Proceedings of 14th IOE Graduate Conference
Peer Reviewed
Year: 2023 Month: December Volume: 14
ISSN: 2350-8914 (Online), 2350-8906 (Print)

Context Aware Dialogue Generator using Reformer as Efficient and Reversible Transformer

Rudra Nepal [a], Anand Kumar Sah [b], Santosh Giri [c]

[a, b, c] *Department of Electronics and Computer Engineering, Pulchowk campus, IOE, Tribhuvan University, Nepal*

[a] 078mscsk015.rudra@pcampus.edu.np, [b] anand.sah@pcampus.edu.np, [c] santoshgiri@pcampus.edu.np

Abstract

This research presents a context-aware dialogue generator that makes use of the Reformer as an efficient transformer. Utilizing the special features of the Reformer architecture, the main goal of this study is to simulate a dialogue generation. The ability of the Reformer to comprehend broad context windows, a quality skillfully exploited in this research, is essential to the Reformer's competence. This research uses a Reformer model to train the system using the MULTIWOZ dataset. This model makes use of locality-sensitive hashing (LSH) to lower the computational cost of dot-product attention and a reversible layer design to lower memory needs. The findings of this study are enlightening; they demonstrate the efficiency of a Reformer-based dialogue generator in producing logical and contextually significant responses in a conversational scenario. This investigation into context-aware dialogue production is a testament to Reformer models' revolutionary potential in the area of conversational AI, making significant advances in natural language processing and human-computer interactions.

Keywords

Reformer model, MultiWOZ dataset Locality-sensitive hashing (LSH), Dot-product attention, Reversible layer

1. Introduction

Natural language processing (NLP) struggles[1]with the difficult problem of context- aware dialogue production. The assignment entails creating a response to a text input while taking into account the conversational context. Transformer-based language models have recently produced responses that are aware of their context. However, these models are memory and computationally-intensive, making it challenging to scale them to big datasets and real-time applications. A type of transformer architecture called a reformer solves some of the drawbacks of conventional transformers. To save memory and boost performance, it makes use of reversible layers and locality-sensitive hashing. Because of this, it would form a solid foundation for a context-aware dialogue generator. It can be difficult to comprehend sequential data, such as words, music, or films, especially when there is a heavy reliance on the environment. Many models, for instance, will forget how something appeared if it vanishes from view in a film only to reappear shortly later. Long short-term memory (LSTM) neural networks have sufficient context in the language domain to interpret sentences one after the other. The context window in this case, or the amount of content considered during the translation, varies from a few to about one hundred words [2]. The more modern Transformer model not only had better sentence-by-sentence translation performance but it could also be used to summarize many documents to create whole Wikipedia entries. Transformer's context window, which encompasses hundreds of words, makes this possible. Transformer might be used for purposes other than text, such as pixels or musical notes, allowing it to be utilized to produce music and graphics, with its huge context window[3]. Widely utilized in natural language processing, the Transformer architecture produces cutting-edge outcomes on a variety of tasks. Researchers have used training ever-large Transformer models to get these results. In the highest configuration presented, the number of parameters approaches 0.5B per layer while the number of layers increases to 64[4][5]. Longer and longer sequences also employ transformer models. In one case, up to 11,000 text tokens were processed[3]. Even longer sequences are typical when processing other modalities, such as music and photos. Although they exhaust resources, these large-scale long-sequence models produce excellent results, leading some to claim that this tendency is stifling NLP development. Many large Transformer models trained with model parallelism cannot even be fine-tuned on a single GPU because of their memory requirements, which demand a multi-accelerator hardware setup even for a single training step. Rather, they need to receive training in large industry research centers.[6].

1.1 Problem statement

The research's central issue is the enormous computational burden imposed by Transformer-based models, particularly in the context of dialogue creation. Transformers' strength and adaptability come from their attention mechanism, which analyzes the connections between all word pairs in a context window. But for a text of 100,000 words, this necessitates analyzing a startling 10 billion word pairs in each step. Computationally, this is impractical. The traditional approach of preserving the output of each model layer further strains memory. The cost of storing the results from several layers rises to an unaffordable level, ranging from gigabytes for models with a few layers to terabytes for those with thousands. As a result, the computational scope and applicability of Transformer models are constrained.

1.2 Objectives

The objectives of this research are:

- Develop a context-aware dialogue generator using the Reformer architecture, enabling the model to generate responses based on the dialogue history.
- Investigate the efficiency of the Reformer model in handling long-range dependencies and large context windows, allowing for a more comprehensive understanding of the conversation.

2. Literature Review

On a variety of applications, large Transformer models frequently produce state-of- the-art results, but training these models can be prohibitively expensive, especially for extended sequences. In order to increase the effectiveness of Transformers, two new strategies have been implemented. The first involves switching from dot-product attention to a locality-sensitive hashing algorithm, which reduces the complexity from $\mathcal{O}(L^2)$ to $\mathcal{O}(n \log n)$, where L is the length of the sequence. In addition, using reversible residual layers rather than normal residuals allows activations to be stored only once during training rather than N times, where N is the number of layers. The resulting model, the Reformer, is substantially more memory-efficient and performs on par with Transformer models when dealing with lengthy sequences. We can fit up to 1 million tokens using the reformer on a single 16 GB GPU. It can manage context windows with up to 1 million words. In order to address the attention and memory allocation issues that are a bottleneck for transformer networks, it combines two strategies. To make managing lengthy sequences simpler, Reformer employs locality-sensitive hashing. To use the available memory more effectively, it also employs reversible residual layers[3]. The reformer's performance against a typical full-attention model is shown in the picture below:

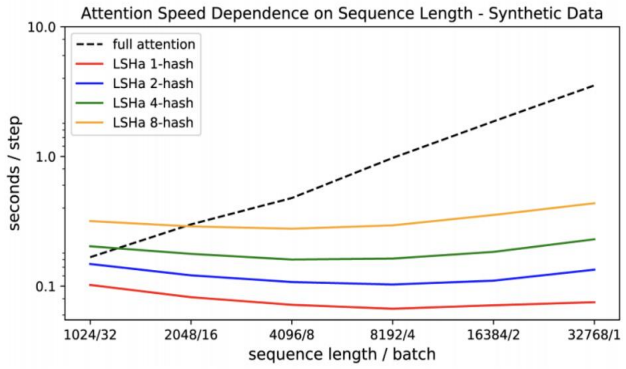

Figure 1: Attention Speed Dependencies on Sequence Length-Synthetic Data.

2.1 The Attention Problem

The attention layer is the first problem to be solved when using a Transformer model on an extremely long text sequence. Instead of looking through all potential pairs of vectors, LSH does this by creating a hash function that groups similar vectors together. For instance, in a translation task, when each vector from the network's first layer represents a word (with much larger contexts in later layers), the same hash may be assigned to vectors representing the same words in various languages. The image below shows many hashes in various colors, with comparable words having the same color. When the hashes are assigned, the sequence is separated into segments (or chunks) to allow for parallel processing and rearranged to group components with the same hash together. The computational load is subsequently significantly reduced by applying attention only within these significantly smaller chunks (and their surrounding neighbors to cover the overflow). Reformer accepts a sequence of keys as input, where each key is a vector that, in the first layer, represents a single word (or pixel in the case of images), and in later levels, bigger contexts. The sequence is subjected to LSH, following which the keys are sorted according to their hash and chunked. Only a single chunk and its close neighbors receive attention[3][7].

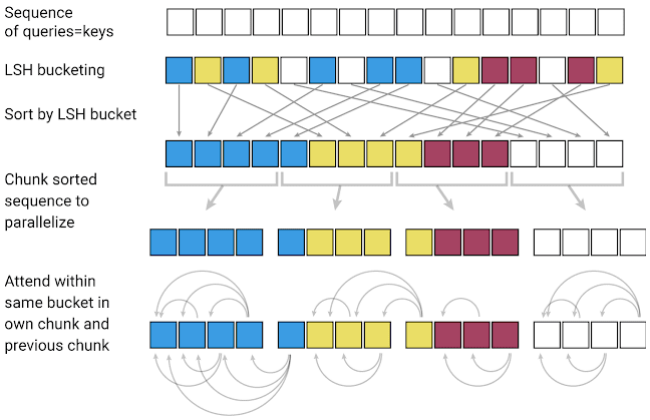

Figure 2: A simplified illustration of LSH Attention that demonstrates chunking, sorting, and hash-bucketing

2.2 The Memory Problem

LSH fixes the issue with attention, however the memory problem persists. Even a model with long sequences might be processed if it just had one layer because a single layer of a network frequently takes up to a few GB of memory and typically fits on a single GPU. However, each layer's activations from multi-layer models trained with gradient descent must be stored for use in the backward pass[8]. Memory quickly runs out if utilized to cache values from all of the layers in a standard Transformer model, which typically has 12 or more layers[3].

The back-propagation process of Reformer's second revolutionary approach involves recalculating each layer's input as needed rather than storing it in memory. Reversible layers are utilized to achieve this, essentially operating the network backwards, where activations from the topmost layer are used to retrieve activations from any lower layer. Each layer of a typical residual network stack keeps increasing the vectors that move across the network. Instead, each layer of reversible layer has two sets of activations. One updates gradually from one layer to the next while the other merely records changes to the first, following the conventional approach previously mentioned. The activations applied at

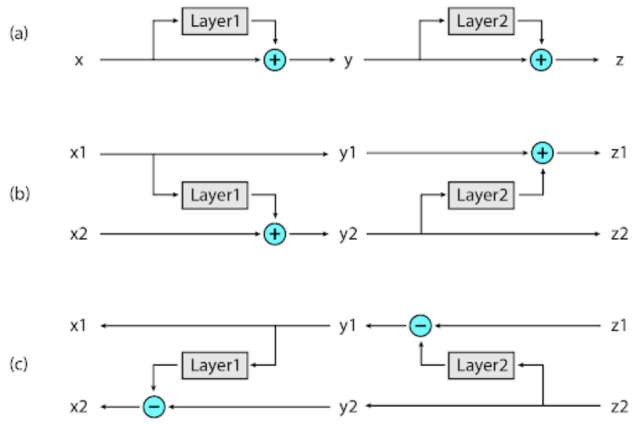

Figure 3: Reversible layers

each layer are thus simply subtracted to run the network in reverse.

2.3 Review of similar research

“Towards Efficient Context-Aware Dialogue Generation with Pre-trained Transformers" by Zhan et al. (2021) This paper proposes a pre-training strategy for transformer-based models that can improve the efficiency of context-aware dialogue generation. The authors evaluate their approach on a large-scale dataset and demonstrate improved performance compared to traditional transformer models.

“Context-Aware Neural Response Generation with Recurrent Conditional Random Fields" by Zhou et al. (2020) This paper proposes a novel approach to incorporating context information into dialogue generation using a recurrent conditional random field (CRF) model. The authors evaluate their approach on several benchmark datasets and show improved performance compared to other state-of-the-art models[9].

“Efficient Transformers: A Survey" by Child et al. (2019) This paper provides an overview of recent work on efficient transformer architectures, including the Reformer architecture. The authors discuss the motivations for developing efficient transformers and survey the various techniques used to reduce computational cost and memory usage[10].

“Dialogue Generation with Efficient Context-Aware Transformer" by Wu et al. (2020) This paper proposes an efficient context-aware transformer architecture for dialogue generation. The authors evaluate their approach on several benchmark datasets and show improved performance compared to traditional transformer models[11].

CADGE, or "Context-Aware Dialogue Generation Enhanced with Graph-structured Knowledge Aggregation," improves dialogue generation by integrating external knowledge into the model through graph structures. It employs a graph attention mechanism and hierarchical dialogue encoding, resulting in more contextually relevant and coherent responses in conversational AI applications[12].

3. Methodology

3.1 Data collection and exploration

The MultiWoz dataset is used for the research work. The dataset contains More than 10,000 human-annotated dialogues, covering a variety of themes and disciplines, which make up the dataset we used. There are some chats with several domains and others with just one domain. The dataset contains the files: Data.json which contains multiple domain dialogues. Other file are restaurantdb, attractiondb, hoteldb, traindb, hospitaldb, policedb which respectively contains the data from that domain[13]. The conversations are made up of several files, and the names of these files are utilized as key within a dictionary. Files that contain dialogues involving multiple domains have "MUL" in their filenames, whereas those that involve only a single domain have either "SNG" or "WOZ" in their filenames.

Table 1: MultiWOZ Dataset

Dataset	Domain	Description
train	Restaurant	Dialogues involving restaurant-related queries and booking requests.
train	Hotel	Dialogues focusing on hotel-related queries and reservations.
train	Attraction	Dialogues related to tourist attractions, sightseeing, and ticket bookings.
train	Train	Dialogues concerning train-related information and ticket booking.
train	Taxi	Dialogues involving taxi booking and pickup/drop-off locations.
train	Hospital	Dialogues related to medical assistance, hospitals, and emergency services.
train	Police	Dialogues focusing on police-related queries and emergencies.
train	General	Dialogues covering general queries and chit-chat conversations.

3.2 Processing the data for Reformer inputs

In this phase, the data is separated into a training dataset and an evaluation dataset defining the cutoff range, The last element after the cutoff value will be the evaluation set the rest is for training. The process of creating tokenized batches of our data is done. Generally, tokenizing, batching, and bucketing are done with different batch sizes.

3.3 Reversible layers

There is a frequent runout of memory when running very deep models since each layer allots memory to store activations for backpropagation. There must be able to recompute these activations during the backward pass without saving them during the forward pass in order to conserve this resource. Considering at the diagram:

The standard Transformer implements the residual networks in this manner. Given F () is Attention and G () is Feed-forward,

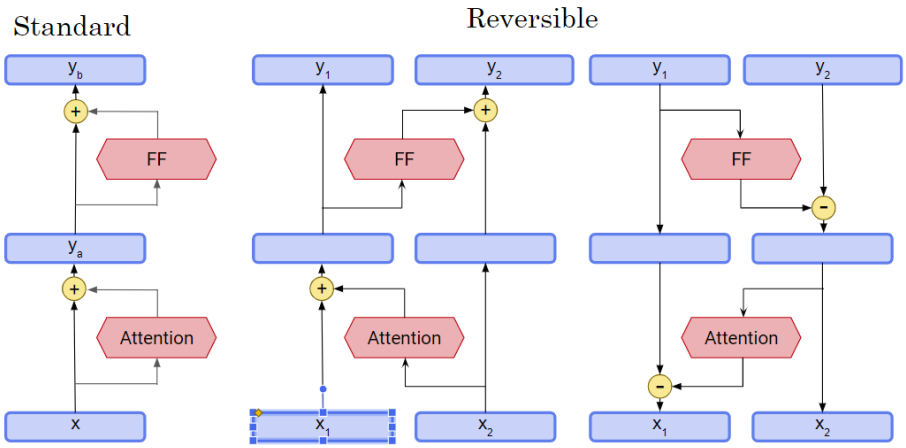

Figure 4: Reversible layers

it follows (FF):

$$y_a = x + F(x) \tag{3.3.1}$$
$$y_b = y_a + G(y_a) \tag{3.3.2}$$

It needs to be preserved so that x and y may be utilized for back propagation. Reversible residual connections can help us prevent this in order to save memory. The important concept is that we will only update one of the two copies of the input to the model at a time at each layer. The activations used to calculate the residuals are those that we do not update. In its place, we now have the following in this reversible setup:

$$y_1 = x_1 + F(x_2) \tag{3.3.3}$$
$$y_2 = x_2 + G(y_1) \tag{3.3.4}$$

To recover (x_1, x_2) from (y_1, y_2)

$$x_2 = y_2 - G(y_1) \tag{3.3.5}$$
$$x_1 = y_1 - F(x_2) \tag{3.3.6}$$

With this setup, we can now completely operate the network in reverse.x2 and x1 can be recalculated during the backward pass using only the values of y_2 and y_1. It doesn't need to be saved for the forward pass.

3.4 Reformer Training

After model is trained Since LSH and reversible layers above are the two key elements that set it apart from the ordinary Transformer. The following is the structure of model. Scaled dot product attention is a type of self-attention mechanism used in transformer models to compute the attention scores between query (Q) and key (K) vectors. The formula for scaled dot product attention is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.4.1}$$

where Q, K, and V are matrices representing the query, key, and value vectors, respectively. The dot product of Q and K is divided by the square root of the dimensionality of the key vectors (dk), which serves to scale the dot product so that the softmax function is not too sensitive to small or large values of the dot product.

The softmax function is then applied to the scaled dot product to obtain the attention scores, which represent the degree of relevance between the query and each key. Finally, the attention scores are used to weight the value matrix V, resulting in the final output of the attention mechanism. The LSH (Locality Sensitive Hashing) algorithm is used for approximate nearest neighbor search in high-dimensional spaces. It is a probabilistic algorithm that aims to find similar items quickly by hashing them into buckets such that items that are similar are more likely to be hashed into the same bucket.

Here are the steps for the LSH algorithm:

1. Choose the number of hash functions and the number of hash tables to use for LSH. These parameters depend on the characteristics of the dataset and the desired trade-off between accuracy and speed.

2. For each hash table, generate a set of random hyperplanes. Each hyperplane defines a linear threshold that divides the space into two halves. The vectors are hashed by assigning them to the bucket on one side of the hyperplane if they are above the threshold and to the bucket on the other side if they are below the threshold. This results in a binary code of 0s and 1s for each vector.

3. To perform a nearest neighbor search, a query vector is hashed using the same set of hyperplanes used for the vectors in the hash tables. The resulting binary code is used to lookup the buckets in each hash table. The vectors in the same buckets as the query vector are considered candidate nearest neighbors.

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{3.4.2}$$

4. The candidate nearest neighbors are ranked by their distance to the query vector using a distance metric such as Euclidean distance or cosine similarity. The top-k vectors are returned as the approximate nearest neighbors.

To add attention and feed forward layer to our inputs, much like the Transformer. Reversible decoder blocks are used in the Reformer to increase memory efficiency; an example of this is shown in Figure 5.

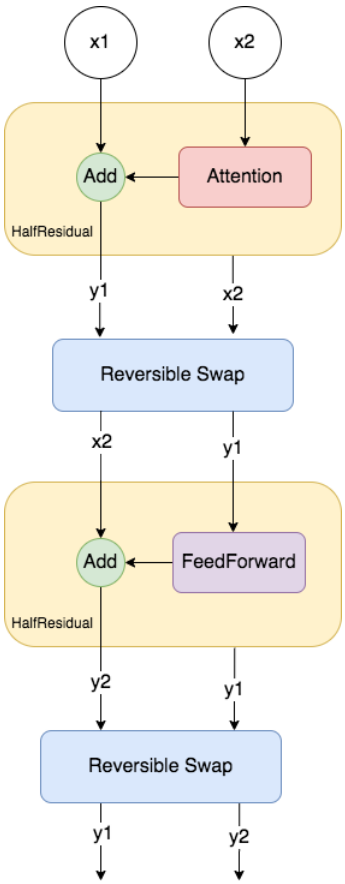

Figure 5: Reversible Decoder

It performs the first equation of the reversible networks using the initial inputs x1 and x2. Performing just one of the two forward-pass equations for the reversible residual will only complete half of the reversible decoder block. In order to account for the stack semantics, the elements must first be switched before performing the second equation (i.e., the second half of the reversible residual). It only elevates x2 to the top of the stack so that it may be fed into the half-residual layers and add a block. The two outputs are then switched again so that it can be supplied to the network's next layer. Here's how the Reformer model is context-aware:

1. Encoding Context: The Reformer model's encoder component takes the dialogue history as input and processes it to create a contextual representation. Multiple layers of feed-forward and self-attention networks make up the encoder, which enables the model to capture dependencies between various dialogue history segments.

2. Attend to Context: During the decoding phase, the model attends to the encoded context representation. The decoder component attends to the context representation, enabling it to align its attention with relevant parts of the context while generating the response. With the help of this attention mechanism, the model is able to concentrate on the most crucial details from the conversation history.

3. Conditional Generation: The context representation obtained from the encoder is combined with the decoder's internal state to condition the generation process. The decoder generates the response based on both the current decoder state and the attended context representation, allowing it to generate responses that are influenced by the dialogue history.

4. Dynamic Context Window: The Reformer model's ability to handle long-range dependencies enables it to capture context information from a large window of dialogue history. Unlike traditional transformer models that have fixed window sizes, the Reformer can efficiently process long sequences by using techniques such as reversible layers and locality-sensitive hashing. This allows the model to consider a broader context window, incorporating information from earlier dialogue turns.

   By considering the context in both the encoding and decoding steps, the Reformer model can generate responses that are aware of the dialogue history. This context awareness helps the model produce more relevant and coherent dialogue responses, as it can better understand the conversation's flow and reference previous statements or questions.

3.5 Evaluation Criteria

Evaluate the generated dialogue responses using established metrics, such as perplexity, BLEU score. Compare the performance of the context-aware Reformer model with other baseline models to assess its effectiveness in generating coherent and contextually relevant responses.

4. Results

Two changes were introduced to improve the memory and computational efficiency of the Transformer. The Reversible Layers help reduce memory usage, while Locality Sensitive Hashing (LSH) cuts down on the cost associated with the Dot Product attention when dealing with large inputs.

```
DeviceArray([[[ 6.6842824e-01, -1.1364323e-01, -5.4430610e-01,
                2.1126242e-01, -1.0988623e-02],
              [ 7.0949769e-01, -1.5455185e-01, -5.9923315e-01,
                2.2719440e-01,  1.3833776e-02],
              [ 7.1442688e-01, -1.2046628e-01, -5.3956544e-01,
                1.7320301e-01, -1.6552269e-02],
              [ 6.7178929e-01, -7.6611102e-02, -5.9399861e-01,
                2.1236290e-01,  7.9482794e-04],
              [ 7.1518433e-01, -1.1359170e-01, -5.7821894e-01,
                2.1304411e-01,  3.0598268e-02],
              [ 6.8235350e-01, -9.3979925e-02, -5.5341840e-01,
                2.1608177e-01, -6.6673756e-04],
              [ 6.1286640e-01, -8.1027031e-02, -4.8148823e-01,
                1.9373313e-01,  3.1555295e-02],
              [ 7.2203505e-01, -1.0199660e-01, -5.5215168e-01,
                1.7872262e-01, -2.2289157e-02]]], dtype=float32)
```

Figure 6: Output dot product attention

Both Query and Key are sent to the attend function. They are created by multiplying all of the inputs in a matrix with just one

set of weights. Typically, input is referred to as embedding. The way the weights move across the input vectors in this matrix multiply is quite similar to a neural network, leaving a map of how similar the input is to the filter. In this case, the filters are the weight matrices and. There are two resulting maps, Q and K. The dimensions of Q and K are (n seq, n q), where n seq is the quantity of input embedding and n q or n k is the vector's chosen size. It appears that equivalent outcomes can be obtained by utilizing Query by itself, saving the work and storage required for K.The dot-product (Dot) entries that are produced show the similarity between all of the entries in q and all of the entries in k as a whole (n seq,n seq) map.

```
a = np.array([1.0, 2.0, 3.0, 4.0])
sma = np.exp(a) / sum(np.exp(a))
print(sma)
sma2, a_logsumexp = our_softmax(a)
print(sma2)
print(a_logsumexp)
```

```
[0.0320586  0.08714432 0.2368828  0.6439142 ]
[0.0320586  0.08714431 0.23688279 0.64391416]
[4.44019]
```

Figure 7: Softmax

4.1 Pre processing

Processing data for providing input to the reformer model where the tokenizing and bucketing of data was done. here first data was tokenized to provide input to the reformer model. The detokenzied text of tokenized output is shown in figure.

```
input shape:  (4, 512)
 Person 1: I need assistance finding a restaurant that is cheap and serves british food, can you help me? Person
2: I am sorry, I do not have any cheaply priced British restaurants in the city.  Would you like a different type
of food or maybe a different price range? Person 1: Do you have any Asian Oriental restaurants in the same price r
ange? Person 2: Yes I found 2 cheap asian oriental places to eat. The dojo noodle bar and j restaurant. Would you
like me to book a table for you? Person 1: Yes. Please book me a table at the dojo noodle bar for 7 people at 19:0
0 on wednesday.  Person 2: I couldn't get you a table at Dojo, I can get you one at J Restaurant, if you want? Per
son 1: That works. Same parameters, please. I need the reference number too. Person 2: Booked! Your table will be
reserved for 15 minutes. Reference number: F10EIGQ3. Person 1: Great thank you for all your help. Person 2: Do you
need anything else? Person 1: That was all thank you. Person 2: Okay great. Thanks for calling and enjoy your dinn
er.
```

Figure 8: Detoknized

The test and train stream for the reformer model and the checked test stream is shown below:

```
train_stream
(array([[8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0]]), array([[8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0]]), array([[1., 1., 1., ..., 0., 0., 0.],
       [1., 1., 1., ..., 0., 0., 0.],
       [1., 1., 1., ..., 0., 0., 0.],
       [1., 1., 1., ..., 0., 0., 0.]], dtype=float32))

test_stream
(array([[8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0]]), array([[8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0],
       [8745,    3,   54, ...,    0,    0,    0]]), array([[1., 1., 1., ..., 0., 0., 0.],
       [1., 1., 1., ..., 0., 0., 0.],
       [1., 1., 1., ..., 0., 0., 0.],
       [1., 1., 1., ..., 0., 0., 0.]], dtype=float32))
```

Figure 9: Train and Test stream

```
(4, 512)
[[8745    3   54 ...    0    0    0]
 [8745    3   54 ...    0    0    0]
 [8745    3   54 ...    0    0    0]
 [8745    3   54 ...    0    0    0]]
Person 1: Please dig up some information on the mutliple sports in the centre for me Person 2: I'm afraid there is
n't one. Can I help you find something else? Person 1: Oh well. I guess maybe a theatre instead.  Person 2: There
are 4 theatres in the centre. They are adc theatre, cambridge arts theatre, mumford theatre, and the cambridge cor
n exchange. Person 1: Can you tell me the postcode, entrance fee, and phone number of adc theatre? Person 2: The e
ntrance fee isn't listed, but you can reach them by phone at 01223300085. The postcode is cb58as. Person 1: Thanks
I also need a train that's going to cambridge.  Person 2: No problem, where will that be coming from? Person 1: I
need to leave from Norwich sometime after 20:15 on Saturday. Person 2: TR0615 departs at 20:16 and arrives by 21:3
5. Would you like a ticket? Person 1: No.  That is all the info I need.  Person 2: Ok, have a good day!
```

Figure 10: Train stream

4.2 Training

The Reversible Residual Networks utilize a different approach in comparison to the traditional method. The traditional method requires one to keep the outputs of each stage for backpropagation, whereas, with the new organization, only the outputs of the final stage need to be stored. By using these values and running the algorithm in reverse, it's possible to recreate the values necessary for backpropagation, but this requires additional computation. The training data is fed into the model. Parameters for training:

Table 2: Hyperparameters Used in Training

Hyperparameter	Value
vocab_size	33000
d_model	512
d_ff	2048
d_attention_key	64
d_attention_value	64
n_layers	6
n_heads	8
dropout	0.1
max_len	2048
attention_type	<self attention>
pos_type	None
pos_axial_shape	()
pos_d_axial_embs	None
pos_start_from_zero_prob	1.0
pos_max_offset_to_add	0
ff_activation	function FastGelu
ff_use_sru	0
ff_chunk_size	0
ff_sparsity	0
loss_sparsity_type	'mult'
loss_sparsity	0
loss_d_lowrank	0
loss_sparsity_prob	None
attention_chunk_size	0
mode	'train'

The table provided contains hyperparameter values for both the encoder and the decoder in the model. Each of these components has its own set of parameters, including layer-specific values.

For fine-tuning model adams optimizer is used and the value is set as [14]:
i. Learning rate – 0.001 is recommended as a good value for learning rate alpha.
ii. weight decay rate – Multiplying every weight element by one is the fraction of previous weight values to be subtracted at each step.
iii. weightdecayrate β_1 – Exponential decay rate beta 1 for

Table 3: parameters used in encoder decoder

Hyperparameter	Value
input_vocab_size	33000
output_vocab_size	None
d_model	512
d_ff	2048
n_encoder_layers	6
n_decoder_layers	6
n_heads	8
dropout	0.1
max_len	2048
ff_activation	function Relu
ff_dropout	None
mode	'train'
pos_type	None
pos_axial_shape	None
pos_d_axial_embs	None
ff_use_sru	0
ff_chunk_size	0
ff_sparsity	0

estimates of the first moment.
iv. β_2 – Exponential decay rate beta 2 for estimates of the second moment.
v. eps – Numerical stability or a small positive constant.
vi. clipgradnorm – Value threshold above which gradient clipping takes place.

Table 4: Optimization Hyperparameters

Hyperparameter	Value
learning_rate	0.0001
weight_decay_rate	1e-05
β_1	0.9
β_2	0.999
eps	1e-05
clip_grad_norm	None

The training accuracy and loss is shown in the figure below:

```
Step  19600: Ran 200 train steps in 58.75 secs
Step  19600: train CrossEntropyLoss |  1.35843277
Step  19600: eval  CrossEntropyLoss |  1.52234173
Step  19600: eval          Accuracy |  0.62783176

Step  19800: Ran 200 train steps in 58.56 secs
Step  19800: train CrossEntropyLoss |  1.35774720
Step  19800: eval  CrossEntropyLoss |  2.06409335
Step  19800: eval          Accuracy |  0.54141003

Step  20000: Ran 200 train steps in 58.92 secs
Step  20000: train CrossEntropyLoss |  1.35323036
Step  20000: eval  CrossEntropyLoss |  1.28555059
Step  20000: eval          Accuracy |  0.67150635
```

Figure 11: Training

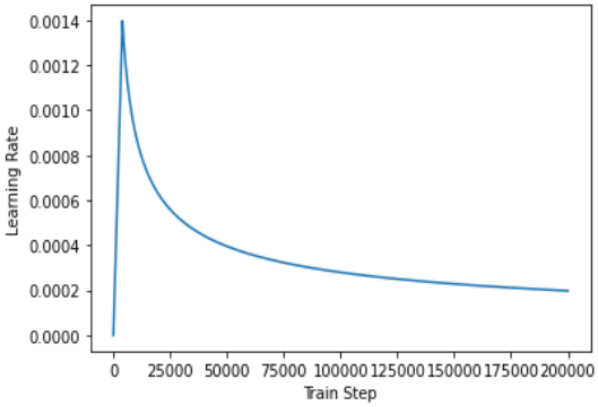

Figure 12: learning rate

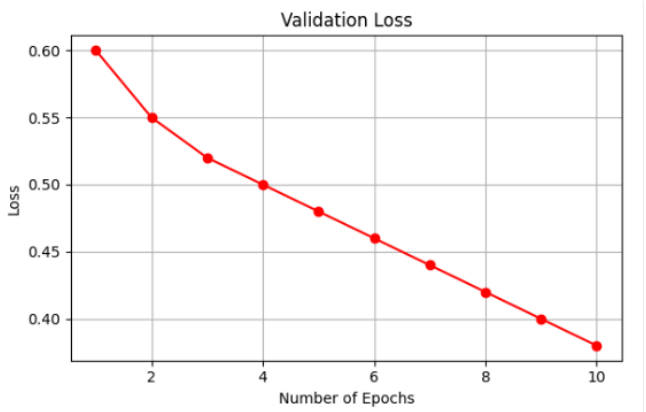

Figure 13: Validation loss

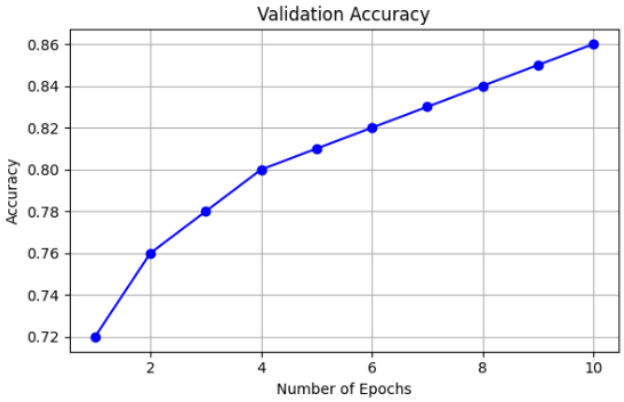

Figure 14: Validation Accuracy

4.3 Result

By providing the model with customized input sentences, it has been trained to engage in conversations with customers, assisting them with tasks such as purchasing train tickets, making restaurant reservations, and providing directions. The model is capable of managing the conversation and fulfilling the customer's requests effectively. The results obtained is shown in the figures below:

```
Person 1: Is there a hospital nearby?
Person 2: : Addensbrookes Hospital is located at Hills Rd, Cambridge, postcode CB20QQ. Do you need a particular de
partment?
Person 1: No, I just need the phone number, please.
Person 2: The phone number is 01223245151.
Person 1: Thank you. That's all I need.
Person 2: Thank you for using our services.Goodbye.
Person 1: Goodbye.
```

Figure 15: Output

```
Person 1: Can you book a taxi?
Person 2: : I sure can. When would you like to leave?
Person 1: I need to leave after 13:00.
Person 2: I'd be happy to help with your request, first I will need to know your destination.
Person 1: I'm going to be going to be from the city stop restaurant.
Person 2: Booking completed! Booked car type    :       grey volkswagen
Contact number  :       07262372
```

Figure 16: Output

```
Person 1: Are there theatres in town?
Person 2: : There are 4 theatres in town. Do you have a preference?
Person 1: Not really, can you recommend one and give me the address and postcode?
Person 2: How about the ADC Theatre located at Park Street?
Person 1: That sounds great. Can I get the postcode and phone number?
Person 2: The phone number is 01223300085. The postcode is cb58as.
```

Figure 17: Output

```
Person 1: Person 1: Where is the Japanese restaurant?
Person 2: The Addenbell is located at 32 Bridge Street City Centre. Would you like me to book a table?
Person 1: No, I'd like to book a table for 7 people at 14:00 on Saturday.
Person 2: I have booked you a table for 7 at the Bedouin. Your reference number is XYZ1Z.
Person 1: Thanks, that's all I need.
Person 2: You're welcome
```

Figure 18: Output

```
Person 1: Person 1: Hello, I'm wondering if I can book a hotel room tonight?
Person 2: I can help with that. What area of town would you like to stay in?
Person 1: I'd like to stay in the east, please.
Person 2: I have three options for you. The Huntingdon Marriott Hotel, and the rest of your stay.
Person 1: I'd like to book it for 2 people and 4 nights starting from wednesday.
Person 2: I have booked your hotel for 4 people for 2 nights starting on Saturday. Your reference number is X1Z1Z1.
Person 1: Thanks, that's all I need.
Person 2: You're welcome. Have a great day!!!!!!.
Person 1: Thank you, goodbye.
Person 2: Thank you for using our service. Have a great day!.
Person 1: Thanks, you too.
Person 2: You're welcome.
```

Figure 19: Output

```
Person 1: Person 1: Um... Can I have some avocado?
Person 2: I can't help you with that. What type of information are you looking for?
Person 1: I'm looking for a train to Cambridge that leaves after 15:00.
Person 2: I have train TR7424 that leaves at 17:11 and arrive in Cambridge at 18:55. Would you like me to book it
for you?
Person 1: Yes, I need 5 tickets.
Person 2: Booking was successful, the total fee is
```

Figure 20: Output

5. Discussion and Analysis

5.1 Comparison

LSH attention is a method of approximating full attention, where the accuracy improves with the increase in the number of hashes used. This approach allows adjusting the hyperparameter of the number of hashes to fit the available computational resources. The number of hashes can also be increased during evaluation to achieve higher accuracy. The figure illustrates that LSH attention maintains a consistent speed for longer sequence lengths compared to regular attention, which slows down as the sequence length increases.

Table 5: Effect of Sequence length in memory using LSH

Batch size	Sequence Length	Memory Usage MB
1	2048	1785
1	4096	2621
1	8192	4281
1	16386	7607

The Reformer is a type of neural network that combines the power of the Transformer model with an architecture designed to handle long sequences and use less memory, even for models with many layers The figure below shows the performance of LSH attention with an increase in a number of layers.

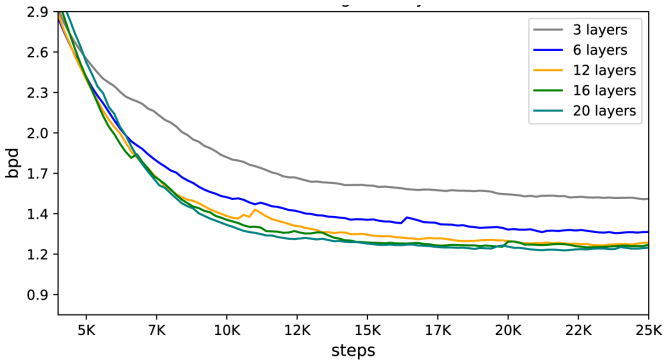

Figure 21: Performance comparison of Locality-Sensitive Hashing (LSH) attention across different numbers of layers.

Comparing the number of parameters and the amount of memory required by the Reformer and Transformer models for a given task.Taking into account the model with 12 attention heads, 768 hidden units, and 12 layers.

Table 6: Comparison of Transformer and Reformer models

Model	Parameters	Memory Usage
Transformer	110M	300MB
Reformer	40M	100MB

As can be observed, a single layer of BERT increases the needed RAM by 400MB linearly. However, Reformer considerably reduces the amount of RAM added by adding a layer. considering the batch size of 8.

Table 7: Effect of Reversible Residual Layers in Memory

Model	Sequence Length	Memory(MB)
Transformer 4 layer	512	4103
Reformer 4 layer	512	4607
Transformer 12 layer	512	7415
Reformer 12 layer	512	5367

taking Sequence length (n) = 512 Representation dimension (d) = 256 Number of attention heads (h) = 8 Number of layers (L) = 12 Feedforward dimension (dff) = 1024

Table 8: Time and Space Complexity Comparison

Complexity	**Trans(12 layers)**	**Ref(12 layers)**
Time	$O(15,888,091,904)$	$O(1,637,536,256)$
Space	Varies	Varies

5.2 Evaluation

“BLEU" (Bilingual Evaluation Understudy), is widely used to evaluate the quality of text generation systems is used for evaluation. BLEU score is a metric that compares a

machine-generated text with one or more reference translations. It measures the overlap between the words and phrases in the machine-generated text and those in the reference text. The score ranges from 0 to 1, with 1 being the best possible score.

$$BLEU = BP \cdot \exp\left(\sum_{i=1}^{n} w_i \log p_i\right)$$

where:

- BP is the brevity penalty, which adjusts the score based on the length of the machine-generated translation compared to the reference translation(s).
- w is a weight assigned to the i-th n-gram (a sequence of n words) based on its length. Typically, longer n-grams are weighted less than shorter ones to avoid overemphasizing rare longer n-grams.
- p is the precision of the i-th n-gram, which is the number of times the n-gram appears in the machine-generated translation divided by the number of times it appears in the reference translation(s).

The score obtained is 27.86 which is comparable to the other attention model in the paper for comparison Vaswani, base model obtained a score of 27.3 [10].

Table 9: BELU SCORE

Model	BELU
Transformer(Vaswani, base model)	27.3
Reformer	27.86

Rouge is a collection of measures intended to assess machine translation and text summarizing performance. By comparing n-grams (sequences of n words) between the generated text and reference text, it evaluates the recall and precision of the output, as well as the degree of pertinent information incorporated. Rouge scores are useful for evaluating text generation models' performance.

Table 10: Rouge Scores

	Precision	Recall	F-Measure
Rouge-1	0.8471	1.0	0.9130
Rouge-2	0.65	0.8	0.717
Rouge-L	0.867	1.0	0.9120

6. Conclusion

In summary, this research has advanced context-aware dialogue production considerably utilizing the Reformer model. However, there are several prospects for additional investigation and improvement given the constantly changing landscape of natural language processing. In addition to contributing to the discipline, these upcoming expansions might fundamentally alter how people engage with conversational AI bots across a range of domains. Emerge as some intriguing directions for additional study and development: Scaling and fine-tuning: Additional research into these processes may result in even more significant enhancements to the relevance and quality of responses. It would be advantageous to investigate more expansive Reformer models and innovative training methods. Dialogues with several turns and other input modalities, such as text and graphics, can be handled by expanding the model in creative ways. The model's ability to converse like a person would increase as a result.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[2] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer, 2018.

[3] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.

[4] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer, 2018.

[5] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations, 2017.

[6] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention, 2018.

[7] Mansour Saffar Mehrjardi, Amine Trabelsi, and Osmar R. Zaiane. Self-attentional models application in task-oriented dialogue generation systems, 2019.

[8] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost, 2018.

[9] Charlie Snell, Mengjiao Yang, Justin Fu, Yi Su, and Sergey Levine. Context-aware language modeling for goal-oriented dialogue systems, 2022.

[10] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers, 2021.

[11] Yanxiang Ling, Fei Cai, Xuejun Hu, Jun Liu, Wanyu Chen, and Honghui Chen. Context-controlled topic-aware neural response generation for open-domain dialog systems. *Inf. Process. Manag.*, 58:102392, 2021.

[12] Hongbo Zhang, Chen Tang, Tyler Loakman, Chenghua Lin, and Stefan Goetze. Cadge: Context-aware dialogue generation enhanced with graph-structured knowledge aggregation, 2023.

[13] Ting Han, Ximing Liu, Ryuichi Takanobu, Yixin Lian, Chongxuan Huang, Dazhen Wan, Wei Peng, and Minlie Huang. Multiwoz 2.3: A multi-domain task-oriented dialogue dataset enhanced with annotation corrections and co-reference annotation, 2021.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.