MicroParade - Deep Contextual Meta-Embeddings for Microblog Retrieval

Liam Hebert
Dalhousie University
Halifax, NS
liam.hebert@dal.ca

Abstract

Information Retrieval when applied to microblog data often cannot benefit from explicit queries to determine what is relevant to show to a user. Instead, microblog retrieval systems attempt to model the target user's prior behaviours to determine what content is most pertinent to that user. However, one limitation of state of the art systems for microblog retrieval today is that they rely solely on metadata without considering the textual content of a post.

In this thesis, we propose MicroParade, a Transformer-based system that creates microblog recommendations by sampling and aggregating past interactions to a user. This approach is centered around utilizing Transformers to aggregate embeddings of past interactions to predict relevance. The textual content of these interactions are encoded by BERT and the whole model is fine-tuned end-to-end. We present details about the pre-processing pipeline used to enable this work, the model architecture, and experimental results.

1 Introduction

Information Retrieval can be manifested in many different forms. The most common of these forms is inputting an explicit search query and using that query to determine the relevance of documents. However, the luxury of an explicit query is not always available when determining the relevance of a document to a specific user. An example of this is determining the selection of relevant tweets that appear on the home screen of a user. In these cases, the behaviours of the user is the query to retrieve relevant documents.

The Twitter RecSys Challenge was created to encourage advancements in microblog retrieval where there is no explicit query (Belli et al., 2020). The challenge consists of a large dataset of over a billion tweet interactions and includes metadata about that interaction. The metadata included with the tweet includes information such as the time the tweet was posted, unique identifiers for the engager and engagee, whether external media is included and much more. The goal of this challenge is to predict whether a user will "like", "comment", "retweet" or "retweet with comment" a target tweet. Previous methods that have championed the leaderboard mostly rely solely on metadata features and disregard the actual textual content of the tweet (Schifferer et al., 2020). This leaves room for exploration as recent advancements in information retrieval methods focusing on text using Transformers have produced promising results (Lin et al., 2021).

Of particular interest is the use of a user's historic engagements, in particular the semantic meaning contained in the tweet text they engage with, to determine the relevancy of future tweets. Towards this, we present MicroParade, a system that seeks to incorporate advancements in Natural Language Processing, specifically Transformers (Vaswani et al., 2017), as part of a Information Retrieval system.

Preprint. Under review.

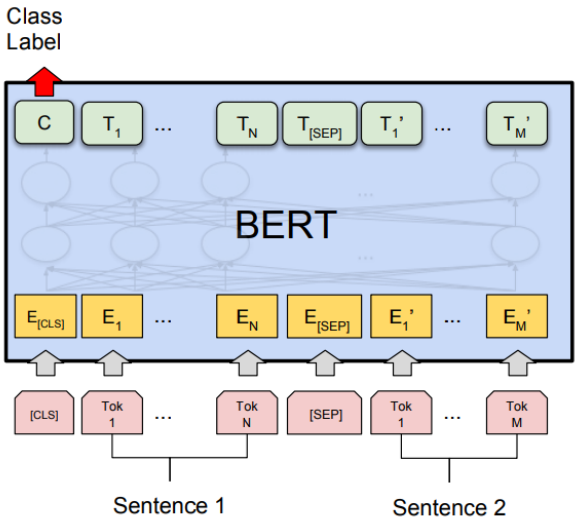


Figure 1: BERT architecture when applied to two sentences

We aim to model the behaviour of the user by aggregating multiple BERT embeddings of historic tweets with respect to a target tweet. This is done by sampling past interactions of the user as indications of the user's interests. These interactions are then aggregated through the use of a Transformer model to create a contextual meta-embeddings. We create this embedding by first cross-encoding the textual content of past interactions with the content of a prospective tweet, creating a set of embeddings. These embeddings are then aggregated with a Transformer model to create a single embedding representing the contextual relationship between the users interests and a prospective tweet. The final relevancy labels can be predicted by passing this embedding together with other feature engineered metadata into a feed-forward neural network. The model was fine-tuned end-to-end using TPUv3-8s.

Our main contributions involve creating a new architecture for including Transformers as part of a Recommender System, primarily through the use of aggregator Transformers. We also propose the addition of time embeddings to enrich embeddings with contextual information before aggregation. We conclude with presenting the results of this system, trained end-to-end, together with an ablation study.

In the following sections we will describe the methodology of MicroParade, the pre-processing pipeline required to process the Twitter RecSys 2021 dataset and the results obtained from our experimentation. We will conclude with prospective future work that can be undertaken to further advance this area.

2 Related Work

Core to this work is the use of BERT (Devlin et al., 2019) for information retrieval. BERT, described in Figure 1, is an extension of the Self-Attention Transformer architecture which consists purely of an encoder stack. This is different than the traditional Transformer architecture which consists of a encoder-decoder architecture (Vaswani et al., 2017). The result is a collection of dense embeddings for each token in the input, including an embedding for a `[CLS]` token representing the entire input sequence. BERT models are often pre-trained on large corpora, such as Wikipedia, and then further fine-tuned on the target dataset (Devlin et al., 2019).

The use of BERT Models for Information Retrieval was first proposed by Nogueira and Cho (2019). In this work, Nogueira and Cho (2019) estimated relevance by passing the embedding of the `[CLS]` token into a feed-forward neural network. Here, the input to the BERT model consisted of the query text, a separator token `[SEP]` and a candidate passage. As a result, the outputted `[CLS]` embedding encodes the semantic relationship between the query text and the candidate passage. The final outputted relevance score of the neural network would then be used to rank passages in terms of

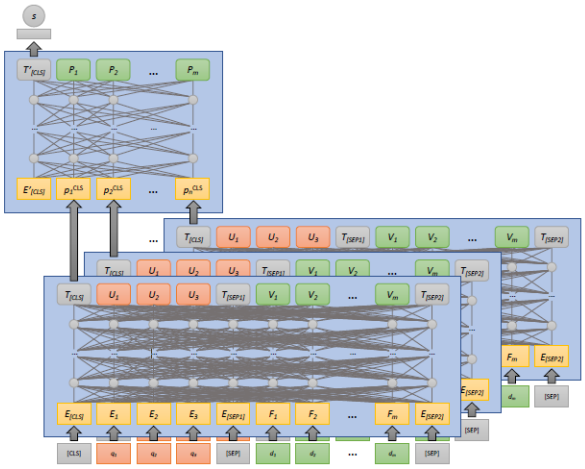


Figure 2: Architecture of the full PARADE model. The [CLS] embedding of each passage aggregated by a final transformer to predict relevance for the entire document
Figure retrieved from Lin et al. (2021)

relevance:

$$rel_i = BERT([[CLS], q, [SEP], p_i, [SEP]])$$

Where q is the query, p_i is the passage to be evaluated and rel_i is the predicted relevance. However, one limitation of BERT and other Masked Language Models is their limited input size, 512 tokens in the case of BERT. (Devlin et al., 2019). This can be particularly limiting for relevance inference on larger documents or passages that exceed this limit. To address this, Birch (Akkalyoncu Yilmaz et al., 2019) and BERT-MaxP (Dai and Callan, 2019) explored predicting sentence-level and passage-level relevance, respectively, to estimate the relevance of larger documents. The intuition of these methods is that strong relevance of smaller components towards a query can be used to extrapolate the relevance of the larger document.

The challenge of document-level relevance was further tackled by PARADE (Li et al., 2021). Rather then aggregate multiple relevance predictions of smaller components, PARADE aggregates embeddings through the use of BERT-like self-attention layers to estimate relevance, demonstrated in Figure 2. This resulted in better performance than prior models. The system functions by creating semantic embeddings of each passage together with a target query. Each embedding is then used as input to a Transformer model to create a final embedding which can be used to predict relevance. This differs from the standard use of Transformers, which typically process text tokens directly (Vaswani et al., 2017). The final outputted embedding is then used an output feed-forward neural network to predict relevance scores of the whole document with respect to the query.

While this system works well for document retrieval with fixed queries, it has yet to be applied in a recommender system, which does not have fixed queries. Additionally, the system does not make use of metadata features to predict relevance, relying strictly on the text. Lastly, the method was not applied to larger datasets such as the RecSys 2021 dataset, a focus for this work.

Turning our attention to state of the art approaches on the 2020 Twitter RecSys challenge, NVIDIA's winning RapidsAI solution heavily relied on feature engineering and utilized XGBoost for inference (Schifferer et al., 2020). Of the many features used in this system, Target Encoding (TE) is heavily attributed to its success. This function encodes categorical features with respect to target labels. This function is given as

$$TE_{target}([Category]) = \frac{count([Category]) * mean_{target}([Category]) + w_{smooth} * mean_{target}}{count([Category]) + w_{smooth}} \tag{1}$$

Where $count(\cdot)$ is the number of observations for that category and w_{smooth} is a smoothing parameter, typically set to 20. $[Category]$ is a list of categorical features to be encoded (Schifferer et al., 2020). The inspiration for this method is to encode catagorical features based on the mean value of the target

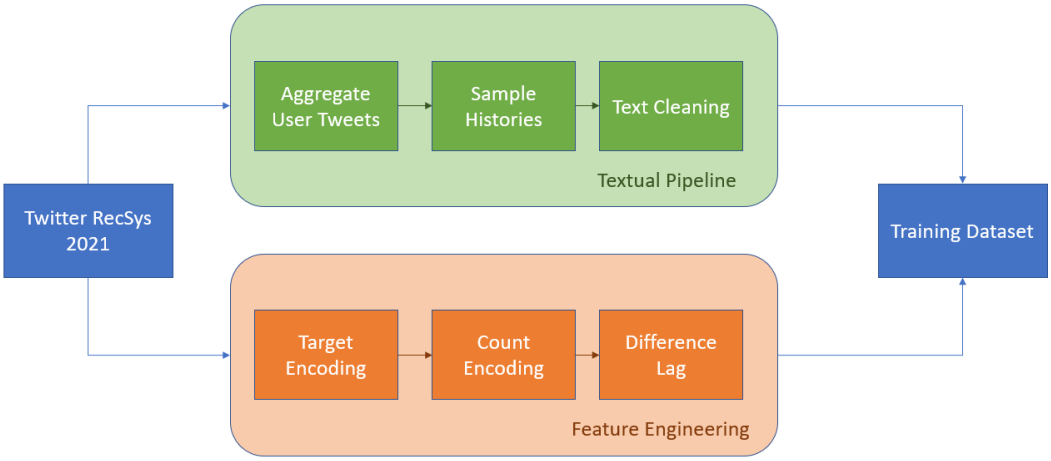


Figure 3: The two data preprocessing pipelines used to prepare the dataset

label for each possible categorical value. This method was used to encode each of the categorical features in the dataset towards each of the four engagement types. As part of their pre-processing pipeline, many features were engineered using GPU-enabled tools such as CuDF, Dask and CuPy. The final model consisted of an ensemble of XGBoost models (Chen and Guestrin, 2016), each trained on different subsets and used to predict one of the four labels. One limitation of this work is the lack of textual information used during inference, instead solely relying on metadata.

Another top performing model on the 2020 Twitter RecSys challenge was Layer6 AI's submission (Volkovs et al., 2020). This system utilized a BERT model to create embeddings of a user's prior engagements and of a candidate tweet prior to training the final model. These independently created embeddings are then aggregated via an attention mechanism to create an embedding representing the past engagements of the user. The resulting embedding is then used in a final feed-forward neural network in conjunction with many engineered tweet features to determine the relevance of a candidate tweet. This system performed very well during the RecSys2020 challenge, reaching second place on the public leaderboard. However, some limitations of this approach include the lack of end-to-end training, the use of attention mechanisms instead of transformers and some of the innovative features present in (Schifferer et al., 2020)

3 Methodology

In this section, we detail the MicroParade method for few-shot tweet retrieval. Given a set of tweets R that are known to be relevant to a user and a candidate tweet q, the method aims to predict $P(q|R)$ which is the probability of relevance of the candidate tweet given the known set of relevant tweets. Since a user's interaction history can often indicate the tweets that she is interested in, this is chosen as the initial set of known relevant tweets. To perform this task, multiple contextual embeddings created of our candidate tweet q appended to each tweet in R are created. Following this, the periodic time information of when the known relevant tweet was posted is encoded and added to each text embedding. The set of final embeddings are then appended with a `[CLS]` token embedding and aggregated using a Transformer Encoder stack. The resulting `[CLS]` embedding is then processed through a final feed-forward neural network to predict the relevance score of the candidate tweet.

3.1 Preprocessing Pipeline

Core to our training process is the pre-processing pipeline required to format the training data. The input data for our model can be divided into two sections: textual data and feature engineering. Regarding textual data, the primary goal of this pipeline involves creating history of engagements for each engaging user for use in MicroParade. Precise sampling of a user's history is employed when creating histories to sample tweets closest to the target tweet. This is required to increase the

relevancy of user's history and due to compute memory constraints. For feature engineering, we take inspiration from Schifferer et al. (2020) and employ Target Encoding to encode many features in the dataset using a GPU accelerated pipeline, together with other feature engineering enhancements.

For both of these pipelines, latency and memory management becomes a core issue due to the size of the Twitter RecSys 2021 dataset ($\approx$ 700 million tweets), especially to process the textual data (Belli et al., 2020). In order to complete this work within our compute and time resources, we focus on the validation set of the dataset ($\approx$ 14 million tweets). Additionally, we employ frameworks such as Dask[1], NVTabular[2], CuDF[3] and UCX[4] which allow us to parallelize and distribute the workload for these pipelines for use in a HPC environment, such as Compute Canada[5]. This allowed the pipeline to compute much faster and within memory constraints of available compute nodes. A full list of the final features used in the system and implementation details can be seen in Appendix section A.

3.1.1 Textual Preprocessing

As stated in the prior section, the purpose of the textual pipeline is to process the textual data present in the Twitter RecSys 2021 dataset (Belli et al., 2020). As it would be infeasible to include the entire user's history to predict a target tweet, we have to intelligently sample which tweets to include during inference. Towards this goal, we aggregate the tweets that each user engages with and sort them based on when they were engaged with. In addition, we prefix the type of engagement to each historic interaction was made (`like:` for like engagements, `RT:` for RT engagements, etc.). Then, for each target tweet in the dataset, we sample the total list of tweets that the user engages with to pick the tweets that are closest to that tweet. This is done in a sliding window over time where the size of the window is a hyperparameter to be tuned.

Due to the focus on the validation set of our dataset, we found that many users had limited histories in the dataset, often less than two tweets. Therefore, in order to address this, we prepend tweets from the training set for a given user to fill in the gaps in the sampling window. It is important to note that these extra tweets are not used in training, but only to fill in the sampled history. Lastly, as we use features about the timestamp of the post to create our time embeddings, we convert the available Unix timestamp to the hour of the day.

The last operation performed in the textual processing pipeline involves cleaning the tweet text. We perform common operations such as replacing full links with `[LINK]` and removing unnecessary tokens `RT`, `@` and `[CLS]`.

3.1.2 Feature Engineering

To process the rest of the metadata features, we take inspiration from Schifferer et al. (2020) and employ three types of feature engineering: Target Encoding, Difference Encoding and Count Encoding. We implement these features by using NVTabular.

Target Encoding, as described in Eq. 1, encodes categorical values with respect to target variables. In our case, our target variables are the four labels for each engagement type. In order to reduce overfitting in our target encoding, we utilize 5 fold validation (provided through NVTabular) to ensure our computed encodings are not biased to our training set. We apply target encoding to many categorical features in the dataset, such as `media`, `tweet_type`, `language` and many more features (see Appendix Table A). Count encoding is used to count the amount of `hashtags`, `domains` and `links` within the text. Lastly, Difference Encoding is used to compute the differences in followers and language between all the users.

Once this pipeline is finished, we merge the results from the Textual Pipeline in order to get our final dataset that we can use during inference. The resulting dataset consists of 63 features, 12 of which correspond to textual features to be used during BERT inference and 51 for our output feed-forward neural network.

[1] https://dask.org/

[2] https://github.com/NVIDIA/NVTabular

[3] https://github.com/rapidsai/cudf

[4] https://openucx.org/

[5] https://www.computecanada.ca/

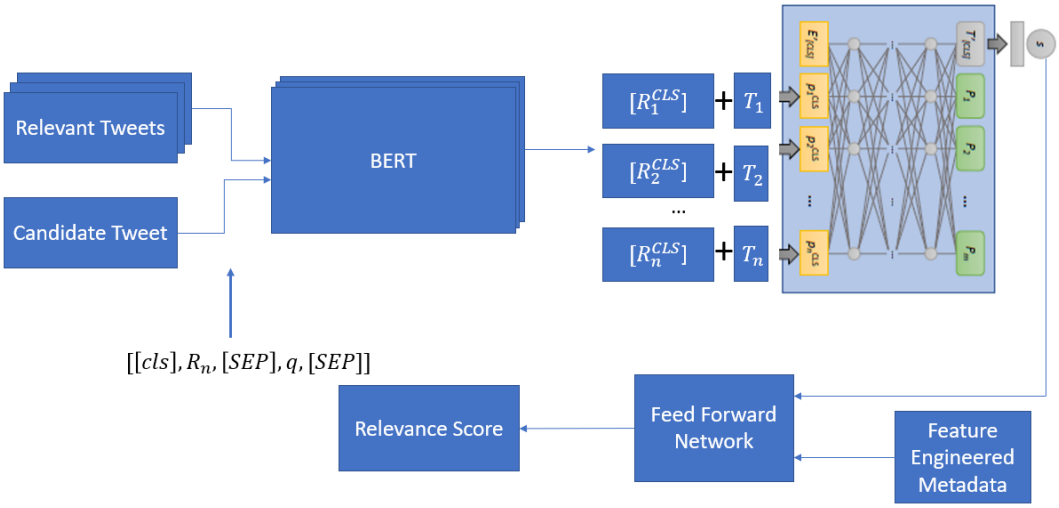


Figure 4: MicroParade Model Architecture

3.2 Model Infrastructure

The MicroParade infrastructure can be broken up into 4 distinct stages: Language Model embeddings, Time embeddings, Meta-embedding Transformer Encoder and finally, the output feed-forward network. The output of our model is a set of four sigmoid units, corresponding to the four possible engagement types, and trained using binary cross entropy loss. Additionally, in order to train the model, we make use of a custom data loader which we describe in this section.

3.2.1 Language Model

The first stage of the system consists of the joint embedding of candidate tweets and known relevant tweets. Adapting prior work by Nogueira and Cho (2019), the input to the language model consists of a candidate tweet q concatenated with a known tweet R_i with a `[SEP]` token appended between them and at the end. Additionally, the input is prepended with a standard `[CLS]` token. The embedding of the `[CLS]` token after this input is processed by the BERT model is chosen as the contextual representation of q according to a R_i, denoted as qR_i^{cls}. The result is a set of contextual embeddings C defined as

$$C = \{qR_1^{cls}, qR_2^{cls}, ..., qR_n^{cls}\}$$

for each $R_n \in R$. It is hypothesised that each of these embeddings will encode the relationship between a candidate tweet and a relevant tweet.

However, it is important to note the computational complexity of this step during inference since increasing the history size will increase the amount of BERT embeddings required to be produced. This can be particularly seen with large history sizes which require more accelerator memory. The BERT model used is accessed through the HuggingFace library (Wolf et al., 2020).

3.2.2 Time Embeddings

A common feature for inputs to Transformer Models is the use of a positional encoding(Vaswani et al., 2017). This is because transformer models inherently do not consider the position of inputs during inference. Therefore, by adding a specific embedding to each element according to the position of the element in the input, the model is encouraged to consider this information.

However, as the input to the meta-embedding layer is a group of embeddings rather than text, it becomes less clear how to apply positional embeddings. As periodic information of when a tweet was posted is known to be important to estimate relevance for a user, we substitute common positional representations with embeddings according to when a tweet was posted.

To do this, a learned embedding according to the periodic time that the reference tweet R_i was interacted with is added to each contextual embedding. This is represented as matrix of 24 trainable embeddings that correspond to the hour of the day. We denote T_i as the periodic embedding for reference tweet R_i. With the addition of time embeddings, our set of contextual embeddings C becomes

$$C = \{qR_1^{cls} + T_1, qR_2^{cls} + T_2..., qR_n^{cls} + T_n\}$$

where each T_i is added via addition to each embedding qR_i^{cls}. This follows prior work in Vaswani et al. (2017) where positional embeddings are added to each input token. As a result, the final representation of C is a series of embeddings that encode the relationship between a tweet and the user's past engagements with the addition of periodic information.

It is hypothesised that adding time information can help the model to understand when a user is interested in certain content. When looking at a dataset consisting of a larger time span, it would be possible to expand the set of possible embeddings to create more granular or more aggregated representations.

3.2.3 Meta-embeddings

After processing our contextual embeddings C, we aggregate them into a single meta-embedding through a series of self-attention Transformer encoder layers. Instead of using a `[CLS]` token to encode to semantic relevance of C, a embedding of the candidate tweet q is prepended to C for this purpose. As a result, the meta-embedding of C would then be the output embedding of q from the Transformer encoder layers, which we denote as C_q^{cls}.

As the size of encoded embedding is very large and dense, we utilize a feed-forward neural network to process C_q^{cls} into a more compact embedding, resulting in an outputted embedding from 768 to 192 in length. This is done to prevent the embedding from taking too much precedence in the final outputted neural network. Additionally, we employ Layer Normalization (Ba et al., 2016), ReLU activation and Dropout on each network layer to regulate our model.

3.2.4 Output Feed-Forward Neural Network

Once C_q^{cls} is computed, we bring in features we create in our feature engineering pipeline, resulting in an input vector of length 243. It is here that we compute our final predictions by passing this vector into a series of feed-forward layers until we reduce down to four outputs. Similar to the Meta-Embedding output neural network, we employ Layer Normalization (Ba et al., 2016), ReLU activation and Dropout on each network layer for regularization. These final four outputs are then processed by a Sigmoid function in order to obtain our prediction logits.

4 Results

To analyze the performance of MicroParade, we ran a number of experiments using the validation set of the Twitter RecSys 2021 challenge. Each experiment was done using a TPUv3-8 graciously provided through the TPU Research Cloud[6] with training data stored on Google Cloud Storage. The TPUv3-8 accelerator consists of 8 TPUv3 cores on a single chip that can be used for distributed training. For all experiments, the Adam optimizer is used with a learning rate of 0.00001 (Kingma and Ba, 2017). This learning rate and optimizer was chosen as it is a standard for many systems, including BERT (Devlin et al., 2019).

Our evaluation consists of an analysis of various history sizes, an ablation test of various elements of the model and a comparison between other models. The evaluation metric used is Average Precision as it is the official metric used by the Twitter RecSys 2021 challenge (Belli et al., 2020). This metric is given as

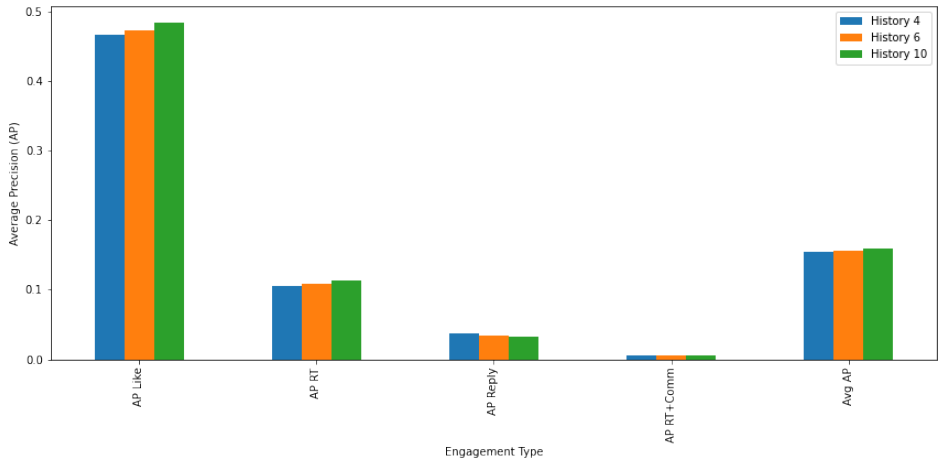
$$AP = \sum_n (R_n - R_{n-1})P_n$$

where P_n and R_n is the precision and recall at the nth threshold.

[6]https://sites.research.google/trc/

4.1 Impact of History Size

To analyze the impact of history size, we evaluated three sizes: 4, 6 and 10. Due to accelerator memory constraints, each model has a slightly different batch size.



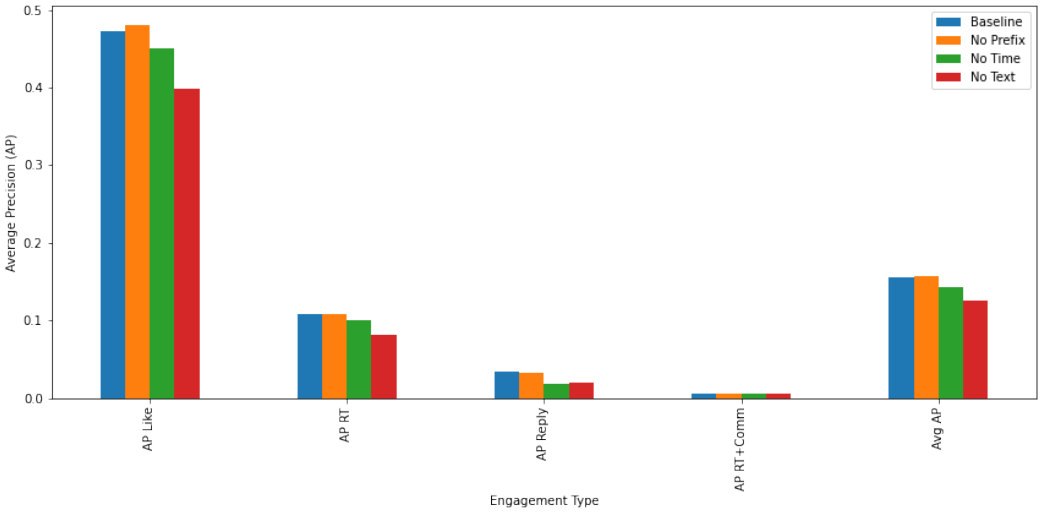
History	Batch Size	AP Like	AP RT	AP Reply	AP RT w/ Comment	Avg AP
4	96	0.4668	0.1054	**0.03713**	**0.0062**	0.1539
6	64	0.4733	0.1085	0.03461	0.0057	0.1555
10	40	**0.4843**	**0.1134**	0.03312	0.0056	**0.1590**

Table 1: Comparison of History Size Performance

We can see the results of our experiments using different history sizes in table 1. Using a history size of size 10 resulted in the best performance with an Average AP of 0.1590. Interestingly, we can see that our model improves performance for `Like` and `Retweet` with increased history size. However, performance for `Reply` and `Retweet w/ Comment` does not change. This can be attributed to the distribution of labels within the dataset where there is a more even distribution of labels for `Like` and and `Retweet`. Despite this, we conclude that performance is best for a history size of 10.

4.2 Ablation Study

In this section, we analyze the performance of adding engagement prefixes to the input history, adding time-embeddings and the effects of including textual data during inference. For each of these experiments, a history size of 6 with batch size of 64 is used.



Model	AP Like	AP RT	AP Reply	AP RT w/ Comment	Avg AP
Baseline	0.4733	**0.1085**	**0.0346**	0.0057	0.1555
No Prefix	**0.4811**	0.1075	0.0331	**0.0059**	**0.1569**
No Time	0.4502	0.0998	0.0190	0.0053	0.1435
No Text	0.3979	0.0809	0.0206	0.0056	0.1262

Table 2: Ablation Study

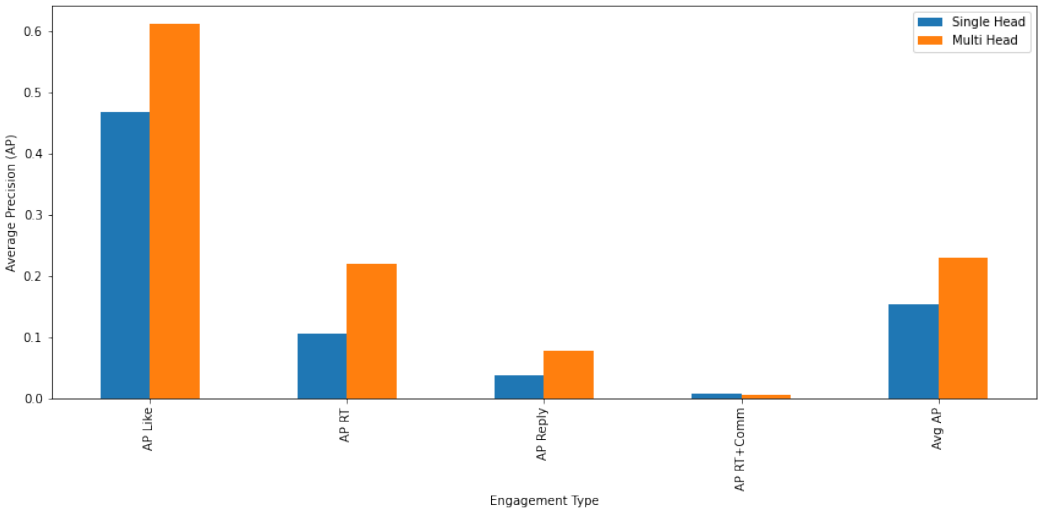
The results of this ablation study can be seen in Table 2. In our experiments, we found that removing input prefixes to the historic tweets can improve the performance in both the `Like` and `Retweet w/ Comment`. This is especially interesting as we hypothesized that adding information regarding how the history was engaged with would improve performance. Outside of removing the prefix, we can see that removing the time-embedding decreases performance by -0.03AP for the `Like` engagement and -0.0135AP on average. This demonstrates that adding time embedding does result in better performance. Lastly, as expected we see that performance suffers greatly when removing textual features. This can be seen in the large decrease in Average Precision, dropping by 0.0293 points.

As such, we can conclude that the addition of prefixes to the model's history can be detrimental to the model's performance. This can potentially be attributed to BERT's pretrained nature, where the input text it was trained on does not have these prefixes.

4.3 Comparison of output neural networks

The last experiment measured compares the performance of utilizing a single output neural network for each output type against utilizing a set of neural networks to predict each engagement type independently. This is inspired by Schifferer et al. (2020) which found better performance by training an XGBoost model for each engagement type separately.

We implement multi-head output by duplicating the meta-embedding output neural network and final output neural network for each engagement type. This results in four output neural networks for which the output of each is concatenated together to form the final prediction of the model. Due to compute constraints, both experiments below are conducted with a history size of 4.



Model	AP Like	AP RT	AP Reply	AP RT w/ Comment	Avg AP
Single Head	0.4668	0.1054	0.03713	**0.0062**	0.1539
Multi Head	**0.6116**	**0.2203**	**0.07764**	0.0053	**0.2287**

Table 3: Comparison of Output Heads

The results of our experiments can be seen in Table 3. We can see that performance is drastically improved across all engagement types by including separate output networks rather than using a single network. The results from using multiple output neural networks bests the results from previous experiments as well, despite using a limited history size. The largest increase can be seen in `Retweet` and `Reply` performance, resulting in over double the performance compared to using a single output head.

The results from this experiment seem to suggest that embeddings and feature importance can dramatically change between engagement types. Additionally, it can be hypothesized that further performance could be gained by training the model with a larger history size with this output architecture.

5 Future Work

There are many potential directions to evolve this work further. One major limitation of this project is the latency and memory requirements of large masked language models such as BERT. With the advent of smaller, distillation based models such as DistilBert (Sanh et al., 2020) and ELECTRA (Clark et al., 2020), it is possible that these memory constraints could be alleviated. Additionally, another avenue would be to fine-tune the language model independently before including it to the rest of the model. This would allow us to train MicroParade without performing gradient updates on the language model, greatly reducing memory constraints.

Due to compute constraints, the experiments that were performed had to be done using just the validation set. To allow this work to be compared to other systems that tackled the same dataset, it would be beneficial to examine the performance of MicroParade using the available test dataset. Additionally, to appropriately evaluate the impact of different model configurations, a statistical significance test should be performed on the predictions of each model. For this, we recommend utilizing an unpaired T-test between the baseline model and alternative options for each category of experimentation done. It is predicted that computing such a t-test should take ≈ 4 hours of compute time on TPUv3-8's per model configuration.

Lastly, it may be interesting to refocus the Twitter RecSys 2021 task to instead focus on retrieving relevant tweets without regard to engagement types. Under this simpler task, the class imbalance of

the original dataset would be resolved as all labels would be treated the same. This refocus could also allow future work under an exploratory analytics context, where the user or researcher could manually select tweets that they are interested in as a replacement for the history input. This could result in progressively more relevant results as the history is expanded, especially in situations where the user does not explicitly know what they are searching for.

6 Conclusions

Despite the many challenges that come with large datasets and language models, we were still able to successfully design and fine-tune a new architecture for recommender systems using textual data. The system we created, MicroParade, functions by utilizing the past interactions of a user to create a set of contextual embeddings of a target tweet we want to estimate relevance for. We then enrich these embeddings with the addition of novel learned periodic embeddings regarding the day a given tweet was interacted with. Finally, these embeddings are then aggregated through a Transformer encoder stack to create a meta-embedding that which is then fed into a final feed-forward neural network together with engineered features to predict relevance.

We believe that there is still much room to explore in regards to introducing Transformer models to recommender systems, especially with many systems forgoing using textual data at all. As we have shown in our results, considerable gains can be achieved by including a users textual history as part of the inference process. With further developments with masked language models, it is possible that these advancements will only get better and better.

Acknowledgments and Disclosure of Funding

Research supported with Cloud TPUs from Google's TPU Research Cloud (TRC). This research was also enabled in part by support provided by ACENET (https://www.ace-net.ca/) and Compute Canada (www.computecanada.ca).

I would also like to thank my friends, family, dogs and supervisors for supporting me throughout the year long research that created this work. This work would not have been possible without each and every one of them.

7 Appendix

A Compute environment used in Preprocessing

Memory and compute time was a large constraint in our preprocessing pipelines. In order to mitigate the large memory cost of manipulating and aggregating text data, we employed a distributed Dask cluster over 8 compute nodes with UCX networking over Infiniband. In addition, as many parts of the pipeline are able to be processed in parallel, utilizing Dask allowed the pipeline to be accelerated and memory properly distributed.

Similar to our text processing pipeline, we also make use of Dask to accelerate the various feature engineering steps performed. As the majority of operations in the feature engineering pipeline are purely numerical, we are able to enhance these operations using GPU's through the use of NVTabular and CuDF. These libraries accelerate common Dask operations by substituting them with GPU accelerated operations. Utilizing these libraries, we perform the operations in this pipeline using two Nvidia V100 GPUs.

Here, we describe each of the features used in our system, the pipeline it originated from, and its description.

Feature name	Feature Type	Feature Description
reply_data	Target Label	Reply label
retweet_data	Target Label	Retweet label
retweet_with_comment_data	Target Label	Retweet with Comment label
like_timestamp_data	Target Label	Like of target tweet
text_tokens_data	Textual	Text tokens of a target tweet
text_tokens_hist	Textual	Text tokens of each historic tweet
reply_hist	Textual	Reply label for each historic tweet
like_hist	Textual	Like label for each historic tweet
retweet_hist	Textual	RT label for each historic tweet
retweet_with_comment_hist	Textual	RT w/ Comment label for each historic tweet
interaction_time_hist	Textual	The time of day for each historic tweet
mask_hist	Textual	Mask for the input of the Transformer
media	Metadata	If the tweet contains media or not
tweet_type	Metadata	The tweet type (RT, reply, etc.)
language	Metadata	The language of the post
a_follower_count	Metadata	The amount of followers the engager has
a_following_count	Metadata	How many users the engager is following
a_is_verified	Metadata	If the engager is Twitter verified
b_follower_count	Metadata	The amount of followers the engagee has
b_following_count	Metadata	How many users the engagee is following
b_is_verified	Metadata	If the engagee is Twitter verified
b_follows_a	Metadata	If the engagee follows the engager
hashtags_count_t	Metadata	The amount of hashtags in the post
domains_count_t	Metadata	The amount of domains in the post
links_count_t	Metadata	The amount of links in the post
media_count	Metadata	Media types converted to categorical values
tweet_type_count	Metadata	Tweet types converted to categorical values
language_count	Metadata	Language converted to categorical values
a_user_id_count	Metadata	Engagee's user id converted to categorical
b_user_id_count	Metadata	Engager's user id converted to categorical
TE_media_* (4x)	Metadata	Target encoding of media per target*
TE_tweet_type_* (4x)	Metadata	Target encoding of tweet type per target*
TE_language_* (4x)	Metadata	Target encoding of language per target*
TE_a_user_id_* (4x)	Metadata	Target encoding of engagee id per target*
TE_b_user_id_* (4x)	Metadata	Target encoding of engager ud per target*
time_to_datetime_to_* (3x)	Metadata	Timestamp to hours, minutes and seconds*
b_follower_count_DiffLag	Metadata	Difference lag of engagee follower counts
b_following_count_DiffLag	Metadata	Difference lag of engagee following counts
language_DiffLag	Metadata	Difference in language between posts

B Hot Swap Distributed Dataloader

One major challenge of the Twitter RecSys 2021 dataset is its size, measuring over 19GB in compressed form for the validation set (Belli et al., 2020). This becomes very limiting in distributed machine learning environments, such as TPUv3-8's, where each compute core requires a copy of the entire training data. With our target environment, this meant utilizing over 152GB of memory in addition to memory usage by other aspects of the model. While this may be suitable for some environments, it would not scale for the full RecSys 2021 training set, which measures over 500GB after pre-processing (requiring 4TB of memory during distributed training).

To mitigate this challenge, we employ Dask to fragment the larger dataset into smaller chunks of ≈ 180MB in size (≈ 63000 training examples) in Apache Parquet[7] format. This compressed form allows for a smaller disk footprint while also allowing for very fast loading. To support distributed training, each parquet file is further segmented into unique partitions for each compute core when the file is loaded.

[7] https://parquet.apache.org/

During training, the dataloader created hot swaps Parquet files when they become fully exhausted by loading them from Cloud Storage. The result is a overall system memory footprint that rarely crosses 60GB, with each hot swap occurring with minimal overhead.

References

Zeynep Akkalyoncu Yilmaz, Shengjin Wang, Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Applying BERT to Document Retrieval with Birch. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*. Association for Computational Linguistics, Hong Kong, China, 19–24. https://doi.org/10.18653/v1/D19-3004

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. arXiv:1607.06450 [stat.ML]

Luca Belli, Sofia Ira Ktena, Alykhan Tejani, Alexandre Lung-Yut-Fon, Frank Portman, Xiao Zhu, Yuanpu Xie, Akshay Gupta, Michael Bronstein, Amra Delić, Gabriele Sottocornola, Walter Anelli, Nazareno Andrade, Jessie Smith, and Wenzhe Shi. 2020. Privacy-Aware Recommender Systems Challenge on Twitter's Home Timeline. arXiv:2004.13715 [cs.SI]

Tianqi Chen and Carlos Guestrin. 2016. XGBoost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Aug 2016). https://doi.org/10.1145/2939672.2939785

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. arXiv:2003.10555 [cs.CL]

Zhuyun Dai and Jamie Callan. 2019. Deeper Text Understanding for IR with Contextual Neural Language Modeling. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Jul 2019). https://doi.org/10.1145/3331184.3331303

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

Canjia Li, Andrew Yates, Sean MacAvaney, Ben He, and Yingfei Sun. 2021. PARADE: Passage Representation Aggregation for Document Reranking. arXiv:2008.09093 [cs.IR]

Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2021. Pretrained Transformers for Text Ranking: BERT and Beyond. arXiv:2010.06467 [cs.IR]

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. arXiv:1901.04085 [cs.IR]

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv:1910.01108 [cs.CL]

Benedikt Schifferer, Gilberto Titericz, Chris Deotte, Christof Henkel, Kazuki Onodera, Jiwei Liu, Bojan Tunguz, Even Oldridge, Gabriel De Souza Pereira Moreira, and Ahmet Erdem. 2020. GPU Accelerated Feature Engineering and Training for Recommender Systems. In *Proceedings of the Recommender Systems Challenge 2020* (Virtual Event, Brazil) *(RecSysChallenge '20)*. Association for Computing Machinery, New York, NY, USA, 16–23. https://doi.org/10.1145/3415959.3415996

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

Maksims Volkovs, Zhaoyue Cheng, Mathieu Ravaut, Hojin Yang, Kevin Shen, Jin Peng Zhou, Anson Wong, Saba Zuberi, Ivan Zhang, Nick Frosst, Helen Ngo, Carol Chen, Bharat Venkitesh, Stephen Gou, and Aidan N. Gomez. 2020. Predicting Twitter Engagement With Deep Language Models. In *Proceedings of the Recommender Systems Challenge 2020* (Virtual Event, Brazil) *(RecSysChallenge '20)*. Association for Computing Machinery, New York, NY, USA, 38–43. https://doi.org/10.1145/3415959.3416000

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6