CS447 Literature Review: Large Language Model (LLM)

Shang Lu,
lu94@illinois.edu

May 4, 2025

Abstract

With the emerging trend of large language model, researchers have persued different ways to generalize their models. We observed a shift from N-gram that captures a fixed length of contextual information to transformers that uses attention heads and various embeddings to extract the similarity among tokens (e.g. words) to capture lexical, syntactic, and semantic relationships. In this literature review, I will use attention-oriented transformer structures introduced in ”Attention is All You Need” (Vaswani et al., 2017) as a starting point, and review transformer models that are built on top of it. Each of the Bidirectional Encoder Representations from Transformers, BERT (Devlin et al., 2019) and (Rogers et al., 2020), Transformer-XL (Dai et al., 2019), Capsule Network (Sabour et al., 2017) and Attention-based EM Capsule Network (Zhang et al., 2018) will be discussed and evaluated against criteria that I defined for a generalized language model. Among all the models, BERT and Transformer-XT are showing promising results in their abilities to better handle the syntactic and semantic relationships. But there are still areas like number identification, understanding the negation, and linguistic annotations that these models may lack and could be further improved upon.

1 Introduction

The language model has undergone quite a progression since the introduction of N-gram model. The goal of this literature review to think about the progression of the models, to focus on the tranformer models and attention-based network, to think about their shortcoming if any and to see if there are areas and ways to improve and achieve possibly better model generalization.

The introduction of N-gram language model provides the ability to estimate the next word in the sequence given the previous word probablistic distribution. Unigram language model where n=1 considers each word individually and generate the next word by sampling from the probability distribution of words. Bigram and trigram model look at the previous and previous two words to predict the next. Comparing to the unigram, N-gram with N>1 draws the next word (i.e. predicted word) from the probability distribion conditioned on words before the predicted word (Jurafsky et al., 2025). N-gram language model captures the word context using a fixed window of size N and the context it captures is relatively simple, entirely relying on probability of words built from the frequency counts from a training corpus. Its context is related to word sequences i.e. surface-level patterns but lacks lexical, syntactical and semantical information.

Then Recurrent Neural Networks (RNN) comes in. Rather than using a fixed window size N in the N-gram model, the RNN has hidden layers that are continuously being updated upon receiving a new input token. The token can be a word, a symbol, basically a unit of sentence construct. To ease the understanding, words and tokens will be used interchangeably in this literature review to both mean words. In a simple RNN language model, the hidden layers are used to encode the sequential context of the input words, allowing the model to assign a conditional probability to a word based on all proceeding words (Jurafsky et al., 2025). In another word, when compared to N-gram language model, the range of the surface-level patterns is extended to the entire vocabulary, eliminating the dependency of a fixed window size. The change in the range of surface-level pattern becomes more apparent from the fact that the model has been used as a zipping algorithm (Goyal et al., 2018) where the word matching pattern spans the whole vocabulary.

It is worth noting that there are RNN variants with added complexity (Jurafsky et al., 2025) where gating mechanisms are added. The gating mechanisms are essentially trainable neural layers with sigmoid activation function that result in the vector of Bernoulli probability. Simply put these added gating mechanism, via training, determine how much of the hidden layers to keep, that is, how much matching pattern to retain. It models lexical pattern in that it recognizes how words appear and co-occur in paragraphs. From the Gated Recurrent Units (GRUs) variant to Long Short-Term Memory Networks (LSTMs) the level of expressiveness increases, getting better at learning the long-range word dependencies.

A trend can be thus observed where as we move along the timeline of language models, newer techniques make the model capture more expressive properties. These language models use these properties trying to realize better generalization. The properties captured have seen a shift from word sequences and arrangements to lexical relations, and as we will see shortly, to syntactic and sematic relationships as well.

The emergence of transformer models have resulted in more of the syntactic and semantic relations captured (Vaswani et al., 2017). These transformer models have embedded self-attention mechanism. In this review we will see how these added self-attention mechanisms provide a door to greater language model generalization and explore in detail about their shortcoming. I will discuss possible ways to achieve greater model generalization and I will go over some of the relatively recent transformer models and their analysis. I will go over the standard transformer model (Vaswani et al., 2017) and its prominent variant BERT (Devlin et al., 2019). I will also review the paper that reviews the BERT model (Rogers et al., 2020) and a paper that introduces a transformer variant, namely Transformer-XL (Dai et al., 2019). At last I will review paper on the Capsule Neural Networks (Sabour et al., 2017) and (Zhang et al., 2018) and discuss how these Convolutional Neural Network (CNN) based paper would provide additional insight into a more generalized transformer model.

2 Background

Before diving into the transformers paper review, it is worth taking a brief moment to loosely define the term language model generalization and describe what I see being a generalized model looks like.

A general language model should be able to understand and extract the lexical, syntactic, and semantic relationships among words and sentences, and should be able to use such relationship to generate language i.e. words and symbols across a wide variety of linguistic domains and tasks, not limited to a specific use case.

Taking us human as an example, I believe humans are intrinsically language models exhibiting a high-degree of generalization.

First, there is no limit on the type of languages we can learn. When we are born, we are able to learn our native language from surroundings. When we grow up we are exposed with a plethora of languages. Despite of not being able to understand every single one of them, we can still differentiate them from our native language and we are able to learn some of the new ones through doing filling in the blank homework (unsupervised learning) and interation with the language teacher (supervised learning). One can further argue that the number of languages learned could help speed up the learning process for other languages, especially if that language to be learned is historically related to the learned language in some ways. I will not dive into any paper proving this point as this is not the main point of this literature review. In a way we can see this as training and building up syntactic and semantic tree in our brain.

Second, we have strong ability to distinguish the lexical difference between numbers and words for their linguistic role and representation. I believe this is through contextual understanding. Words are treated as part of speech (nouns, verbs, noun phases, and etc) and numbers can be both numerals and treated also as part of speech when used as determiners, nouns, noun phases and etc. Given a math equation with numbers we are able to instantly treat it as a math question and apply computation on it without incurring any part of speech relationship on it. And when the number appears in a sentence, for instance, as a Post Code, we can recognize its placement in the part of speech as treat it as such.

Third, we maintain a fixed set of ground truth. This includes facts, theories, laws, and hypotheses. We take facts as things are observed to be true or tested and shown to be true. Most of the time, with facts, once we learn it we do not offer much explanation and take it for granted. Theories and laws can be similar, and they are things that have been rigorously tested and are shown to have a high predictive power. Hypotheses are basically educated guess without rigourous testing backing it up and it sometimes comes up after observing things that trigger a thought process. For facts, we do not need to learn it the second time. Once we know about it and we can use it as a ground truth and derive other things. For theories and laws, we still need continous learning on them, but most of the time i.e. 99% of the chance, we treat it like fact. For hypotheses, these are basically inferences based on the prior knowledge.

Lastly, chains of thought are a trait we possess. Simply put, given $A \rightarrow B$, $B \rightarrow C$, we can infer $A \rightarrow C$. This can be seen as building a logic tree with edging connected the nodes being the reasoning.

These four aforementioned aspects that I believe constitute a more generalized langugage model will be used throughout this literature review. These traits will be also used in the discussion section to evaluate models from individual paper and help answer the overarching question.

3 Transformers

The paper "Attention is all you need" (Vaswani et al., 2017) introduces the transformer language model. The draft, "Speed and Language Processing" (Jurafsky et al., 2025) from Stanford has also detailed its mechanism. This section will review and discuss this transformer architecture.

The underlying idea behind the transformer is to, simply put, generate a contextual embedding for each word at each position. As shown in Figure 1, given a sample sentence, "The chicken didn't cross the road because it was too tired", a contextual embedding is generated for the word "it" at Layer k+1 at position 7 with most of its attention weight associated with Layer k word position 0, 1, and 5 (Jurafsky et al., 2025).

We can observe from this simplied example in Figure 1 that the computation of the word "it" draws most heavily on the representation for "The chicken", and "road". The self-attention weight distribution α is steered towards "The chicken" and "road" and "it" will end up coreferring with either "The chicken" or "road".

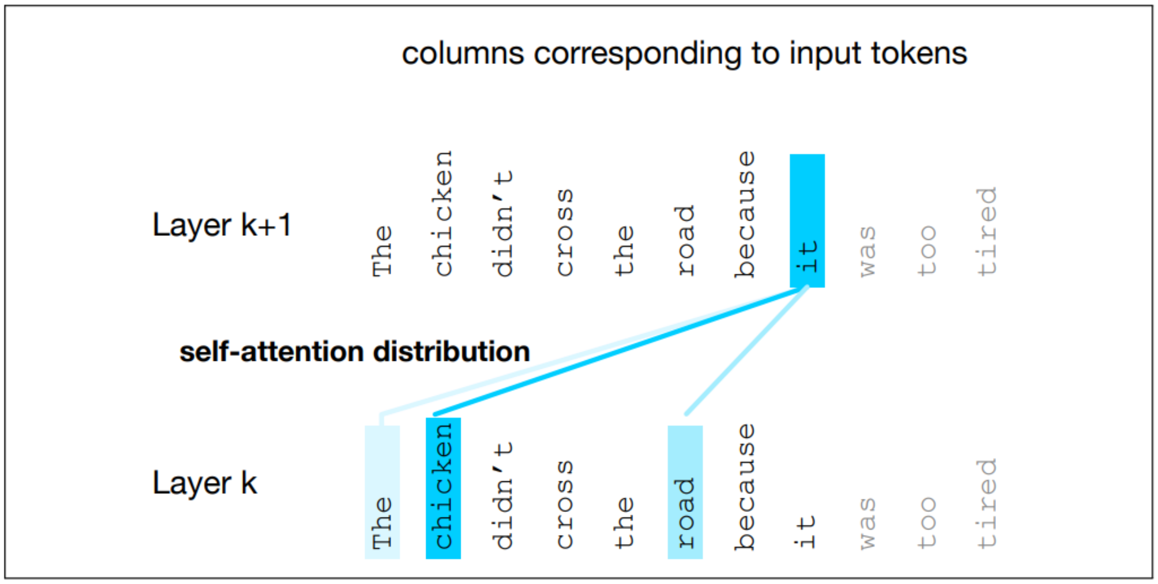


Figure 1: A sample self-attention weight distribution α at Layer k+1 for the word "it". The darker the color the more attention weight a word at Layer k contributes to the word "it". (Jurafsky et al., 2025)

This self-attention (weight) layer is more formally defined as $a_i = \sum_{j \le i} \alpha_{ij} x_j$ as shown in Figure 2 (Jurafsky et al., 2025). The input in Figure 1 are now x_i and the output of the self-attention layer has the same length as the input. α_{ij} is essentially a similarity score calculated by the dot product of x_i and x_j, then normalized with a softmax function. a_3 for instance, is computed by softmax$[x_3 \cdot x_1, x_3 \cdot x_2, x_3 \cdot x_3]$.

The paper (Vaswani et al., 2017) formally defines the attention head in terms of Q, K, and V. Assuming there are N input tokens and each token has gone through token embedding and has embedding size d, the input tokens can be represented as a matrix X of size $[N \times d]$. Q would

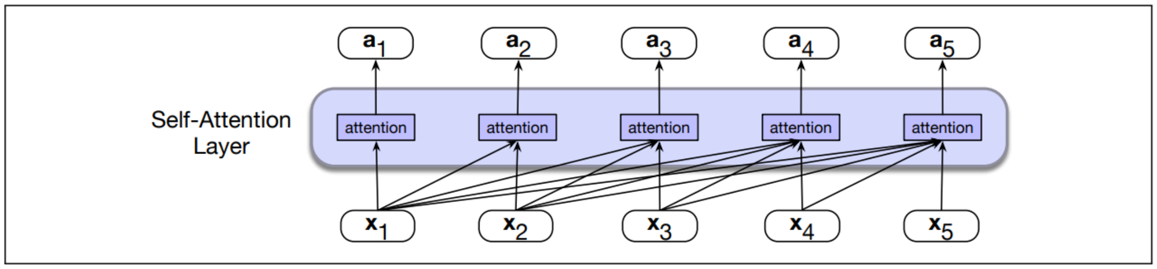


Figure 2: A more formally defined self-attention layer (Jurafsky et al., 2025)

have shape $[N \times d_k]$, K would have shape $[N \times d_k]$, and V would have shape $[N \times d_v]$. The vectors Q, K, and V are constructed by XW^Q, XW^K, and XW^V where W^Q, W^K, and W^V are all trainable attention weight matrices of shape $[d \times d_k]$, $[d \times d_k]$, and $[d \times d_v]$ respectively. d_k and d_v have a similar concept as to the dimension of hidden layer where the embedding size are transformed into dimension d_k and d_v via weight matrices.

If we break down Q, K, and V into $q_i = x_i W_Q$, $k_i = x_i W_K$, and $v_i = x_i W_V$.
q_i means query representing the current element (being compared to).
k_i means key representing the preceding input element (being comapred with).
v_i means value, representing the preceding element that gets weighted (whose dimension has undergone mapping from embedding size d to d_v).

These matrices together generates the output of self-attention and an illustration of self-attention calculation for a_3 with token size N=3 and input token X_1, X_2, and X_3 is shown in Figure 3. The output a_3 of the self-attention contains the contextual embedding of X_1, X_2, and X_3 with different attention weights.

If we scale this up, the attention head is computed by head = softmax($\frac{QK_T}{\sqrt{d_k}}$)V. The QK_T matrix would then be a NxN matrix as shown in Figure 4. The paper (Vaswani et al., 2017) introduces a masking function masking out the future keys for the query. And as I will review later, this is the key difference between BERT language model and this vanilla transformer model. And there are implications in how much more the contextual information would be retained and the way training would be performed.

Overall this paper (Vaswani et al., 2017) provides a foundation and also serves as a building block for later transformer architectures. This is crucial in understanding the transformer architectures that I will review in the next a few sections as those transformer models (Devlin et al., 2019) and (Dai et al., 2019) build directly on top of this. The introduction of the transformer is also cruicial in a sense that we can now extract the contextual embedding of input tokens by observing at the output of the attention heads. The presence of the attention heads with mask layer allow each token to maintain a "half-sentence" i.e. up to the token position scope which in turn removes the context window size that was the main limitation of N-gram model. The attention heads also enable word at the irregular position to associate with each other via attention weight, which we can think of it as to start capturing lexical and semantic relationships.

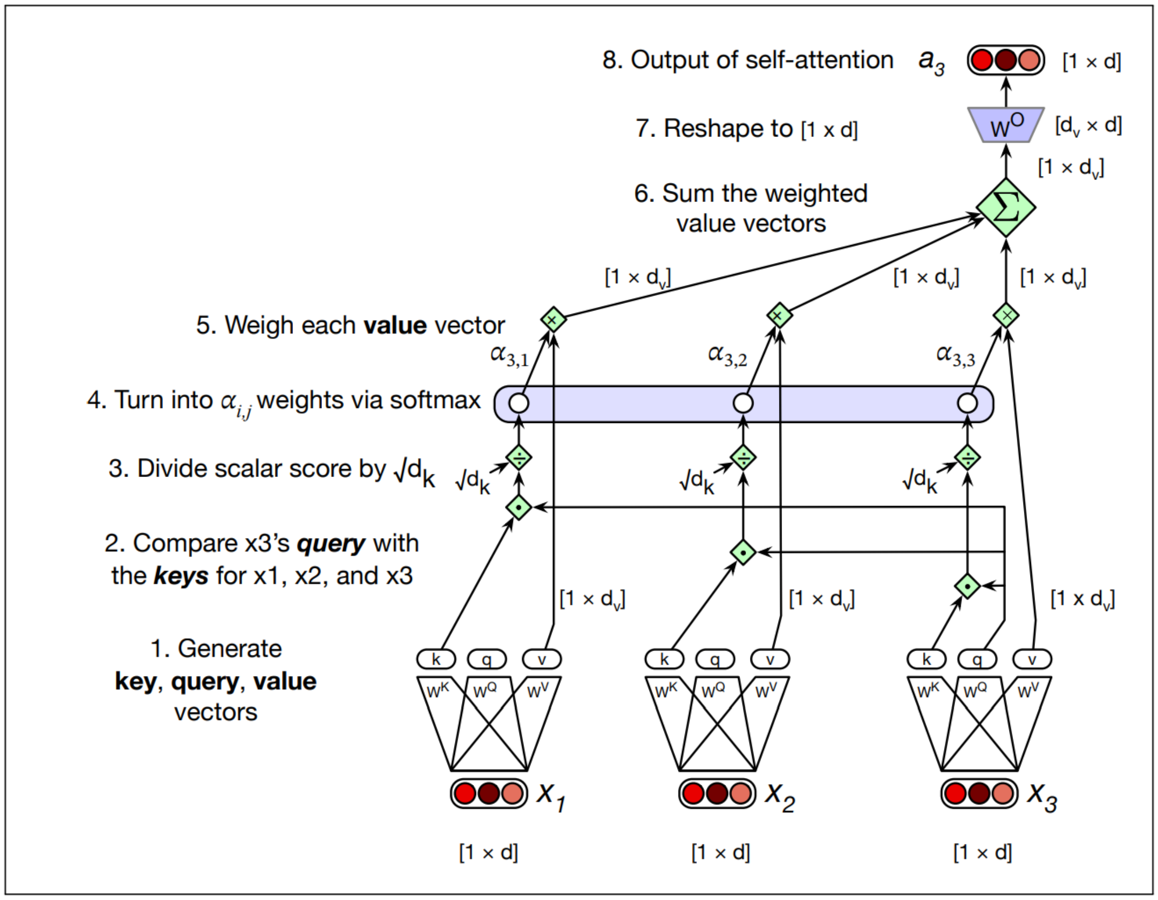

8. Output of self-attention
a_3
[1 × d]
7. Reshape to [1 x d]
W^O
[d_v × d]
[1 × d_v]
6. Sum the weighted value vectors
[1 × d_v]
[1 × d_v]
[1 × d_v]
5. Weigh each **value** vector
$\alpha_{3,1}$
$\alpha_{3,2}$
$\alpha_{3,3}$
4. Turn into $\alpha_{i,j}$ weights via softmax
3. Divide scalar score by $\sqrt{d_k}$
$\sqrt{d_k}$
2. Compare x3's ***query*** with the ***keys*** for x1, x2, and x3
[1 × d_v]
1. Generate **key**, **query**, **value** vectors
k
q
v
W^K
W^Q
W^V
x_1
x_2
x_3
[1 × d]

Figure 3: An illustration of self-attention calculation for a_3 with three input tokens X_1, X_2, and X_3 with q_i, k_i, and v_i weight matrices (Jurafsky et al., 2025)

4 BERT

The idea behind BERT architecture is straight-forward: Instead of having a mask layer that masks out the future keys for the query. As seen in Figure 4, the model removes the mask and captures all the attention for a given token. This means an output of the attention head now contains the contextual embedding of all other tokens (both before and after) (Devlin et al., 2019). An example implementation is illustrated in Figure 5.

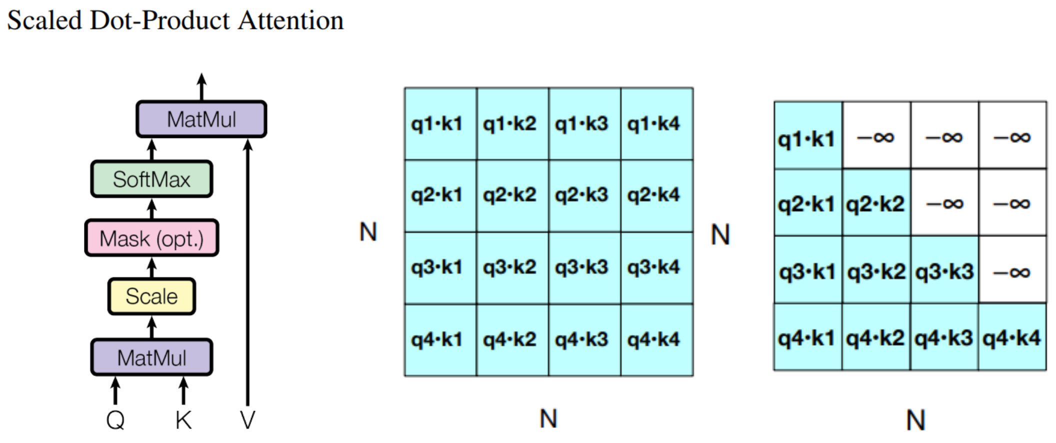


Figure 4: The masking of attention head in the transformer (Vaswani et al., 2017) and (Jurafsky et al., 2025)

The main difference is in the training. Because the removal of the mask breaks causality in the self-attention layer, it makes little sense to train it by iteratively predict the next token - the answer would simply be a replay of the trained model. Therefore the main theme of BERT is to introduce new training methodologies, namely, dividing the language model training into intput manipulation, pre-training and fune-tuning.

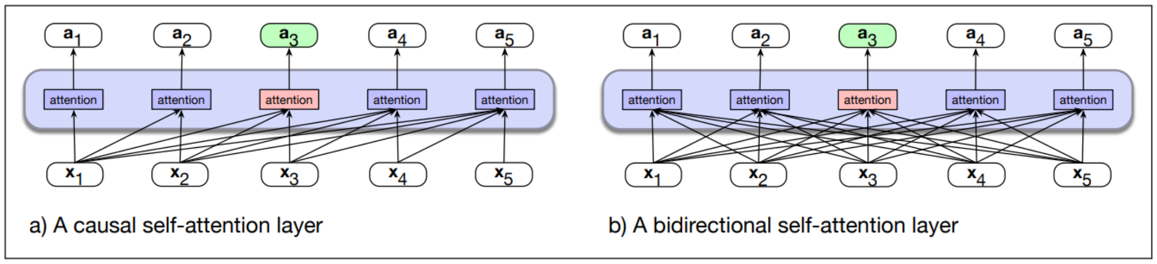

a₁ a₂ a₃ a₄ a₅
attention
x₁ x₂ x₃ x₄ x₅
a) A causal self-attention layer
b) A bidirectional self-attention layer

Figure 5: A BERT implementation vs. a causal self-attention tranformer implementation (Jurafsky et al., 2025)

4.1 Input Representation Manipulation

The input representation is adjusted in one of the following formats (Devlin et al., 2019):

1. A single sentence or two sentences packed together

2. A special token [CLS] is added to the beginning of every sequence.

3. A special token [SEP] is added to the beginning of each sentence except for the beginning of sequence as sentence separator

4. A learnable embedding is added to the indicate which part of the sentence i.e. sentence A or B, the token belongs to.

Figure 6 illustrates how this is done with an input sequence containing two sample sentences, ”my dog is cute” followed by ”he likes play##ing”.

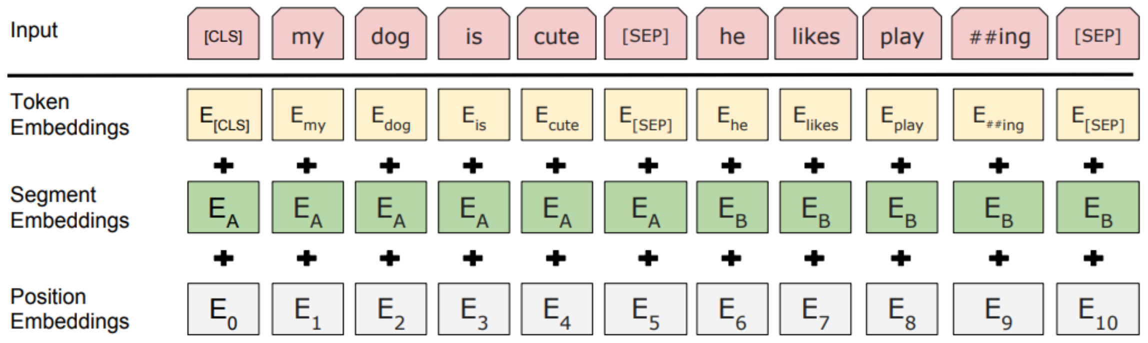


Figure 6: BERT input representation (Devlin et al., 2019)

4.2 Unsupervised Pre-training

Pre-training uses two unsupervised tasks: a ”masked language model” (MLM) and Next Sentence Prediction (NSP) (Devlin et al., 2019).

The Masked LM task essentially chooses 15% of the token positions in the input sentence at random. When the i-th token is chosen, 80% chance it is replaced with [MASK], 10% of the time it is replaced with other random token, and for the remaining 10% of the time it is unchanged. Let T_i represent this ”masked” or randomized token. T_i will be used to predict the original token and trained with cross entropy loss (Devlin et al., 2019). This is illustrated in the pre-training Figure 7.

The NSP on the other hand, trains from any monolingual corpurs where it chooses sentences A and B where 50% of the time B is the actual next sentence that follows A, while the other 50% of the time the sentence B is randomly selected (Devlin et al., 2019).

4.3 Supervised Fine-tuning

Fine-tuning is straight-forward as task-specific inputs and outputs can be plugged into BERT and fine-tune all the parameters end-to-end (Devlin et al., 2019). At the output, the special token [CLS] can be used to locate the beginning of the output can be fed into the output layer for classification. This is illustrated in the fine-tuning Figure 7.

Overall BERT, through a simple implementation change and training methodologies change, has addressed one downside of the causal attention head in the vanilla transformer model (Vaswani et al., 2017). The removal of the mask layer allows each token to maintain a ”full-sentence” i.e. global

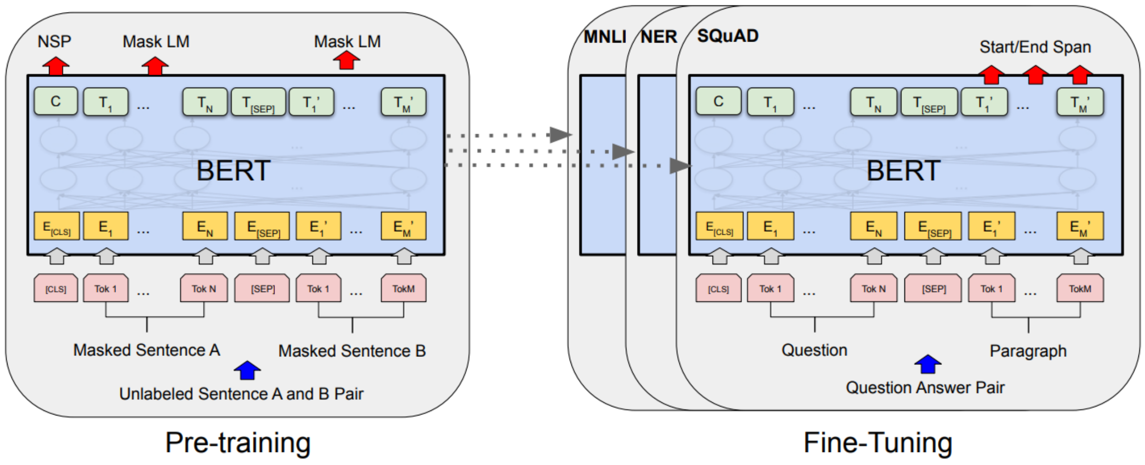


Figure 7: BERT unsupervised pre-training and supervised fine-tuning (Devlin et al., 2019)

scope which in turn makes the token fully context-aware. The addition of unsupervised MLM and NSP, and supervised training captures a certain level of syntactic and semanitc relationships among tokens. This will evident when we review (Rogers et al., 2020) next.

5 How BERT Works

The paper ”A Primer in BERTology: What We Know About How BERT Workds” by (Rogers et al., 2020) surveyed more than 150 studies of the popular BERT model and reviewed the how BERT works. The paper (Rogers et al., 2020) started with giving an overview of the BERT architecture which I have reviewed previously. As a recap, compared to the transformers (Vaswani et al., 2017), BERT applies additional embedding layers to the the input tokens and these additional layers are token and segment embeddings. Special tokens such as [CLS] and [SEP] are used to denote the beginning of the input sequence and to delimit the input segments. The paper (Rogers et al., 2020) explained how BERT works from the following perspective.

5.1 Syntactic, Semantic, World, and Localizing Linguistic Knowledge

The paper (Rogers et al., 2020) referenced many other papers in explaining how BERT extracted or is related to these knowledge.

The paper (Rogers et al., 2020) pointed out that BERT uses a syntactic-tree-like structure to hold information about words. It also showed that BERT embedding parts of speech information, and syntactic roles. It suggested the syntactic structure is not directly encoded in the self-attention weight as there are no successful attempts to extract the full parse tree from BERT heads. It also suggested the syntactic information can be recovered from BERT token representation by quoting a recent research paper by (Wu et al., 2020) with data shown in Figure 8. It concluded that BERT ”naturally” learns some syntactic information but does not necessarily perform part-of-speech tagging and other linguistic annotations and BERT is not well behaved on malformed input and the concept of negation.

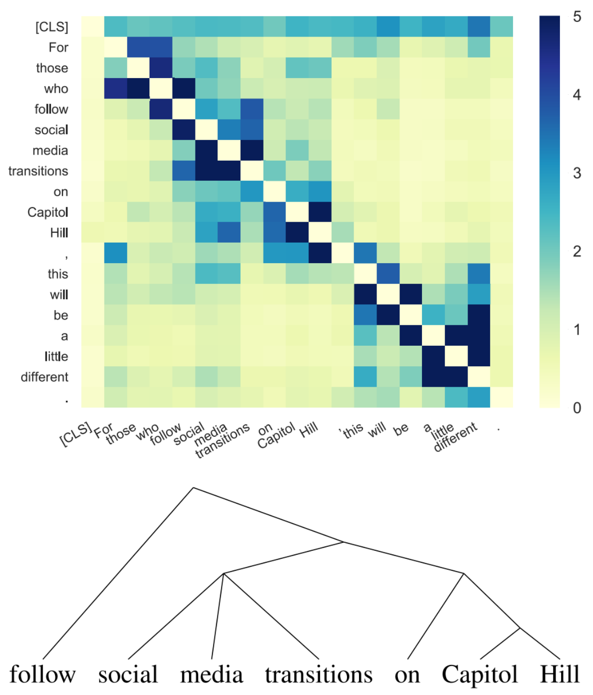


Figure 8: A parameter-free approach to represent token representations as a syntact tree, based on measuring the impact that one word has on predicting another word within a sequence in MLM task (Wu et al., 2020).

The paper (Rogers et al., 2020) pointed out BERT has some knowledge of semantic roles in that it prefers e.g., ”to tip a chef” compared to ”to tip a robin” and prefers e.g., ”to tip a waiter” over ”to tip a chef”. In addition to semantic roles, it further showed BERT encodes information about entity types and relations, however sruggles with number representation, e.g. floating point numbers. It also showed that BERT is susceptible to name entity changes and suggested entity knowledge is not fully absorbed during unsupervised pre-training.

In terms of world knowledge, despite being competitive for some relation types, BERT was shown to struggle with pragmatic inference and cannot reason based on its world knowledge. BERT can guess based on some of the properties on its world knowledge but cannot reason about the relationships between those properties. The paper quoted from one of the other papers that ”some of the BERT’s world knowledge success comes from learning stereotypical associations” and gave an example of BERT predicting an Italian-sounding name to be Italian, even when it is incorrect.

The paper (Rogers et al., 2020) further reported that the output of BERT transformer layer contains contextual embeddings exhibiting better lexical encoding as these contextual embeddings are better at word similarity task and word sense disambiguation task. But the word sense captured in BERT’s contextual embedding depends on the word’s position in the sentence.

5.2 How Attention Heads Retain Syntactic Relation and BERT Layers

The paper (Rogers et al., 2020) showed that most self-attention heads do not directly encode any non-trivial linguistic information.

Viewing the attention patterns similar to that in Figure 8, it references (Kovaleva et al., 2019) that the "heterogeneous" attention pattern shown in Figure 9 could potentially be interpreted linguistically. And some BERT attention heads seem to specialize in certain areas of syntactic relations.

It (Rogers et al., 2020) showed that no single attention head has the complete syntactic tree information. And despite of attention weights showing weak indications of subject-verb agreement, stacking BERT attention layers does seem to help hold more syntactic information. It showed that the lower layers of BERT have most information about linear word order, concerning mostly with word index i.e. word positioning in the sentence. The middle layers of BERT hold the largest amount of syntact information and the find layers hold the most task-specific information. And there is a point of limited return on the stacking.

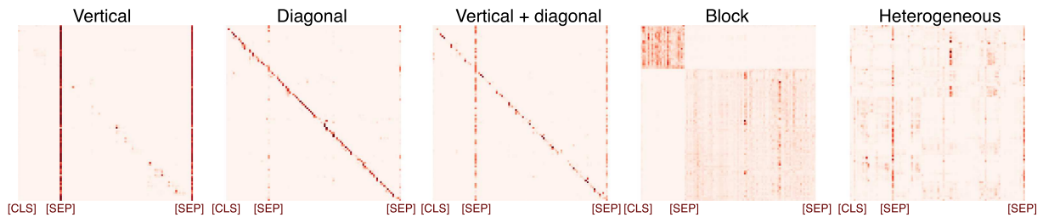


Figure 9: Various BERT attention heads pattern (Kovaleva et al., 2019)

5.3 Training BERT and Addressing BERT Parameter Size

The paper (Rogers et al., 2020) also touched upon possible optimization in training the BERT and it investigated BERT parameter sizing and showed ways to compress it.

Experiments (Rogers et al., 2020) have shown the number of layers are more significant than the number of heads in optimizing the training and performance of the BERT architecture. The paper (Rogers et al., 2020) showed there is a benefit to use large batch size in training and lower layers can be trained first and then the trained parameters can be copied over to deeper layers to provide a "warm-start". It quoted a 25% speed up in training without sacrificing performance. The paper (Rogers et al., 2020) also showed that, during the pre-training of BERT, replacing [MASK] with [UNK] token helped the model learn a representation of unknown. Masking can be extended to phrases and named entities instead of a single token and deletion of token spans can be used instead of infilling. It also referenced (Bao et al., 2020), and showed that replacing the conventional mask with a pseudo-mask allowed combining autoencoding and partially autoregressive tranining together. It then moved on and mentioned removing NSP does not hurt or slightly improves performance.

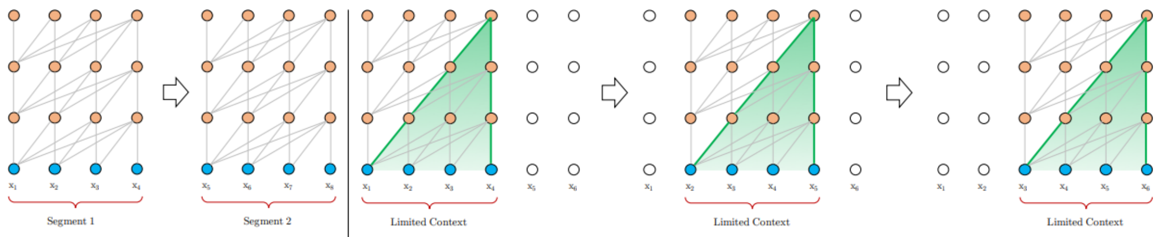
To account for the BERT parameter size, the paper (Rogers et al., 2020) gathered data showing the growth of model parameters is a raising concern and needs to be addressed. (e.g., Turing-NLG by Microsoft, 2020, with 17B parameters and GPT-3, 2020 with 175B parameters). It referenced other paper to show that all but a few transformer heads could be pruned without significant losses in performance and most BERT heads in the same layer show similar self-attention patterns. It briefly mentioned two techniques in reducing the BERT overparametrization: one through quantization i.e. reducing the precision of attention weights and another by prunning, essentially zero-out region of attention matrices and making them into sparse matrices.

Overall the paper (Rogers et al., 2020) provided a comprehensive view of how BERT works by going over how it retains syntactic, semantic, and world knowledge and how effective are the attention heads and BERT layers. These are key data that will later back up our discussions about model generalization.

6 A Transformer Variant: Transformer-XL, a Model to Resolve Some of the Transformers' Shortcoming

The paper "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context" by (Dai et al., 2019) aimed to address a practical shortcoming of the transformers, that is, loss of context across segments.

The shortcoming of the transformer model comes into play when segmentation is performed due to batching in training where a long input sequence needs to be broken into segments and trained separately. The contextual embedding will be broken across the segment boundary because one attention head at all levels in one segment would not be able to attend any of the heads in other segments. There is another shortcoming with the existing model where all the hidden states i.e. attention heads need to be re-computed when generating the next token. This is inefficient. Both of these shortcomings are illustrated in the following Figure 10 (Dai et al., 2019).



(a) Train phase.

(b) Evaluation phase.

Figure 10: Illustration of the contextual boundary due to segmentation in training where attention heads are physically isolated across segments and the inefficient next-token generation in the evaluation phase where the hidden states are computed from scratch (Dai et al., 2019)

In attempts to resolve these two shortcomings, the paper (Dai et al., 2019) used the following techniques:

1. Rework the Q (query), K (key), and V (value) matrices generation so it uses the concatenated hidden attention layer in both previous and current segments.
2. A new position encoding that encodes relative position information of a token instead of the absolute position

The rework on the Q, K, V are formally defined in the paper (Dai et al., 2019) as follows.

Let two consecutive segments of length L be defined as: $s_\tau = [x_{\tau,1}, \ldots, x_{\tau,L}]$, $\quad s_{\tau+1} = [x_{\tau+1,1}, \ldots, x_{\tau+1,L}]$. And let the n-th layer hidden state for segment s_τ be denoted:$h_\tau^n \in \mathbb{R}^{L\times d}$ where d is the dimension of hidden layer. For the next segment $s_{\tau+1}$, the concatenation of the previous segment's hidden (attention head) states and the current hidden states is computed by $\hat{h}_{\tau+1}^{n-1} = [\mathrm{SG}(h_\tau^{n-1}) \cdot h_{\tau+1}^{n-1}]$. The $\mathrm{SG}(\cdot)$ denotes the *stop-gradient* operation, preventing backpropagation through past hidden states. The concatentation happens along the L dimension and the Q, K, V calculations are then updated to use the concatenated hidden states where $q_{\tau+1}^n = h_{\tau+1}^{n-1} W_q^\top$, $k_{\tau+1}^n = \hat{h}_{\tau+1}^{n-1} W_k^\top$, and $v_{\tau+1}^n = \hat{h}_{\tau+1}^{n-1} W_v^\top$. The updated hidden attention head states for the next segment then becomes $h_{\tau+1}^n = \mathrm{TransformerLayer}(q_{\tau+1}^n, k_{\tau+1}^n, v_{\tau+1}^n)$ (Dai et al., 2019).

As a result, as illustrated in Figure 11 the updated model exhibits a RNN like behavior where the previous contextual embedding is carried forward and to combine with the hidden states in the next segment.

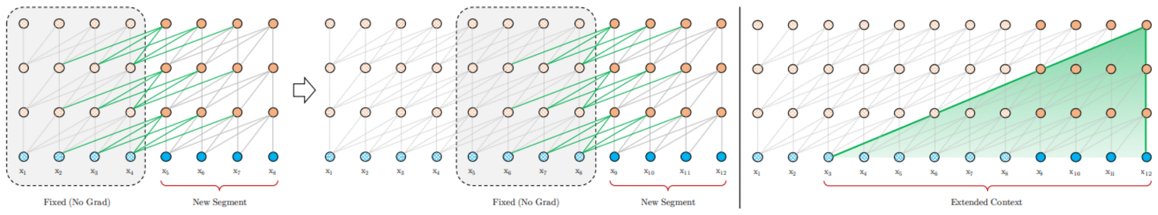


Figure 11: Illustration of updated transformer model (Transformer-XT) with updated hidden states (Dai et al., 2019)

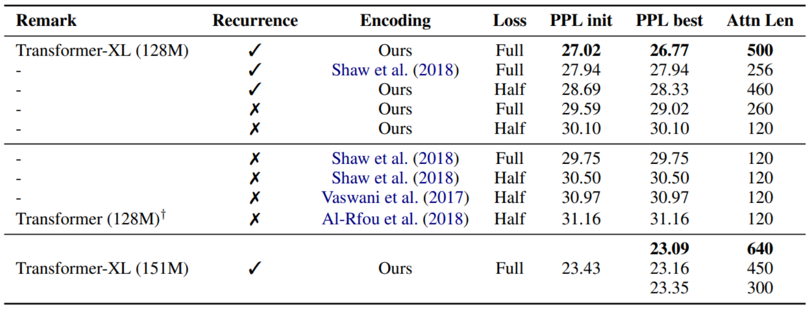
A new position encoding on the other hand, is needed because using absolute position embedding will mess up the attention score calculation between Q (query, the current token) and K (key, all other previous tokens). The problem happens when one of the current hidden state (attention head) tries to attend any one in the previous segment. The absolute position will make it as if the key is in the current segment even it is not.

The paper (Dai et al., 2019) used the relative position in the updated embedding where the attention score is updated to:

$$A_{i,j}^{\mathrm{rel}} = \underbrace{(E^\top x_i W_q W_k^\top E x_j)}_{\text{(a) content-to-content}} \underbrace{(E^\top x_i W_q W_k^\top R_{i-j})}_{\text{(b) content-to-position}} \underbrace{(u^\top W_k^\top E x_j)}_{\text{(c) global content bias}} \underbrace{(v^\top W_k^\top R_{i-j})}_{\text{(d) global position bias}}$$

It is worth noting that R_{i-j} is a sinusoid encoding matrix and u^T, and v^T are trainable parameters. The query vector is the same for all query positions as the attentive bias towards different words in "memory" i.e. the previous segment remains the same.

The result illustrated in Figure 12 shows these two techniques have a positive impact on the model performance, inferring a longer context embedding has been achieved.



Remark	Recurrence	Encoding	Loss	PPL init	PPL best	Attn Len
Transformer-XL (128M)	✓	Ours	Full	**27.02**	**26.77**	**500**
-	✓	Shaw et al. (2018)	Full	27.94	27.94	256
-	✓	Ours	Half	28.69	28.33	460
-	✗	Ours	Full	29.59	29.02	260
-	✗	Ours	Half	30.10	30.10	120
-	✗	Shaw et al. (2018)	Full	29.75	29.75	120
-	✗	Shaw et al. (2018)	Half	30.50	30.50	120
-	✗	Vaswani et al. (2017)	Half	30.97	30.97	120
Transformer (128M)†	✗	Al-Rfou et al. (2018)	Half	31.16	31.16	120
					23.09	**640**
Transformer-XL (151M)	✓	Ours	Full	23.43	23.16	450
					23.35	300

Figure 12: Transformer-XL performance result with updated hidden states concatenation and relative position encoding (Dai et al., 2019)

7 Capsule Network and Attension-Based Capsule Network

I will review these two papers (Sabour et al., 2017) and (Zhang et al., 2018) together as they share a very similar goal. The former aimed to extract properties of an object and then construct part-whole relationship among these properties. Individual parts are aggregated to a form a whole if these parts share similar properties. The latter aimed at relation extraction specifically multiple entity pairs relation extraction.

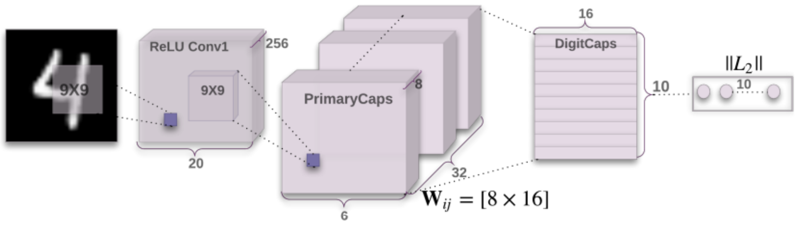


Figure 13: A sample Capsule Network architecture (Sabour et al., 2017)

As illustrated in the following Figure 13, the Capsule Network take a digit as an input image. It goes to a 20x256 ReLU Convolutional layer that generate the feature map. Then, the primary capsules at the second layer extract lower-level features from this 20x256 feature map. One primary capsule is essentially 8 convolutional units stacked and each with a 9x9 kernel and stride of 2. The primary capsule in another word, are full of low-level feature extracted from the feature map. The output of each of these 8D stacked vector from the primary capsules represents the presence of a feature and its properties. This is in a way similar but not quite the same as word embeddings. Now that we have 32 channels of convolutional 8D capsules in a 6x6 grid, simply put, we have 6x6x32 primary capsules and each primary capsule outputs a 8D vector. Each primary capsule within a 6x6 grid shares weight with other capsules in the grid. Each of these capsules sends to the digital capsules above to each of the consecutive capsules. These primary capsules and digital capsules are connected by dynamic routing algorithm. The digital capsules in the layer above has a dimension of # of class instances x 16 (16D per digital capsule). Each 16D vector is associated with an instance of class, each vector can be interpreted as high-level features that represent the class instance and are made by low-leve features below. In another word, the length of the each of the 16D vector indicates presence of an class instance.

Overall, the Capsule Network constructs feature maps, from which it uses primary capsules to gather lower-level features and use digital capsules to ”combine” low-level features into high-level features and match against the expected feature sets that an object should have. This essentially demonstrates its ability to capture the part-whole relationship.

For the paper (Zhang et al., 2018) I will briefly mention it as a complementary to the Capsule Network. The model (Zhang et al., 2018) is illustrated in Figure 14.

The primary capsule layer and dynamic routing are added after the bidirectional LSTM layer. Then it uses parent capsules capsules to capture the number of relations. Using dynamic routing, it iteratively amends the connection strength between the child and parent capsules to strength the represented relations.

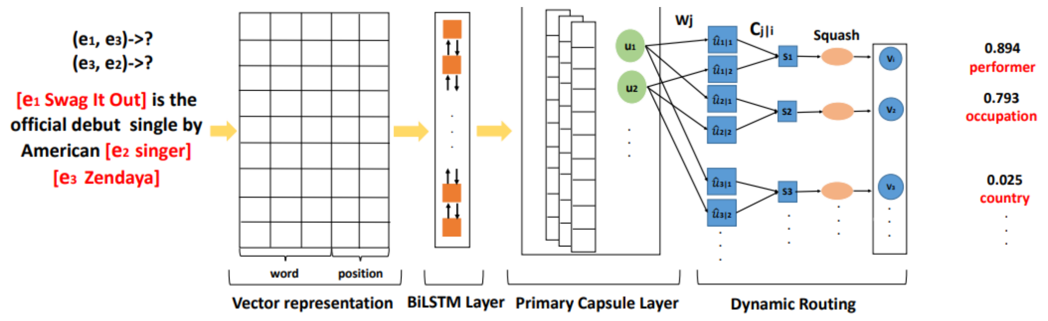


Figure 14: A sample Capsule Network architecture with attention mechanism (Zhang et al., 2018)
.

8 Discussion

With all the paper reviewed it is time to return to the overarching question: Given the progression of the transformers and attention-based networks, can we further improve the generlization of language model? The answer is a resounding yes.

If we use the four points I described in the Background Section 2 as a rubric to evaluate all the aforementioned models we can see these models do not satisfies all of the criteria.

First, transformers, both the vanilla in Section 3 and BERT variant reviewed in Section 4 and in Section 5 can be said to possess borderline syntactic and semantic information. Despite of these models seemingly learn ”naturally” about syntactic and semantic information, they seem to lack some key ”components” that could enable them to tag part-of-speech and retain linguistic annotations. For instance, as concluded in Section 5 BERT is not well behaved on malformed input and does not seem to have the concept of negation. It is worth thinking that these missing key ”compoents” could be due to context segmentation introduced in batch training. Because of this, the Transformer-XT reviewed in Section 11 might have improved the ability to extract and retain syntactic and semantic information through hidden states carry-forwarding and merging. I personally do believe the direction of training with MLM, NSP and carrying forward previous hidden states are on the right path and here to stay. This is because I can relate to this as human in day-to-day life where we learn language, grammars by filling in the blanks, by doing reading comprehension that requires answering contextualized questions, and by reciting vocabularies and phrases.

Second, transformer models that are reviewed in Section 3, Section 4 are said to have a hard time distinguishing between numbers and words. The Transformer-XT model reviewed in Section 6 improved on the contextual understanding and but did not explicitly state about its number-identifying ability. Numbers have irregular intervals in their positions in a sequence and they can have distant meanings depending on the context. For instance, math equations would have numbers close to each other. This is a trait and if being identified it would require no further inference as the computation will be done right after. Numbers can also be the name of a specific alcoholic drink and can be a Post Code and can be a year or date. Identification of the numbers and perform the expected action would be a major improvement to achieve greater model generalization. One thought is to use techniques similar to capsule network for word trait extraction then train with batches of ground truth where once the weight are properly trained, only an extremely slight change they would be updated.

This idea can be combined the point I am trying to make next where I believe we need to maintain a set of ground truth with non-updatable weight in the model and providing an offload mechanism. The offload mechanism behaves like us humans, where encountering things that requires computation, halt inference and offload the action to a floating-pointer compute unit to do the computation.

Lastly, chains of thought are a trait we possess but are not covered in any of the reviews I have in this literature review. None of the reviewed models cover this. However there are recent papers on developing chain of thoughts by command prompts and that is interesting and could be a future topic on its own.

9 Conclusion

In conclusion, as I have covered in review Section 3, Section 4, Section 5, Section 6, and Section 7, modern large language models have been experiencing a steady and yet fast pace of advancement. There are areas where these recent architectures are still lacking making them not as a general language model as humans do. But these shortcomings are being identified, and some of them have been addressed. The gap between the current and a truly generalized language model is shrinking. I do sincerely hope this literature review could help the reader gain a clearer understanding in recent advancement in language models, and provide a fresh view on some of the road blocks in front.

References

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019)*, pages 4171–4186, Minneapolis, Minnesota, June 2–7, 2019. Association for Computational Linguistics. https://aclanthology.org/N19-1423.pdf.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020. https://aclanthology.org/2020.tacl-1.54.pdf.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019)*, pages 2978–2988, Florence, Italy, July 28 – August 2, 2019. Association for Computational Linguistics. https://aclanthology.org/P19-1285.pdf.

Sara Sabour, Geoffrey E. Hinton, and Nicholas Frosst. 2017. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017. https://arxiv.org/pdf/1710.09829.

Ningyu Zhang, Shumin Deng, Zhanlin Sun, Xi Chen, Wei Zhang, and Huajun Chen. 2018. Attention-based capsule networks with dynamic routing for relation extraction. *arXiv preprint arXiv:1812.11321*, 2018. https://arxiv.org/pdf/1812.11321.

Daniel Jurafsky and James H. Martin. 2025. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. *Online manuscript* released January 12, 2025. https://web.stanford.edu/~jurafsky/slp3.

Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. 2018. DeepZip: Lossless Data Compression using Recurrent Neural Networks. *arXiv preprint arXiv:1811.08162*, 2018. https://arxiv.org/pdf/1811.08162.

Zhiyong Wu, Yun Chen, Ben Kao, and Qun Liu. 2020. Perturbed Masking: Parameter-free Probing for Analyzing and Interpreting BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4166–4176. Association for Computational Linguistics, 2020. https://aclanthology.org/2020.acl-main.383/

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. 2019. Revealing the Dark Secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 4365–4374, Hong Kong, China, November 3–7, 2019. Association for Computational Linguistics. https://aclanthology.org/D19-1445.pdf.

Hangbo Bao, Li Dong, Furu Wei, Wenhui Wang, Nan Yang, Xiaodong Liu, Yu Wang, Songhao Piao, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2020. UniLMv2: Pseudo-Masked Language Models for Unified Language Model Pre-Training. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119 of *Proceedings of Machine Learning Research*, pages 6507–6518, Online, 13–18 July 2020. PMLR. https://proceedings.mlr.press/v119/bao20a.html.