Blanco, Kevin Swersky, Arun Ahuja, Thi Avrahami, Vedant Misra, Raoul de Liedekerke, Mariko Iinuma, Alex Polozov, Sarah York, George van den Driessche, Paul Michel, Justin Chiu, Rory Blevins, Zach Gleicher, Adrià Recasens, Alban Rrustemi, Elena Gribovskaya, Aurko Roy, Wiktor Gworek, Séb Arnold, Lisa Lee, James Lee-Thorp, Marcello Maggioni, Enrique Piqueras, Kartikeya Badola, Sharad Vikram, Lucas Gonzalez, Anirudh Baddepudi, Evan Senter, Jacob Devlin, James Qin, Michael Azzam, Maja Trebacz, Martin Polacek, Kashyap Krishnakumar, Shuo yiin Chang, Matthew Tung, Ivo Penchev, Rishabh Joshi, Kate Olszewska, Carrie Muir, Mateo Wirth, Ale Jakse Hartman, Josh Newlan, Sheleem Kashem, Vijay Bolina, Elahe Dabir, Joost van Amersfoort, Zafarali Ahmed, James Cobon-Kerr, Aishwarya Kamath, Arnar Mar Hrafnkelsson, Le Hou, Ian Mackinnon, Alexandre Frechette, Eric Noland, Xiance Si, Emanuel Taropa, Dong Li, Phil Crone, Anmol Gulati, Sébastien Cevey, Jonas Adler, Ada Ma, David Silver, Simon Tokumine, Richard Powell, Stephan Lee, Michael Chang, Samer Hassan, Diana Mincu, Antoine Yang, Nir Levine, Jenny Brennan, Mingqiu Wang, Sarah Hodkinson, Jeffrey Zhao, Josh Lipschultz, Aedan Pope, Michael B. Chang, Cheng Li, Laurent El Shafey, Michela Paganini, Sholto Douglas, Bernd Bohnet, Fabio Pardo, Seth Odoom, Mihaela Rosca, Cicero Nogueira dos Santos, Kedar Soparkar, Arthur Guez, Tom Hudson, Steven Hansen, Chulayuth Asawaroengchai, Ravi Addanki, Tianhe Yu, Wojciech Stokowiec, Mina Khan, Justin Gilmer, Jaehoon Lee, Carrie Grimes Bostock, Keran Rong, Jonathan Caton, Pedram Pejman, Filip Pavetic, Geoff Brown, Vivek Sharma, Mario Lučić, Rajkumar Samuel, Josip Djolonga, Amol Mandhane, Lars Lowe Sjösund, Elena Buchatskaya, Elspeth White, Natalie Clay, Jiepu Jiang, Hyeontaek Lim, Ross Hemsley, Jane Labanowski, Nicola De Cao, David Steiner, Sayed Hadi Hashemi, Jacob Austin, Anita Gergely, Tim Blyth, Joe Stanton, Kaushik Shivakumar, Aditya Siddhant, Anders Andreassen, Carlos Araya, Nikhil Sethi, Rakesh Shivanna, Steven Hand, Ankur Bapna, Ali Khodaei, Antoine Miech, Garrett Tanzer, Andy Swing, Shantanu Thakoor, Zhufeng Pan, Zachary Nado, Stephanie Winkler, Dian Yu, Mohammad Saleh, Loren Maggiore, Iain Barr, Minh Giang, Thais Kagohara, Ivo Danihelka, Amit Marathe, Vladimir Feinberg, Mohamed Elhawaty, Nimesh Ghelani, Dan Horgan, Helen Miller, Lexi Walker, Richard Tanburn, Mukarram Tariq, Disha Shrivastava, Fei Xia, Chung-Cheng Chiu, Zoe Ashwood, Khuslen Baatarsukh, Sina Samangooei, Fred Alcober, Axel Stjerngren, Paul Komarek, Katerina Tsihlas, Anudhyan Boral, Ramona Comanescu, Jeremy Chen, Ruibo Liu, Dawn Bloxwich, Charlie Chen, Yanhua Sun, Fangxiaoyu Feng, Matthew Mauger, Xerxes Dotiwalla, Vincent Hellendoorn, Michael Sharman, Ivy Zheng, Krishna Haridasan, Gabe Barth-Maron, Craig Swanson, Dominika Rogozińska, Alek Andreev, Paul Kishan Rubenstein, Ruoxin Sang, Dan Hurt, Gamaleldin Elsayed, Renshen Wang, Dave Lacey, Anastasija Ilić, Yao Zhao, Lora Aroyo, Chimezie Iwuanyanwu, Vitaly Nikolaev, Balaji Lakshminarayanan, Sadegh Jazayeri, Raphaël Lopez Kaufman, Mani Varadarajan, Chetan Tekur, Doug Fritz, Misha Khalman, David Reitter, Kingshuk Dasgupta, Shourya Sarcar, Tina Ornduff, Javier Snaider, Fantine Huot, Johnson Jia, Rupert Kemp, Nejc Trdin, Anitha Vijayakumar, Lucy Kim, Christof Angermueller, Li Lao, Tianqi Liu, Haibin Zhang, David Engel, Somer Greene, Anaïs White, Jessica Austin, Lilly Taylor, Shereen Ashraf, Dangyi Liu, Maria Georgaki, Irene Cai, Yana Kulizhskaya, Sonam Goenka, Brennan Saeta, Kiran Vodrahalli, Christian Frank, Dario de Cesare, Brona Robenek, Harry Richardson, Mahmoud Alnahlawi, Christopher Yew, Priya Ponnapalli, Marco Tagliasacchi, Alex Korchemniy, Yelin Kim, Dinghua Li, Bill Rosgen, Zoe Ashwood, Kyle Levin, Jeremy Wiesner, Praseem Banzal, Praveen Srinivasan, Hongkun Yu, Çağlar Ünlü, David Reid, Zora Tung, Daniel Finchelstein, Ravin Kumar, Andre Elisseeff, Jin Huang, Ming Zhang, Rui Zhu, Ricardo Aguilar, Mai Giménez, Jiawei Xia, Olivier Dousse, Willi Gierke, Soheil Hassas Yeganeh, Damion Yates, Komal Jalan, Lu Li, Eri Latorre-Chimoto, Duc Dung Nguyen, Ken Durden, Praveen Kallakuri, Yaxin Liu, Matthew Johnson, Tomy Tsai, Alice Talbert, Jasmine Liu, Alexander Neitz, Chen Elkind, Marco Selvi, Mimi Jasarevic, Livio Baldini Soares, Albert Cui, Pidong Wang, Alek Wenjiao Wang, Xinyu Ye, Krystal Kallarackal, Lucia Loher, Hoi Lam, Josef Broder, Dan Holtmann-Rice, Nina Martin, Bramandia Ramadhana, Daniel Toyama, Mrinal Shukla, Sujoy Basu, Abhi Mohan, Nick Fernando, Noah Fiedel, Kim Paterson, Hui Li, Ankush Garg, Jane Park, DongHyun Choi, Diane Wu, Sankalp Singh, Zhishuai Zhang, Amir Globerson, Lily Yu, John Carpenter, Félix de Chaumont Quitry, Carey Radebaugh, Chu-Cheng Lin, Alex Tudor, Prakash Shroff, Drew Garmon, Dayou Du, Neera Vats, Han Lu, Shariq Iqbal, Alex Yakubovich, Nilesh Tripuraneni, James Manyika, Haroon Qureshi, Nan Hua, Christel Ngani, Maria Abi Raad, Hannah Forbes, Anna Bulanova, Jeff Stanway, Mukund Sundararajan, Victor Ungureanu, Colton Bishop, Yunjie Li, Balaji Venkatraman, Bo Li, Chloe Thornton, Salvatore Scellato, Nishesh Gupta, Yicheng Wang, Ian Tenney, Xihui Wu, Ashish Shenoy, Gabriel Carvajal, Diana Gage Wright, Ben Bariach, Zhuyun Xiao, Peter Hawkins, Sid Dalmia, Clement Farabet, Pedro Valenzuela, Quan Yuan, Chris Welty, Ananth Agarwal, Mia Chen, Wooyeol Kim, Brice Hulse, Nandita Dukkipati, Adam Paszke, Andrew Bolt, Elnaz Davoodi, Kiam Choo, Jennifer Beattie, Jennifer Prendki, Harsha Vashisht, Rebeca Santamaria-Fernandez, Luis C. Cobo, Jarek Wilkiewicz, David Madras, Ali Elqursh, Grant Uy, Kevin Ramirez, Matt Harvey, Tyler Liechty, Heiga Zen, Jeff Seibert, Clara Huiyi Hu, Mohamed Elhawaty, Andrey Khorlin, Maigo Le, Asaf Aharoni, Megan Li, Lily Wang, Sandeep Kumar, Alejandro Lince, Norman Casagrande, Jay Hoover, Dalia El Badawy, David Soergel, Denis Vnukov, Matt Miecnikowski, Jiri Simsa, Anna Koop, Praveen Kumar, Thibault Sellam, Daniel Vlasic, Samira Daruki, Nir Shabat, John Zhang, Guolong Su, Jiageng Zhang, Jeremiah Liu, Yi Sun, Evan Palmer, Alireza Ghaffarkhah, Xi Xiong, Victor Cotruta, Michael Fink, Lucas Dixon, Ashwin Sreevatsa, Adrian Goedeckemeyer, Alek Dimitriev, Mohsen Jafari, Remi Crocker, Nicholas FitzGerald, Aviral Kumar, Sanjay Ghemawat, Ivan Philips, Frederick Liu, Yannie Liang, Rachel Sterneck, Alena Repina, Marcus Wu, Laura Knight, Marin Georgiev, Hyo Lee, Harry Askham, Abhishek Chakladar, Annie Louis, Carl Crous, Hardie Cate, Dessie Petrova, Michael Quinn, Denese Owusu-Afriyie, Achintya Singhal, Nan Wei, Solomon Kim, Damien Vincent, Milad Nasr, Christopher A. Choquette-Choo, Reiko Tojo, Shawn Lu, Diego de Las Casas, Yuchung Cheng, Tolga Bolukbasi, Katherine Lee, Saaber Fatehi, Rajagopal Ananthanarayanan, Miteyan Patel, Charbel Kaed, Jing Li, Jakub Sygnowski, Shreyas Rammohan Belle, Zhe Chen, Jaclyn Konzelmann, Siim Põder, Roopal Garg, Vinod Koverkathu, Adam Brown, Chris Dyer, Rosanne Liu, Azade Nova, Jun Xu, Slav Petrov, Demis Hassabis, Koray Kavukcuoglu, Jeffrey Dean, and Oriol Vinyals. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv:2403.05530 [cs.CL]

[57] Shanchieh Jay Yang Reza Fayyazi, Rozhina Taghdimi. 2024. Advancing TTP Analysis: Harnessing the Power of Encoder-Only and Decoder-Only Language Models with Retrieval Augmented Generation. arXiv:2401.00280v2

[58] Arjun Sarkar. 2022. All you need to know about 'Attention' and 'Transformers' — In-depth Understanding — Part 1. *Towards Data Science* (2022). https://towardsdatascience.com/all-you-need-to-know-about-attention-and-transformers-in-depth-understanding-part-1-552f0b41d021

[59] Saumya. 2024. Training Large Language Models: Delving Deep into Methodologies, Challenges, and Best Practices for Training LLMs. *appypie* (2024). https://www.appypie.com/blog/large-language-models-training

[60] Jessica Schulze. 2024. What is GPT? GPT-3, GPT-4, and More Explained. *Coursera* (2024). https://www.coursera.org/articles/what-is-gpt

[61] Natassha Selvaraj. 2022. 8 Machine Learning Models Explained in 20 Minutes. *DataCamp* (2022). https://www.datacamp.com/blog/machine-learning-models-explained

[62] Deval Shah. 2024. The Ultimate Guide to Deploying Large Language Models Safely and Securely. *Lakera* (2024). https://www.lakera.ai/blog/how-to-deploy-an-llm

[63] Karthik Shiraly. 2023. BART Text Summarization vs. GPT-3 vs. BERT: An In-Depth Comparison. *width.ai* (2023). https://www.width.ai/post/bart-text-summarization

[64] Neeraj Shukla. 2024. Architecture and Components of Large Language Models. *appypie* (2024). https://www.appypie.com/blog/architecture-and-components-of-llms

[65] Xuchen Suo. 2024. Signed-Prompt: A New Approach to Prevent Prompt Injection Attacks Against LLM-Integrated Applications. *ArXiv* abs/2401.07612 (2024). https://api.semanticscholar.org/CorpusID:266999840

[66] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[68] Priyanka Vergadia. 2024. LLM Parameters Explained. *The Cloud Girl* (2024). https://www.thecloudgirl.dev/blog/llm-parameters-explained#:~:text=Conclusion%3A%20Parameter%20count%20in%20an%20LLM%20is%20like,for%20training%20and%20use%2C%20making%20them%20less%20accessible.

[69] Georgia Weston. 2023. What is Large Language Model (LLM)? *101 Blockchains* (2023). https://101blockchains.com/large-language-model-llm/

[70] Chen Yanhui. 2021. A Battle Against Amnesia: A Brief History and Introduction of Recurrent Neural Networks. *Towards Data Science* (2021). https://towardsdatascience.com/a-battle-against-amnesia-a-brief-history-and-introduction-of-recurrent-neural-networks-50496aae6740

[71] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL]

[72] Jun Zhan, Junqi Dai, Jiasheng Ye, Yunhua Zhou, Dong Zhang, Zhigeng Liu, Xin Zhang, Ruibin Yuan, Ge Zhang, Linyang Li, Hang Yan, Jie Fu, Tao Gui, Tianxiang Sun, Yugang Jiang, and Xipeng Qiu. 2024. AnyGPT: Unified Multimodal LLM with Discrete Sequence Modeling. arXiv:2402.12226 [cs.CL]

[73] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. TinyLlama: An Open-Source Small Language Model. arXiv:2401.02385 [cs.CL]