



MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES



GeNN: The Art of Adgroup Generation

ABDELRAHMAN MAGED ABDELRAHMAN MAHMOUD

MASTER THESIS

Department of Computer Science and Engineering

Thesis Supervisor

Assoc. Prof. Dr. Mustafa Ağaoğlu

Thesis Co-Supervisor

Assoc. Prof. Dr. Ahmet Bulut

ISTANBUL, 2022



MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES



GeNN: The Art of Adgroup Generation

ABDELRAHMAN MAGED ABDELRAHMAN MAHMOUD
(527619018)

MASTER THESIS

Department of Computer Science and Engineering

Thesis Supervisor

Assoc. Prof. Dr. Mustafa Ağaoğlu

Thesis Co-Supervisor

Assoc. Prof. Dr. Ahmet Bulut

ISTANBUL, 2022

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

Abdelrahman Maged Abdelrahman Mahmoud, a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended his thesis entitled, “**GeNN: The Art of Adgroup Generation**”, on April 27, 2022 and has been found to be satisfactory by the jury members.

Jury Members

Assoc. Prof. Dr. Mustafa Ađaođlu (Co-Advisor)
Marmara University, Computer Engineering Department

Assoc. Prof. Dr. Ahmet Bulut (Co-Advisor)
Acibadem University, Head of Computer Science Department

Assoc. Prof. Dr. Murat Can Ganiz (Jury Member)
Marmara University, Computer Engineering Department

Asst. Prof. Dr. Barıř Arslan (Jury Member)
Acibadem University, Computer Science Department

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Abdelrahman Maged Abdelrahman Mahmoud be granted the degree of Master of Science in Department of Computer Engineering, Data Science Program on April 27, 2022. (Resolution no:).

Director of the Institute
Prof. Dr. Bülent EKİCİ

ACKNOWLEDGEMENT

First and foremost, I would like to praise Allah the Most Gracious for His endless blessings.

I am deeply indebted to my advisor Assoc. Prof. Ahmet Bulut for granting me the opportunity to be his student. His remarkably high standards in setting the knowledge foundations, instilling academic and technical skills, and conducting rigorous research throughout my undergraduate and graduate studies can not be overstated. It is my honor to write this thesis, the fruit of our labor and diligence.

My ultimate thanks is dedicated to my parents whose invaluable advice and wisdom continue to guide me throughout my life.

I would like to express my gratitude for my siblings' love, care, and support, and I truly wish all of them the best.

TABLE OF CONTENTS

	Page
1 INTRODUCTION	1
1.1 Our contribution	2
2 RELATED WORK	4
2.1 Ad generation	4
2.1.1 From Summaries to Ads	4
2.2 Keyword generation	5
3 FOUNDATIONS AND DATA DESCRIPTION	7
3.1 Ad-creative Dataset	7
3.2 Keyword Dataset	7
3.3 Pre-processing	9
3.4 Vectorization	9
3.4.1 GloVe	10
3.4.2 fastText	10
4 GENERATIVE NEURAL NETWORKS	11
4.1 RNN	12
4.1.1 LSTM	12
4.1.2 GRU	13
4.1.3 Training generative RNNs	14
4.2 Transformer	15
4.2.1 Encoder	15
4.2.2 Decoder	15
4.3 Sampling methods	16
4.4 Ad generation	16
4.5 Keyword generation	17
5 EXPERIMENTAL EVALUATION	20
5.1 Ad-creative Evaluation	20
5.2 Keyword Evaluation	22
6 CONCLUSION	26
7 REFERENCES	28
Appendices	32
A GENN	33
A.1 LSTM	33
A.2 GRU	34

A.3 GPT-2	35
---------------------	----

ABSTRACT

GENN: THE ART OF ADGROUP GENERATION

Keywords: search marketing, generating ads & keywords, deep learning

A search campaign consists of a set of ad groups, each of which contains a related set of keywords and ads. During a typical campaign run, search marketers experiment with different marketing messages from subtle to strong, with different keywords from broadly relevant to specifically relevant, and with different landing pages from informative to transactional. The ability to generate new ads and keywords when needed is key to increase the efficiency of search campaign management because advertisers pause poorly performing ads and keywords and replace them with new ones. We propose *GeNN* for the generation of both ads and keywords. For this purpose, we built an easy to use library called *GeNN* for generating ads and keywords programmatically. We validated the quality of the ads and keywords generated by GeNN using standard text summarization metrics. Further, we perform a field experiment with generated ads to validate their performance and use Google's forecasts to validate the performance of generated keywords.

ÖZET

GENN: REKLAM GRUBU ÜRETME SANATI

Anahtar Kelimeler: sponsorlu arama, anahtar kelime ve reklam üretme, derin öğrenme

Arama motoru reklam kampanyası, bir dizi anahtar kelime ve reklam grubundan oluşur. İyi seçilmiş anahtar kelimeler, reklamı yapılacak ürünün arama motoru sorgu sonuçlarında ön sırada yer almasını sağlar. Reklamcılar, düzinelerce farklı anahtar kelime ve reklam mesaj varyasyonlarını kampanya süresince test ederler. Elde edilen test sonuçlarına göre de düşük performans gösteren anahtar kelimeler elenir ve yerlerine başarı olasılığı daha yüksek alternatif anahtar kelimeler konur. Her bir reklam kampanyası için bu tür dögüsel ve deneysel çalışmalar yapmak ve de sonucunda başarı elde etmek reklamcılar açısından bir hayli güçtür. Bu yükü hafifletmek için bir sistematik dahilinde reklam ve anahtar kelimeleri otomatik olarak oluşturan GeNN adlı kullanımı kolay bir kütüphane geliştirdik. GeNN tarafından oluşturulan reklamların ve anahtar kelimelerin kalitesini, standart metin özetleme metriklerine ek olarak Google'un sunduğu anahtar kelime performans tahmin servislerini kullanarak ölçtük. Elde ettiğimiz sonuçlar, GeNN'in online reklamcılık sektöründe gerçek saha testlerine ve pratik kullanıma uygun olduğunu göstermiştir.

LIST OF FIGURES

- 1.1 Advertisers create campaigns, which consist of multiple ad groups. Each ad group contains a set of related keywords and ads. 1
- 4.1 The training and inference pipelines used for keyword and ad generation with the proposed models. 12
- 4.2 The basic building blocks of (a) LSTM networks and (b) GRU networks. 18
- 4.3 The RNNs are trained using shifted versions of the keywords as shown here for the keyword “free java course online”. At each time step, the keyword is shifted to the left. 19
- 4.4 A transformer consists of eight sequentially connected encoders and eight sequentially connected decoders. Input is fed to the first encoder and is evicted from the last encoder. Then, it is fed to the encoder-decoder attention layer of every decoder. 19
- 5.1 The expected CTRs of the generated keywords of varying length. The ground-truth data is shown as a solid line. 25

1 INTRODUCTION

In order to advertise a product online in a search campaign, an advertiser needs to work with keywords and ad creatives, i.e., ads, that best describe the product. The keywords and ads are placed in an ad group, which lives underneath a campaign. According to the keywords chosen, the ad brokers, i.e. search engines, determine which search engine result pages (SERPs) are eligible to display the product's ad. A SERP is the search engine's response to a user's query. If the user's query matches verbally or semantically with a keyword in a certain ad group, then one of the ads in that ad group is displayed to the user within the SERP.

All stakeholders from users and advertisers to ad brokers benefit from the ability to show relevant ads to relevant users, thereby charging advertisers fairly and making users happy. It's imperative for advertisers to choose the right set of keywords and create the right set of ads.

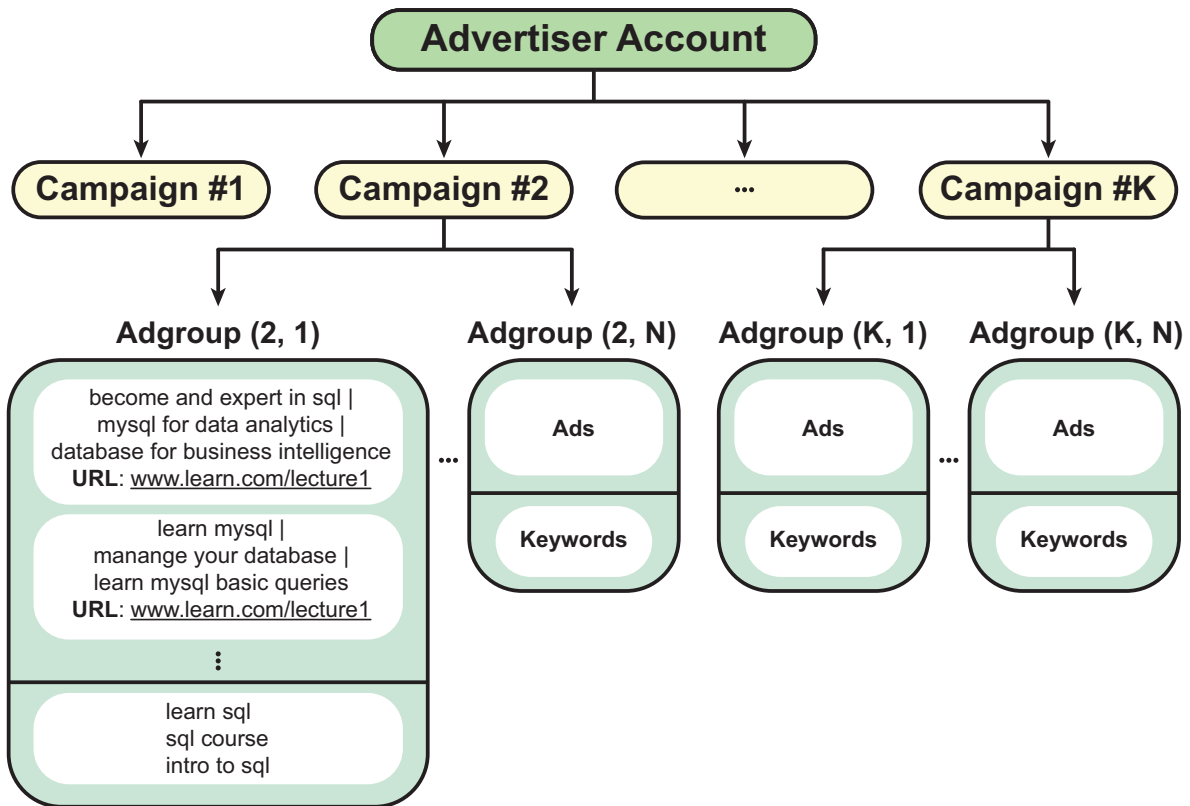


Figure 1.1: Advertisers create campaigns, which consist of multiple ad groups. Each ad group contains a set of related keywords and ads.

Figure 1.1 depicts an example campaign structure. An ad contains a marketing message and a target URL, i.e., the landing page for the advertised product. Typically, an ad has three headlines, each with a maximum of 30 characters. Headlines are separated by a pipe symbol, i.e. |. In addition, the ad has two descriptions, each containing at most 90 characters. Since managing thousands of ads is labor intensive, advertisers tend to resort to a handful of ad templates, which usually results in sub-optimal performance. Additionally, the marketing team has to experiment with different keywords from broadly relevant (to the product) to exactly relevant. For instance, bidding on the keyword “online java course” for online learning is common sense. Since it is an obvious keyword, other advertisers would bid on it as well. Hence, the profit margins on obvious keywords tend to be low. It is sometimes better to use less obvious keywords, which cost less per user acquisition [1].

1.1 Our contribution

The failure to fail fast would result in marketing budget being wasted. For rapid experimentation, we built GeNN for generating ads and keywords programmatically. Ads and keywords that have poor field performance could be paused and replaced with other keywords and ads that are generated automatically. We benchmarked the performance of GeNN using standard evaluation metrics found in the language modeling literature. We deployed the ads generated via GeNN in an actual search campaign for collecting field performance data. Additionally, we reported the performance forecast provided by Google in order to quantify the efficacy of keywords generated via GeNN.

Contrary to the existing literature on keyword and ad generation, GeNN is primarily based on language modeling. All internal models in GeNN learn patterns in the input by treating the output as an adaptation of the input. In LSTMs and GRUs, the “shifted-by-one” mechanism allows these RNN variants to capture the patterns in the input with proper weights. Therefore, these models can be used in generating different alternatives during inference for the same seed.

Both GPT-2 generator and GPT-2 summarizer models in GeNN are language models each with a different overhead. These models treat text as input and output pairs. In GPT-2 summarizer, the task designator “TL;DR” is used to trigger different behavior or weights in the same language model. But, the input and output are the exactly the same text. The treatment of ad generation and keyword generation tasks as language

modeling tasks gave rise to the general purpose framework GeNN, which could be used in the search advertising practice.

2 RELATED WORK

We divided this section into two in order to discuss the existing literature on the generation of ads and keywords separately.

2.1 Ad generation

Ad generation is composed of two tasks, often tackled individually. The first is text extraction from landing pages. The second is re-writing the extracted summary to make it suitable for advertising. Since in both operations, text is reduced in length to meet the character length limitations, both tasks are considered text summarization tasks.

It is straightforward to extract unique text from HTML documents using rule-based information retrieval methods. The output is a short summary composed of a few sentences that ideally represent the key points in the original web page. The three key steps in extractive text summarization systems are:

1. **Creating intermediate representations for sentences.** These are typically frequency-based vectors such as word count or TF-IDF vectors rather than context-based vectors.
2. **Scoring each sentence.** Each sentence is scored in order to quantify its importance in the whole text.
3. **Selecting summary sentences.** A subset of the sentences are chosen according to their importance in order to comprise the final summary [2].

Optimization strategies of selecting the final summary sentences have shown good performance [3]. TextRank, an approach inspired by the popular PageRank algorithm, creates a graph of sentences where sentence importance converges in an iterative process [4].

2.1.1 From Summaries to Ads

Thomaidou et al. extracted the promotional text from the product’s landing page and used a traditional summarization method in order to shorten the summary to meet any length limits. They finalized each summary with a call to action such as “Order Now!” [5]. In a different study, they proposed an n -gram language model for writing such summaries [6]. Hughes et al. proposed the usage of two encoder-decoder RNNs in

parallel, one for generating the title, i.e the headlines, and the other for generating the body, i.e. the descriptions. Using a dataset containing landing page-ad pairs, they were able to learn the mapping between the two [7].

2.2 Keyword generation

Seed keywords are fed into search engines, and the resulting web pages are used to generate new keywords. Joshi and Motwani used text snippets from such web pages to construct TermsNet, a directed relevance graph where vertices are words and edges are similarity scores [8]. Using this graph, new keywords were suggested by navigating its links. Another approach called Wordy extended TermsNet to include text snippets as well as the whole documents [9]. In this way, they were able to enrich the suggested keywords. A semi-supervised method relied on user-defined labels to improve accuracy [10]. Using label propagation, all prospective keywords are determined.

To improve the ranking of a website in organic search results, content writers use meta-tags in web pages. Meta-tag crawlers exploit such keywords. As in proximity search, seed keywords are looked up using search engines, but instead of mining the actual content of the pages, meta-tags are extracted. Using these tags, similarity and co-occurrence matrices can be built to identify relevant keywords. Popular tools utilizing meta-tags are WordTracker and WordStream [11, 12].

The query logs provided by search engines as well as the advertiser logs are mined for creating co-occurrence matrices of keywords. Query logs reveal the association between two user queries. Advertiser logs are used to compare competitors lists of keywords. Using the associations and the co-occurrence matrices, Google’s Keyword Planner suggests new keywords [9]. Since query logs are human-generated, some of the terms can be ambiguous. Bartz et al. resolved this ambiguity by attaching non-ambiguous tags mined directly from such logs [13].

Chen et al. combined traditional query log mining with deep learning to generate new keywords [14]. Using query logs, they built two attention-based recurrent neural networks in order to model user behavior and suggest new keywords [15]. He et al. utilized query rewriting for creating variants of a seed keyword [16]. In their approach, an encoder-decoder architecture was used to learn the mapping between the original keyword and its variants. Li et al. argued that simple encoder-decoder architectures are not suitable for text generation [17]; Zhou et al. proposed a latent variable network to

alleviate the aforementioned drawback [18]. A generative adversarial network consisting of an encoder-decoder generator with a discriminator RNN was used in generating rare queries [19].

Unlike previous works in both keyword and ad generation, our proposed framework focuses more on language modeling. All of our proposed architectures learn patterns in the input text in a similar way: by treating the output as an adaptation of the input. In LSTM's and GRU's, the "shifted-by-one" mechanism allows the RNN's to encapsulate the patterns in the input text within their weights. By doing so, the RNN's can be used for generating many alternatives for some seed input at inference time. Likewise, both the generator and summarizer GPT2 models used are language models at heart but with different overhead. They treat text as the input and the output. In the summarizer GPT2, the task designator "TL;DR" is used to trigger different behavior or weights in the same language model. But, the input and output are the exactly the same text. We show that treating both the ad and keyword generation problems as language modeling problems yields results suitable for use in real ad campaigns.

3 FOUNDATIONS AND DATA DESCRIPTION

3.1 Ad-creative Dataset

There are four different ads datasets:

1. D_{rich} consists of 4795 instances. Each instance is a pair of landing page content and the corresponding ad.
2. D_{temp} consists of 3363 instances. In contrast to D_{rich} , the ads in D_{temp} are written using a small number of ad templates. An example for ads in D_{temp} is “online sql course . quality videos by domain experts . why wait ? learn sql now .”, where “sql” is easily replaceable in the template.
3. D_{rich}^* is a modification of D_{rich} where ads are rewritten by a domain expert in the Google Ads format as follows:

“Title #1 (30) | Title #2 (30) | Title #3 (30) . Description #1 (90) .
Description #2 (90)”

4. D_{rich}^+ is a slight modification of D_{rich}^* where the landing page title is also included in the landing page content collected.

Table 1 shows an instance from each dataset.

3.2 Keyword Dataset

The search campaign dataset contains 52K keywords in 260 campaigns. There is a row of data for each keyword, which includes match type, campaign and ad group identifiers, ad impressions, ad clicks, click-through rate (CTR), which is the rate of user clicks per ad impression, average cost-per-click, average position, advertisement cost, conversions, cost per converted click, click conversion rate, quality score, bounce rate, conversion value, and return on investment.

It is important to note that less obvious keywords with low competition may drive a significant share of total conversions. Among the keywords related to learning java online, the non-obvious keywords (those with 1 impression and 1 click) received 1,690 clicks and drove in 25 conversions, which corresponds to a conversion rate of 1.48%. The remaining ones had 6,416 clicks and 85 conversions for an aggregate conversion rate of

Table 1: Samples from all four datasets. The format is maintained in D_{rich}^* and D_{rich}^+ where the headlines pipe-separated (|) and the descriptions are dot-separated.

Dataset	Landing Page Text	Ad
D_{rich}	create your own ios 10 apps . apply for ios developer jobs . choose the best design pattern for your app . monetize your skills . upload your own ios apps to the app store	the complete ios swift + objec- tive c developer course . this course will teach you both the swift & objective c program- ming languages and how to build ios mobile apps
D_{temp}	beginner nikon digital slr (dslr) photography . beginner canon digital slr (dslr) photography . digital photography for begin- ners with dslr cameras .	the ultimate photography course for beginners . learn all the essentials of photog- raphy , and develop your skills to become an ultimate photographer yourself
D_{rich}^*	create your own ios apps . apply for ios developer jobs . choose the best design pattern for your app . monetize your skills . upload your own ios apps to the app store .	the complete ios swift objec- tive c developer course cre- ate ios apps . learn the swift and objective c programming . learn how to build ios mobile apps .
D_{rich}^+	create your own ios apps . apply for ios developer jobs . choose the best design pattern for your app . monetize your skills . upload your own ios apps to the app store . the complete ios swift + objective c developer course . this course will teach you both the swift and objective c programming languages and how to build ios mobile apps .	the complete ios swift objec- tive c developer course cre- ate ios apps . learn the swift and objective c programming . learn how to build ios mobile apps .

1.32%. In terms of price, the obvious keywords (those with more than 1 impression) were 12% more expensive than non-obvious ones. That is, non-obvious keywords are effective with lower cost per conversion.

In order to evaluate the generated keywords, we estimate their CTRs and compare them with the actual CTRs found in the dataset. Nearly half of the keywords had one impression and one click with a CTR of 100%, which is highly skewed. In order to handle the label imbalance in the training dataset, we compared the unlabeled data instances with the labeled ones, identified similar labels, and propagated those labels for 80% of the non-obvious keywords.

3.3 Pre-processing

Text is first decomposed into individual words called tokens. Each token is assigned a unique id. This mapping of unique tokens to ids constitutes the vocabulary of the dataset. While this methodology is suitable for most natural language processing (NLP) tasks, it could be enhanced by using named-entity recognition (NER) [20]. Some sequences of tokens, such as locations or organization names, are better considered as a singular token and not split up. By simple tokenization, the city name “Los Angeles” is split up into two independent tokens as “Los” and “Angeles”. The NER would capture this relation and produce a single token as expected.

3.4 Vectorization

Tokens do not encode any context. Related words such as “motel” and “hotel” would be treated as equally distant as any other word pair in the vocabulary. Distributed word representations, or embeddings, map all tokens into a vector space where each token is represented with a unique vector of a fixed dimension. The goal is to preserve the context of each word. Embeddings are initialized randomly where the values are uniformly distributed around a fixed interval centered at zero, or from a zero-mean normal distribution with a standard deviation varying from 0.001 to 10 [21]. The weights of these embeddings are trained alongside the model to improve the vector representation of the semantics of each token in the data. As the data size increases, tokens appear in more contexts and their meanings are reflected in their embeddings. Pre-trained embeddings on large datasets performed well in sentence classification and translation [22, 23]. A popular pre-trained set of vectors is Global Vectors (GloVe)

[24]. An improved method of training embeddings from scratch is fastText [25]. In our experiments, we used both GloVe and fastText.

3.4.1 GloVe

GloVe is trained on 2014 Wikipedia articles and a news dataset called English Gigaword 5 [24, 26]. It represents the context of each word in a co-occurrence matrix. The word co-occurrence in multiple contexts encodes the affiliation between word pairs. Establishing this association enables the model to learn the context of each word and therefore have a good understanding of its meaning and its similarity with other words. Each row in the co-occurrence matrix is a vector that represents the co-occurrence of a given word against all words, and encodes the context of that word. Since the co-occurrence matrix could have outliers, e.g. stop-words, each vector is normalized and log-smoothed.

3.4.2 fastText

fastText is different than GloVe such that it creates character-level n -grams [25]. In order to handle word containment, all words are padded with special start and end of word characters. Similar words will have multiple overlapping n -grams that indicate their semantic similarity. Those n -grams are aggregated to create each word's embedding.

4 Generative Neural Networks

In order to predict the next token from a given a sequence of tokens, a generative model estimates probabilities for all possible tokens. The estimates are higher for words that appeared more often in the same position previously.

The maximum likelihood estimation for a word given the $n-1$ previous words is computed as follows:

$$p(w_n | w_{n-1}, \dots, w_2, w_1) = \frac{\text{count}(w_1, w_2, \dots, w_n)}{\sum_w \text{count}(w_1, w_2, \dots, w_{n-1}, w)} \quad (4.1)$$

This is known as an n -gram model [27]. Neural language models use a continuous vector space for token representation. When a new token is added to the vocabulary, it is embedded in the vector space in order to accommodate its meaning. A recurrent neural network (RNN) processes text a token at a time in order to capture its meaning [15]. By feeding the output of the network as input again, it could capture temporal context. Although an RNN retains information about the previous tokens, it pays equal attention to all tokens. As the length of the sequence increases, an RNN may forget important information due to its limited memory. Long short-term memory network (LSTM) has a local memory for persisting important information [28]. It formulates what information to forget and what to retain at every time step. Gated recurrent unit (GRU) is a simpler RNN variant that changes the control mechanism of LSTMs [29]. With fewer number of parameters, it is faster to train but it encodes less context. Since keywords consist of a handful of tokens, GRU is a suitable model as well. With the attention mechanism, RNNs in encoder-decoder models learn which time steps to outweigh. The transformer proposed by Vaswani et al. uses a series of attention-based encoders and decoders to process text [30]. The transformer outperformed previous methods in many applications such as machine translation, text generation, and abstractive summarization [30, 31]. Contrary to RNNs, the transformer does not process text sequentially, and therefore can be parallelized on GPUs. RNN models were shown to work well in modeling consumer preference [32, 33]. In this work, we used LSTMs and GRUs for generating keywords, and a transformer called GPT-2 for generating both ads and keywords [34]. Figure 4.1 shows the pipelines for training and inference for the three proposed models.

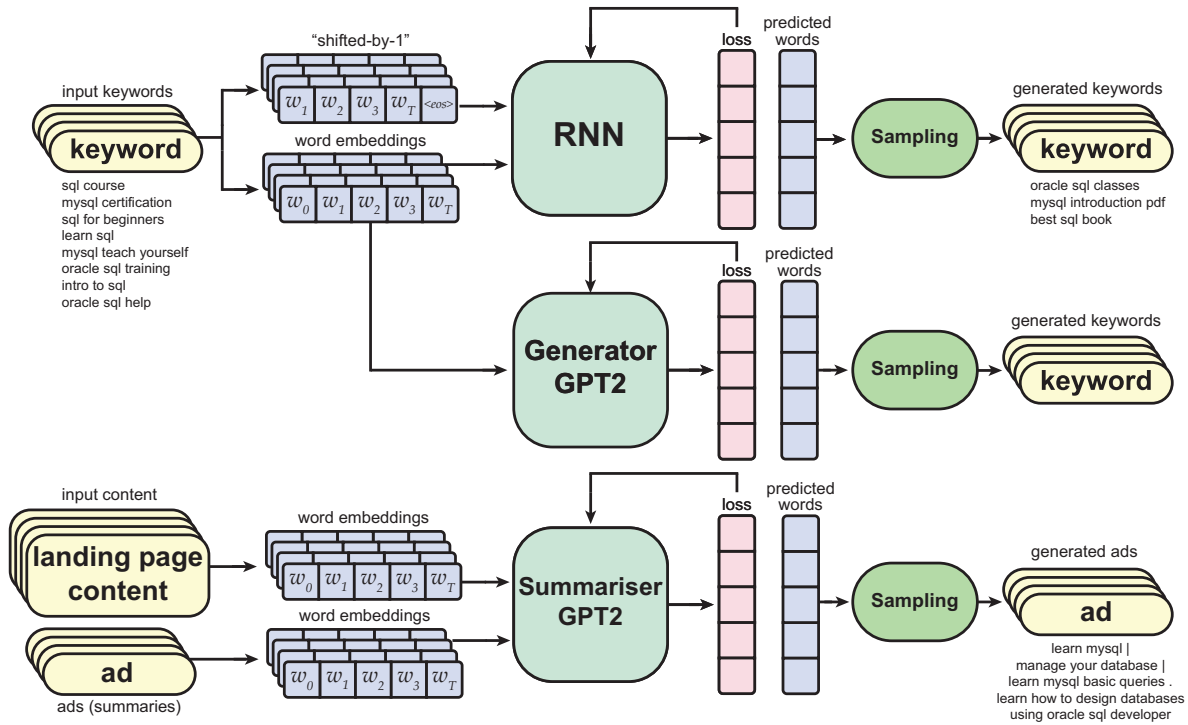


Figure 4.1: The training and inference pipelines used for keyword and ad generation with the proposed models.

4.1 RNN

We implemented LSTM and GRU using GeNN¹. Figure 4.2 depicts the basic units of these two networks. Next, we explain each of these networks in detail.

4.1.1 LSTM

The hidden and cell states are initialized to zeros. After each batch, back-propagation is performed to update the weight matrices W_f , W_c , and W_i of the forget gate, cell state, and input gate respectively. The functions *sigmoid* (σ) and *tanh* are activation functions that map input values to $[0, 1]$ and $[-1, 1]$ intervals respectively. At time step t , the hidden state of the previous time step $t-1$ is fed to the forget gate f_t along with the current input x_t . The forget gate is formulated as

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

¹See Appendix A on how to use GeNN.

and it controls what to forget from the previous hidden state given the current input. A candidate cell state c_t is then calculated as

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

which materializes the new memory to be stored according to the input and hidden state. The input gate controls what to store in the actual cell given the previous hidden state and input, and it is defined as

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

The output gate controls what to pass on to the next time step as the hidden state, and is formulated as

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

The cell state is the new memory of the current cell and it is based on the values provided by both the forget gate and the input gate:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

where \odot represents pointwise multiplication and \cdot is the dot product. Using the new cell state c_t and the output o_t , the hidden state is calculated as

$$h_t = o_t \odot \tanh(c_t)$$

and is passed on to the next time step. The ability to distinguish and decide what to write to the current cell and what to pass on to the next step is what allows an LSTM to retain important information in a long sequence.

4.1.2 GRU

The GRU reduces the number of gates in an LSTM to just two: a reset gate r and an update gate u [29]. The hidden state is initialized to zeros. Given the current input x_t at time step t , the update gate u controls what parts of the hidden states to preserve for future use and what to pass on to the next step. The update gate is formulated as

$$u_t = \sigma(W_u \cdot [h_{t-1}, x_t])$$

The reset gate controls which parts of the previous hidden states are not needed for the current step and what is important. It is formulated as

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

The candidate hidden state \tilde{h}_t hints at what to store in the current hidden state and is calculated as

$$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t] + b_c)$$

Finally, u_t will be used with the previous hidden state and the candidate hidden state to determine the current hidden state for passing on to the next step. It is calculated as

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t$$

4.1.3 Training generative RNNs

Our dataset contains D keywords $\{kw_0, kw_1, \dots, kw_D\}$ where each keyword is composed of tokens $\{w_0, w_1, \dots, w_T\}$ of variable length T . To train an RNN for next token prediction, the loss of the output at a given time step t should be minimized with respect to the output at time step $t - 1$. Therefore, we create “shifted-by-one” versions of all keywords as in $\{w_1, w_2, \dots, w_T, \langle eos \rangle\}$ where $\langle eos \rangle$ is the end-of-sentence token. Figure 4.3 shows the input and output at each time step of training an RNN. For a given keyword of length T , the loss is

$$L = -\frac{1}{T} \sum_{t=0}^T \sum_{j=1}^{|V|} P(w_{j,t+1}) \log \hat{P}(w_{j,t+1} | w_{j,t}) \quad (4.2)$$

where $P(w_{j,t+1})$ is the probability of the true token and $|V|$ is the vocabulary size. Back-propagation is performed for computing the average loss for a batch of keywords as follows

$$L_B = \frac{1}{B} \sum_{i=1}^B L_i \quad (4.3)$$

where B is the batch size. This methodology trains an RNN that could be used in predicting the next token.

4.2 Transformer

The GPT is based on the transformer proposed by Vaswani et al. [30], and uses self-attention decoders only. On the other hand, GPT-2 is trained using ten times more data and parameters [35]. A transformer is created using eight sequentially connected encoders and sequentially connected decoders. The input is fed to the first encoder and propagates through all eight encoders before being passed on to every decoder. Figure 4.4 illustrates the building blocks of a transformer.

4.2.1 Encoder

Each encoder is composed of a self-attention layer and a feed-forward neural network (FFNN) layer. The first encoder's self-attention layer receives all tokens in a sentence and builds attention vectors identifying the dependencies among the tokens. For the self-attention layer to encode the positions of all tokens, positional encoding vectors of the tokens are added to the embedding vectors. The output attention vectors are added to the original embeddings, which are passed on with a residual connection. The resulting values for each token are then normalized and passed on to a separate FFNN. The outputs of the FFNNs are added and normalized at the final step of the first encoder. These values are passed on to the next encoder through the final encoder in the same methodology. The output of the final encoder is a set of rich representation and attention vectors that assist the decoders in applying the objective transformation to the input.

4.2.2 Decoder

The decoder architecture is nearly identical to that of the encoder with an additional encoder-decoder attention layer. This layer is added between the self-attention and the FFNN layers and uses the output of the final encoder in order to guide the decoder in applying attention to the correct parts of the input sequence. To generate tokens, the decoder set operate in discrete time steps. In the first time step, the decoders use the encoded vectors to generate a token. In the remaining time steps, they use the previously generated token along with the encoded vectors in order to repeat the generation process until an end-of-sentence token is generated.

4.3 Sampling methods

At a given time step during generation, predicting the next token is not as simple as selecting the token with the highest score. Such a greedy approach in token selection forces the model to repeat itself [36]. The repetition is alleviated by introducing randomness in the selection process.

Top- k sampling is a method to introduce randomness [37]. At a given time step, instead of selecting the token with maximum likelihood, it selects a subset of the top candidates and samples one according to a new normalized probability distribution. Zhu et al. noted that for top- k sampling to truly match the quality of human text, large values of k are required [38]. However as k grows, low probability tokens could be selected especially when the perplexity of the model is low. We set k to 5.

Nucleus sampling adjusts the value of k according to the perplexity of the model [36]. When the sequence to complete is “online free course in ...”, the next token should be “java” or “javascript”. Since the perplexity is low, fewer choices in the candidate pool is better. For the sequence “how to ...”, the next token could be “write”, “learn”, or “code”. The perplexity is higher, and the model should pick from a richer pool.

4.4 Ad generation

We generate ads using a GPT-2 summarizer based on GPT-2 Small, which is composed of 12 decoder units. Its vocabulary size is around 50K, and has 117M parameters. All GPT-2 variants can be easily implemented for either generation or summarization using GeNN.

GPT-2 allows for in-context learning, which means that a model can be used in downstream tasks such as question answering, summarization, and translation simply by providing a task name [39]. Task names are prompts that describe the task and are provided alongside the input. GPT-2 has shown remarkable zero-shot or few-shot performance using prompts [40].

Likewise for fine-tuning, the user specifies the task name which helps the model identify the training objective. The task token for text summarization is the abbreviation “TL;DR”, which stands for “too long; didn’t read”. It is used in the following format to induce summarization:

“source document TL;DR: summary”

GPT-2 recognizes the mappings as input-output pairs for summarization and minimizes the loss accordingly.

Since ad generation is a text summarization task with explicit input-output pairs, we adhered to this format. We learnt a separate model for each of our ads datasets and reported its performance.

4.5 Keyword generation

We generate keywords using LSTM, GRU, and GPT-2. In LSTM and GRU, we follow the “shifted-by-one” methodology to train the RNNs. For GPT-2, we specify “keyword:” as the task token. This token is inserted at the beginning of all training instances that are fed to the model.

The main objective of language models is to predict the next token based on the previous text. Initially, a seed word has to be provided for the model to start predicting or generating text. Since keywords are generally short, we use only one seed. We select the initial seed by random weighted sampling. First, a frequency distribution is obtained using the first token of each keyword. Then, a seed is sampled from this distribution during prediction. This method results in a distribution of generated keywords that nearly matches that of the original, increasing the chances of generating non-obvious keywords. In an RNN model, the embedding for the random seed is fed to the model. In GPT-2, the random seed is concatenated with the task token “keyword:”.

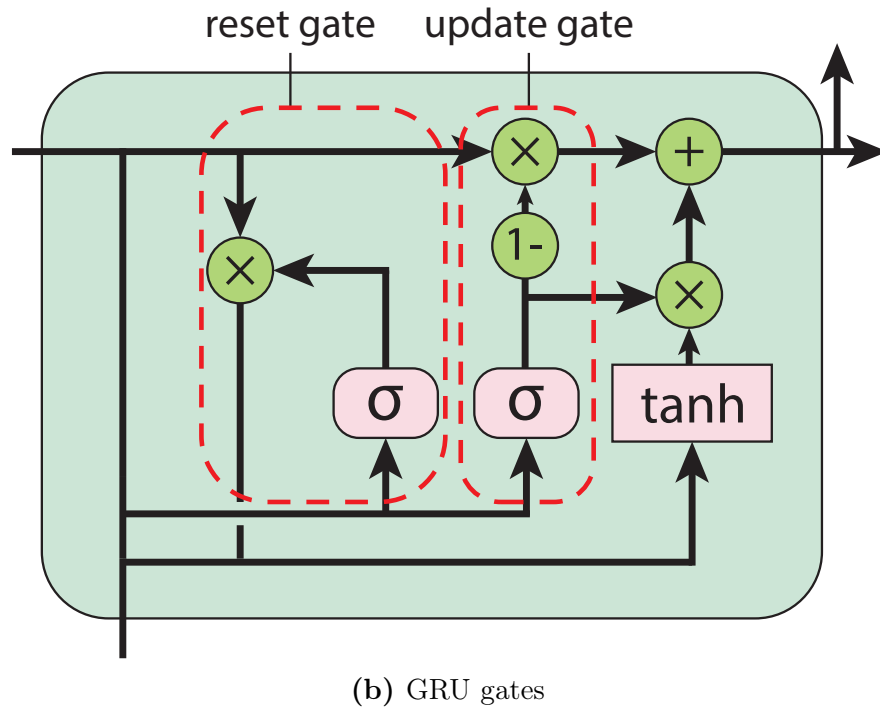
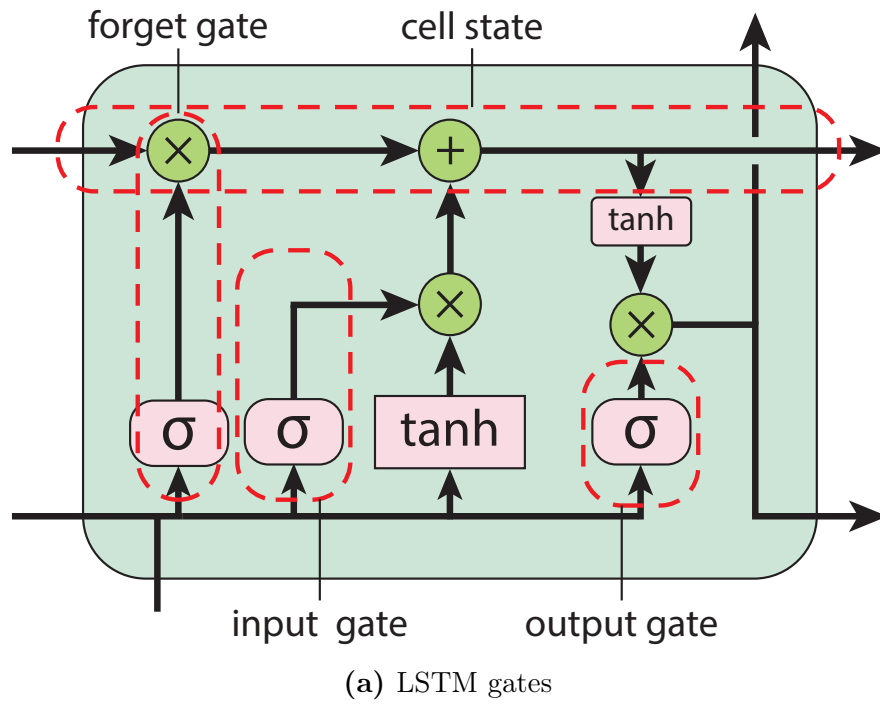


Figure 4.2: The basic building blocks of (a) LSTM networks and (b) GRU networks.

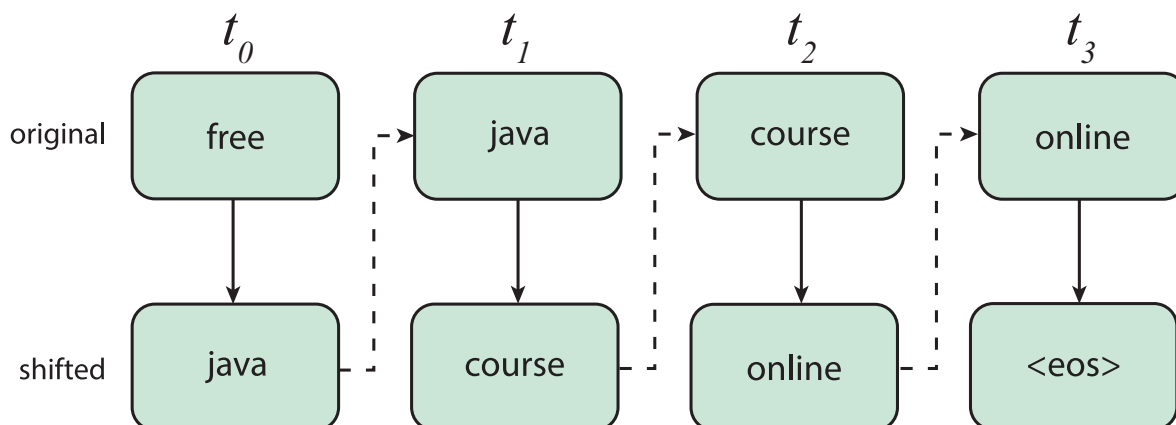


Figure 4.3: The RNNs are trained using shifted versions of the keywords as shown here for the keyword “free java course online”. At each time step, the keyword is shifted to the left.

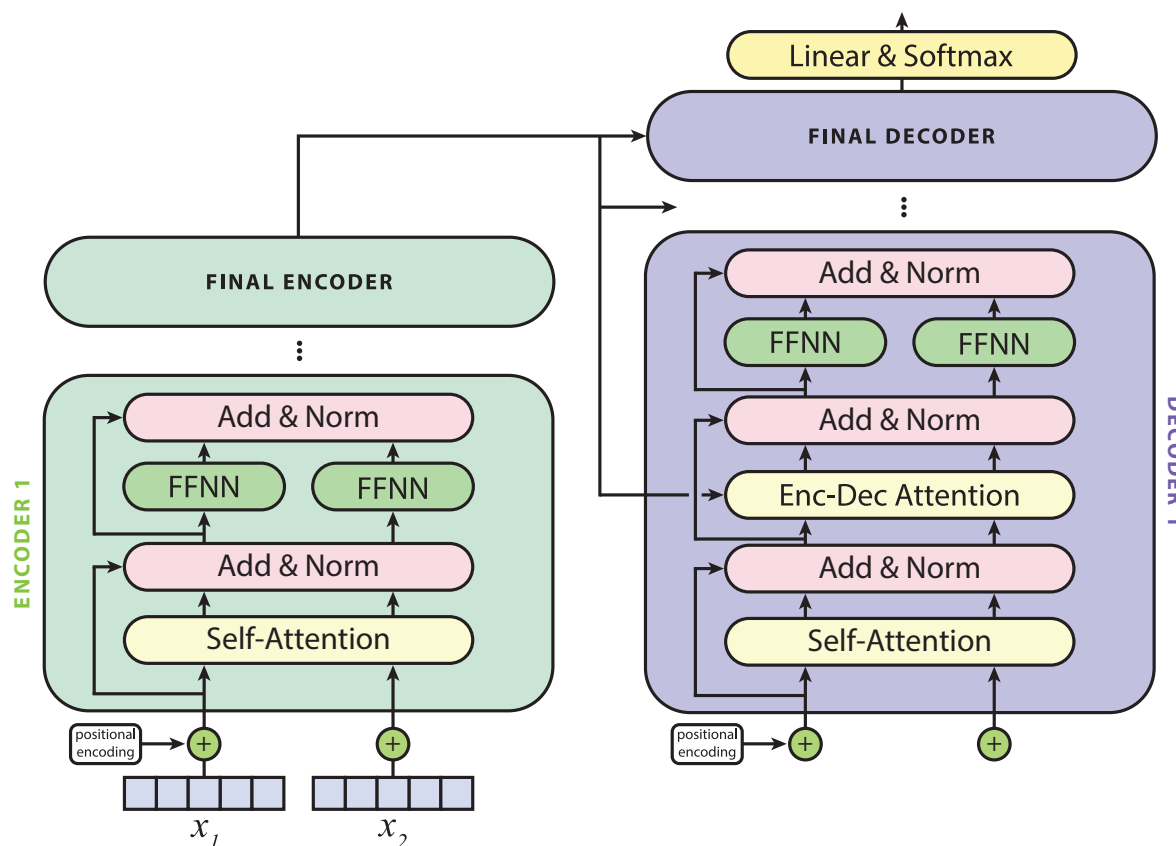


Figure 4.4: A transformer consists of eight sequentially connected encoders and eight sequentially connected decoders. Input is fed to the first encoder and is evicted from the last encoder. Then, it is fed to the encoder-decoder attention layer of every decoder.

Table 2: The ROUGE and BLEU scores of ad-creatives generated by GPT-2 after fine-tuning on each dataset. The addition of ad titles in D_{rich}^+ shows a significant improvement in the performance of the model.

Dataset	Sampling	ROUGE-1	ROUGE-2	ROUGE- L	BLEU
D_{rich}	Nucleus	21.4	4.8	21.0	7.3
	Top-5	21.7	5.0	21.7	6.3
D_{temp}	Nucleus	38.1	19.1	38.0	8.8
	Top-5	36.1	18.0	36.0	10.6
D_{rich}^*	Nucleus	31.1	10.8	31.3	5.0
	Top-5	31.1	10.2	31.9	5.9
D_{rich}^+	Nucleus	60	46.1	62	6.0
	Top-5	55.8	40.1	58.6	5.0

5 EXPERIMENTAL EVALUATION

We evaluated the quality of ads and keywords generated via GeNN both qualitatively and quantitatively in order to assert the practical value of GeNN for advertisers. The Bilingual Evaluation Understudy (BLEU) is used for evaluating the quality of translated text [41]. It was shown to reflect human evaluation for text quality [42, 43]. BLEU is defined as the fraction of n -grams from the generated text that also appear in the reference text. Hence, BLEU is a measure of precision. Another widely used metric for text evaluation is Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [44]. In contrast to BLEU, ROUGE represents the ratio of n -grams found in the original data that also appear in the generated text, and therefore, is a measure of recall. Specifically, ROUGE- L corresponds to the longest overlapping subsequence between the generated text and the original data. Together, BLEU and ROUGE quantify coherence and syntactic quality.

5.1 Ad-creative Evaluation

We tuned GPT-2 on each of the four datasets and reported the final ROUGE and BLEU scores. Since GPT-2 requires the length of the batch, i.e. the number of tokens, to be less than 1024, the proper batch size for our datasets is 4. As such, no one batch can exceed the fixed size of GPT-2. It is essential to properly set the batch size when using GeNN’s GPT-2 for generation or summarization. The results are reported in Table 2. We treated ROUGE as a measure of quality and BLEU as a measure of perplexity.

Table 3: The input landing page, ground-truth ad-creative, and generated ad-creative for a randomly selected instance from each test set. The generated text is grammatically correct, coherent, and meets the required length restrictions.

Dataset	Landing Page Text	Ground-truth Ad-creative	Generated Ad-creative
D_{rich}	write and compose music using proven techniques and tips . create their own music using software , or good old pencil and paper . program music and metaphors . character sketches . writing through canon . form . binary , ternary , and rondo forms . starting with a plan . graphic form representations	music composition bundle : composition film scoring 1 2 . learn music composition techniques through music composition lessons , music composition software , and techniques	music theory, character sketches, and rondo. learn how to use basic musical notation, create your own musical ideas, and much more!
D_{temp}	build responsive real world websites with html5 and css3 . html5 and css3 for beginners : create a website from scratch . learn html in 2 hours ! learn html 5 with hands on example step by step . a web development crash course in html5 and css3	learn html online . self paced html courses . starting from \$9 . signup today .	on demand html courses. the best price for the best value. learn at your own pace. start now!
D_{rich}^*	work in 3d space . create basic animation . create basic 3d models . work with materials and textures . basic lighting and render setups including image based lighting	3ds max mentor animation in 3ds max . guided bootcamp on the 3ds max basics from 3d modeling to animation . learn basic lighting and render setups including image based lighting .	the complete guide to 3d space and photoshop 3d scene for beginners. master basic scene creation and lighting, render with materials and textures.
D_{rich}^+	learn how to shoot in manual mode . learn how to get more / less depth of field . learn how to get more / less movement . learn how to talk to your camera , and how it talks to you . digital photography : shooting in manual . learn the skills , settings , and build confidence to shoot your dslr in manual mode for better and more creative photos .	digital photography learn shooting in manual . learn the skills , settings , and build confidence to shoot your dslr in manual mode for better photos .	digital photography shooting in manual. learn the skills, settings, and build confidence to shoot your dslr in manual mode for better and more creative photos.

GPT-2 suggested sequences that were never seen in the training data but were very common in the language itself especially when the perplexity was low. This was because the model was originally trained on a much larger dataset consisting of 8 million web pages. Imposing a structure on the data as in D_{rich}^* and D_{rich}^+ significantly improved the quality when compared to D_{rich} and decreased the model perplexity as indicated by the BLEU scores. D_{temp} 's scores are not comparable to the rest since the number of templates in the dataset is small. Thus, the model is encouraged by the data to avoid exploration and adhere to the templates. This explains the higher BLEU scores. Moreover, the addition of the ad titles in D_{rich}^+ enriched the source context and improved the performance significantly.

Table 3 shows the input, the ground-truth ad-creative, and the generated ad-creative side by side for a randomly selected landing page from each test set. The observed quality of the generated ads is encouraging for field use in real advertisement campaigns.

In the field experiment, we used an ad generated by our summarizer GPT2 in a real campaign for a large company in the US. The ad was run for two weeks and accumulated 500 impressions, a CTR of 5.89%, and 1 conversion.

For rapid experimentation, any generative model should be able to produce multiple alternatives for the same input. We tested the ability of our approach to generate different ad copies for the same landing page. GPT-2 achieved a high level of generalization on all datasets (except for D_{temp} where creativity is not a training objective). Out of 10 generated ad copies, we observed that the model was able to generate 8 different ad copies on average.

5.2 Keyword Evaluation

Table 4 shows the BLEU and ROUGE- L scores for the keywords generated. From these results, we can state that all models were able to generate *relevant* keywords. The best and worst keywords according to their expected CTRs are shown in Table 5. We observed that the top keywords were short and concise whereas the laggards were relatively longer and less intuitive.

Figure 5.1 shows the expected CTR of the generated keywords of varying length. For each keyword generated, we identify its near neighbors using locality-sensitive hashing (LSH) and estimate the field performance from the known CTRs of its neighbors. LSH is used for efficient nearest neighbor search in high dimensions [45]. The expected

Table 4: The ROUGE-*L* and BLEU scores for 300 generated keywords after five different training runs. The top-*k* and nucleus sampling performed similarly in all tests.

Generative Model	Embedding	Candidate Selection	ROUGE- <i>L</i>	BLEU
LSTM	GloVe	top- <i>k</i>	0.997	0.995
		nucleus	0.992	0.995
	fastText	top- <i>k</i>	0.995	0.999
		nucleus	0.992	0.999
GRU	GloVe	top- <i>k</i>	0.996	0.989
		nucleus	0.998	0.991
	fastText	top- <i>k</i>	0.994	0.989
		nucleus	0.998	0.989
GPT-2	-	top- <i>k</i>	0.994	0.995
		nucleus	0.993	0.999

CTR is calculated by dividing the sum of all clicks by the sum of all impressions of all nearest neighbors. We reported the average performance across five runs. In each individual run, we trained the respective model from scratch and generated a total of 300 keywords. While LSTM with fastText traced the true CTRs better compared to the other generative models, all models remained similar to the true CTR’s for keywords longer than two words in lengths. The similarity indicates that the models were able to replicate the patterns found in the original text in their generated text.

Table 6 shows the monthly performance forecasts according to Google’s keyword planner. GPT-2 generated keywords of much better quality with higher number of clicks as well as conversions. LSTM with fastText generated relevant keywords that closely resembled the existing keywords in the dataset. Hence, the generated keywords contributed less additive value over the clicks & conversions driven by the existing keywords. On the contrary, GPT-2 exploited the keyword space better and generated newer keywords with extra clicks and conversions, which was validated using Google’s own data. In summary, GPT-2 should be used for keyword exploration. LSTM could be used in generating ad copies where a certain level of consistency in wording is preferred for branding purposes.

Table 5: The best and worst keywords according to Google’s performance forecasts. The top keywords were shorter and concise whereas the laggards were longer and less intuitive.

Generative Model	Embedding	Generated Keyword	CTR Forecast by Google
LSTM	GloVe	best way to learn javascript advanced java free online courses	7.0% 0.0%
	fastText	core java course where to start java online free	14.2% 0.0%
GRU	GloVe	java programming language introduction to learn java free	13.6% 0.0%
	fastText	javascript free download how to learn basic java	21.0% 0.0%
GPT-2	-	how to learn java online introduce yourself to java	19.0% 0.0%

Table 6: The monthly performance forecasts by Google’s keyword planner. The keywords generated by GPT-2 are more valuable with higher number of clicks as well as conversions.

Generative Model	Clicks Forecast by Google	Conversions Forecast by Google
LSTM with fastText	10.34	1.00
GPT-2	720.13	55.00

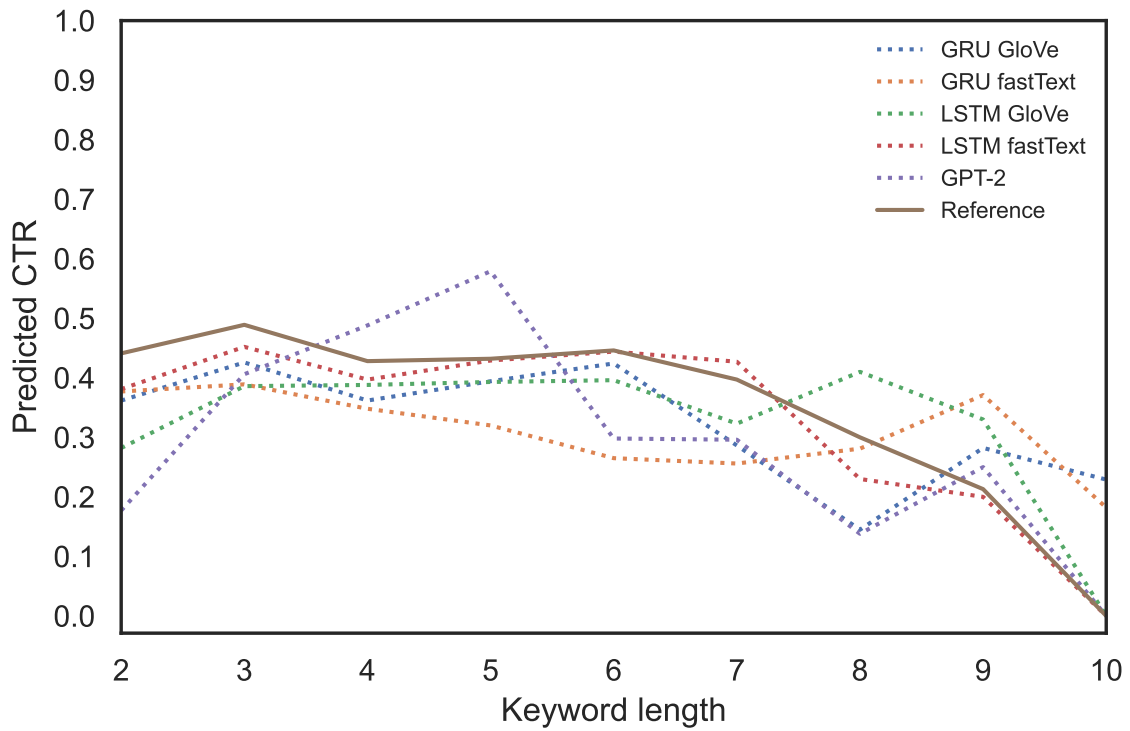


Figure 5.1: The expected CTRs of the generated keywords of varying length. The ground-truth data is shown as a solid line.

6 CONCLUSION

We introduced a framework called GeNN in order to generate keywords and ads for search advertising. The high BLEU and ROUGE-*L* scores of our models solidified the domain relevance of the text generated. The click-through rate extrapolation using LSH revealed that the keyword CTRs were close to the actual CTRs, and the field experiment for ads asserted the value of the ad generation in a real campaign. The field study done with Google’s keyword planner showed that the generated keywords have true market potential in driving actual user clicks and conversions. The generated ads were coherent and relevant, and they adhered to the Google Ads format.

Acknowledgments

This project is funded by Turkish National Science Foundation (Tübitak) under grant number 119E031.

Moreover, we would like to thank Kevser Nur Çoğalmış for granting us access to the ad-creative datasets.

7 REFERENCES

- [1] Cookhwan Kim et al. “How to select search keywords for online advertising depending on consumer involvement: An empirical investigation”. In: *Expert Systems with Applications* 39.1 (2012), pp. 594–610.
- [2] Ani Nenkova and Kathleen McKeown. “A Survey of Text Summarization Techniques”. In: *Mining Text Data*. Ed. by Charu C. Aggarwal and ChengXiang Zhai. Boston, MA: Springer US, 2012, pp. 43–76. ISBN: 978-1-4614-3223-4.
- [3] Rasim M. Alguliev et al. “MCMR: Maximum coverage and minimum redundant text summarization model”. In: *Expert Systems with Applications* 38.12 (2011), pp. 14514–14522. ISSN: 0957-4174.
- [4] Chirantana Mallick et al. “Graph-Based Text Summarization Using Modified TextRank”. In: *Soft Computing in Data Analytics*. Ed. by Janmenjoy Nayak et al. Singapore: Springer Singapore, 2019, pp. 137–146. ISBN: 978-981-13-0514-6.
- [5] Stamatina Thomaidou, Michalis Vazirgiannis, and Kyriakos Liakopoulos. “Toward an Integrated Framework for Automated Development and Optimization of Online Advertising Campaigns”. In: *CoRR* abs/1208.1187 (2012).
- [6] Stamatina Thomaidou et al. “Automated Snippet Generation for Online Advertising”. In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. CIKM '13. San Francisco, California, USA: Association for Computing Machinery, 2013, pp. 1841–1844. ISBN: 9781450322638.
- [7] J. Weston Hughes, Keng-hao Chang, and Ruofei Zhang. “Generating Better Search Engine Text Advertisements with Deep Reinforcement Learning”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2269–2277. ISBN: 9781450362016.
- [8] Amruta Joshi and Rajeev Motwani. “Keyword Generation for Search Engine Advertising”. In: *6th IEEE International Conference on Data Mining-Workshops*. 2006, pp. 490–496.
- [9] Vibhanshu Abhishek and Kartik Hosanagar. “Keyword Generation for Search Engine Advertising using Semantic Similarity between Terms”. In: *Proceedings of the 9th International Conference on Electronic Commerce*. 2007, pp. 89–94.
- [10] Hao Wu et al. “Advertising Keyword Generation using Active Learning”. In: *Proceedings of the 18th International Conference on World Wide Web (WWW)*. 2009, pp. 1095–1096.

- [11] Wordtracker. *Wordtracker*. 2020. URL: www.wordtracker.com.
- [12] WordStream. *WordStream*. 2020. URL: www.wordstream.com/keywords.
- [13] Kevin Bartz, Cory Barr, and Adil Aijaz. “Natural Language Generation for Sponsored-Search Advertisements”. In: Chicago, IL, USA: Association for Computing Machinery, 2008, pp. 1–9. ISBN: 9781605581699.
- [14] Wanyu Chen et al. “Attention-based Hierarchical Neural Query Suggestion”. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 2018, pp. 1093–1096.
- [15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning Representations by Back-propagating Errors”. In: *Nature* 323.6088 (1986), pp. 533–536.
- [16] Yunlong He et al. “Learning to Rewrite Queries”. In: *Proceedings of the 25th ACM International Conference on Information and Knowledge Management (CIKM)*. 2016, pp. 1443–1452.
- [17] Jiwei Li, Will Monroe, and Dan Jurafsky. “A Simple, Fast Diverse Decoding Algorithm for Neural Generation”. In: *arXiv e-prints*, arXiv:1611.08562 (Nov. 2016), arXiv:1611.08562. arXiv: 1611.08562.
- [18] Hao Zhou et al. “Domain-Constrained Advertising Keyword Generation”. In: *The 19th World Wide Web Conference (WWW)*. 2019, pp. 2448–2459.
- [19] Mu-Chu Lee, Bin Gao, and Ruofei Zhang. “Rare Query Expansion through Generative Adversarial Networks in Search Advertising”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 500–508.
- [20] Matthew Honnibal and Ines Montani. “spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing”. In: (2017).
- [21] Tom Kocmi and Ondřej Bojar. “An Exploration of Word Embedding Initialization in Deep-Learning Tasks”. In: *arXiv preprint arXiv:1711.09160* (2017).
- [22] Yoon Kim. “Convolutional Neural Networks for Sentence Classification”. In: *arXiv e-prints*, arXiv:1408.5882 (Aug. 2014), arXiv:1408.5882. arXiv: 1408.5882.
- [23] Ye Qi et al. “When and Why are Pre-trained Word Embeddings useful for Neural Machine Translation?” In: *arXiv preprint arXiv:1804.06323* (2018).
- [24] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

- [25] Piotr Bojanowski et al. “Enriching Word Vectors with Subword Information”. In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146.
- [26] Robert Parker et al. “English Gigaword 5th Edition”. In: *Linguistic Data Consortium, Philadelphia, PA, USA* (2011).
- [27] Claude E Shannon. “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27.3 (1948), pp. 379–423.
- [28] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.
- [29] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *arXiv e-prints*, arXiv:1406.1078 (June 2014), arXiv:1406.1078. arXiv: 1406.1078.
- [30] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2017, pp. 5998–6008.
- [31] Peter J Liu et al. “Generating Wikipedia by Summarizing Long Sequences”. In: *arXiv preprint arXiv:1801.10198* (2018).
- [32] James Chambua, Zhendong Niu, and Yifan Zhu. “User preferences prediction approach based on embedded deep summaries”. In: *Expert Systems with Applications* 132 (2019), pp. 87–98.
- [33] Dennis Koehn, Stefan Lessmann, and Markus Schaal. “Predicting online shopping behaviour from clickstream data using deep learning”. In: *Expert Systems with Applications* 150 (2020).
- [34] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: *OpenAI Blog* 1.8 (2019), p. 9.
- [35] Alec Radford et al. *Improving Language Understanding by Generative Pre-training*. 2018.
- [36] Ari Holtzman et al. “The Curious Case of Neural Text Degeneration”. In: *arXiv e-prints*, arXiv:1904.09751 (Apr. 2019), arXiv:1904.09751. arXiv: 1904.09751.
- [37] Ari Holtzman et al. “Learning to Write with Cooperative Discriminators”. In: *arXiv e-prints*, arXiv:1805.06087 (May 2018), arXiv:1805.06087. arXiv: 1805.06087.
- [38] Yaoming Zhu et al. “Texygen: A Benchmarking Platform for Text Generation Models”. In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 2018, pp. 1097–1100.

- [39] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2021.
- [40] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020.
- [41] Kishore Papineni et al. “BLEU: A Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL)*. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 311–318.
- [42] George Doddington. “Automatic Evaluation of Machine Translation Quality Using N-Gram Co-Occurrence Statistics”. In: *Proceedings of the 2nd International Conference on Human Language Technology Research*. HLT '02. San Diego, California, 2002, pp. 138–145.
- [43] Deborah Coughlin. “Correlating Automated and Human Assessments of Machine Translation Quality”. In: *Proceedings of MT Summit IX* (Jan. 2001).
- [44] Chin-Yew Lin. “ROUGE: A Package for Automatic Evaluation of Summaries”. In: *Text Summarization Branches Out*. Association for Computational Linguistics, July 2004, pp. 74–81.
- [45] Piotr Indyk and Rajeev Motwani. “Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality”. In: *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*. STOC '98. Dallas, Texas, USA, 1998, pp. 604–613. ISBN: 0897919629.

Appendices

A GeNN

For reproducibility, all methods mentioned in this work are wrapped into a Python package called **Generative Neural Networks**. GeNN is a high-level interface for our PyTorch implementations based on LSTM, GRU, and GPT-2. It is available on pypi.org/project/genn or via the PIP command `pip install genn`. The following code snippets illustrate the usage of GeNN.

To import the modules:

```
from genn import Preprocessing, LSTMGenerator, \
    GRUGenerator, GPT2, GPT2Summarizer
```

The module `Preprocessing` handles parsing files, tokenizing keywords, creating the random seed distribution, and creating shifted input-output pairs.

A.1 LSTM

```
# Do pre-processing
ds = Preprocessing("keywords.txt")

# Initialize an LSTM generator with default fastText
lstm = LSTMGenerator(ds, nLayers = 1,
                    batchSize = 16,
                    epochs = 10)

# Train the model
lstm.run()

# Generate 300 keywords
print(lstm.generate_document(300))
```

A.2 GRU

```
# Use the most frequent first token as seed
ds = Preprocessing("keywords.txt", seedParams={})

# Initialize a GRU generator with GloVe
gru = GRUGenerator(ds, nLayers = 1,
                   embeddingType = "glove",
                   glovePath = "glove.6B.50d",
                   batchSize = 16,
                   epochs = 10)

# Train the model
gru.run()

# Generate 300 keywords using top-3 selection
print(gru.generate_document(300,
                           selection = "topk",
                           k = 3))
```

A.3 GPT-2

```
# Initialize a GPT-2 Medium generator
gpt2 = GPT2("keywords.txt",
            taskToken = "keyword:",
            variant = "medium",
            epochs = 10)

# Train the model
gpt2.run()

# Generate 300 keywords using nucleus with p=0.8
print(gpt2.generate_document(300, p = 0.8))

# Initialize a GPT-2 summarizer.
summ = GPT2Summarizer("creatives.txt",
                      epochs = 10,
                      batch_size = 4)

# Train the model
summ.run()

# Generate 10 ad texts from a landing page
src_doc = "This is the landing page to summarize."
print(summ.summarize_document(n=10, source=src_doc))
```