GPU Accelerated Boosted Trees and Deep Neural Networks for Better Recommender Systems

CHRIS DEOTTE*, NVIDIA, United States
BO LIU*, NVIDIA, United States
BENEDIKT SCHIFFERER*, NVIDIA, United States
GILBERTO TITERICZ*, NVIDIA, Brazil

In this paper we present our 1st place solution of the ACM RecSys 2021 challenge. Twitter provided a dataset of around 1 billion tweets-user pairs to develop models predicting user engagements. The challenge simulates a realistic production environment by introducing latency constraints of an average of 6ms per example on a single CPU core with 64GB memory. Our solution ranked first having the highest score in each of the eight performance metrics to calculate the final position. Our final submission is an ensemble of stacked models, using in total 5 XGBoost models and 3 neural networks. Although ensembles are rarely used in production environments, we demonstrate their superiority and feasibility to inference the test dataset given the time limitation and computational resources. As an alternative to retraining, continuous training or fine-tuning models when new data is available, we stacked trained models to leverage additional data. Stacking requires less data to calibrate the trained models. Finally, we analyze the benefits of a GPU-accelerated production environment. Using open source libraries, such as Forrest Inference Library, NVTabular, RAPIDS cuDF, PyTorch and TensorFlow, we are able to accelerate our final solution by 257x from 23 hours and 40 minutes to 5.5 minutes and reduce total cost by 88% using a single NVIDIA A100 GPU with 40GB memory, enabling opportunities for significant cost savings and larger models with higher accuracies. We published our solution on github.com[1]

CCS Concepts: • **Information systems → Personalization**.

Additional Key Words and Phrases: Recommender Systems, RecSys Challenge, Boosting, Deep Learning, Neural Networks, Stacking, Continuous Learning, Feature Engineering, GPU Acceleration

ACM Reference Format:
Chris Deotte, Bo Liu, Benedikt Schifferer, and Gilberto Titericz. 2021. GPU Accelerated Boosted Trees and Deep Neural Networks for Better Recommender Systems. In . ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The annual ACM Recsys challenge was an especially interesting one in 2021 as it was once again hosted by Twitter, making the open source solutions from last year an excellent starting point. As in the 2020 competition, the goal is to predict user engagement given a tweet-user combination [2]. The new aspects introduced this year were the larger dataset with 1 billion tweet-user pairs, more interactions per user, an evaluation for fair recommendation and a runtime limitation given fixed hardware resources for predicting the test dataset to provide a more realistic environment [17]. Last year, we won the competition by leveraging GPU accelerated feature engineering and XGBoost models [3],

*Authors contributed equally to this research.
[1]https://github.com/NVIDIA-Merlin/competitions/tree/main/RecSys2021_Challenge

reducing the runtime of our end-to-end pipeline from 9 hours to 2 minutes [16]. This year an accelerated pipeline for training and quick offline experiments was a key component as well, and can be reviewed in our previous paper [16] and our GitHub repositories [11] [12]. In particular, our challenges were to improve our previous year solutions and train models which can predict the validation dataset in the given compute and time constraints on CPU. Our winning solution was a stacked ensemble of in total 5 XGBoost models and 3 neural networks as visulized in Figure 1. Our contributions are as follows:

- We provide an analysis comparing the effectiveness of our ensembling approaches of diverse models with the individual model.
- We propose a calibration of RecSys models on newer and smaller datasets by stacking them instead of retraining, fine tuning or continuous learning, which are often used in production environments.
- We demonstrate how to build very effective ensembling and stacking pipelines that can be used in real-world production environments, with runtime under 6ms per example on a single core CPU.
- We demonstrate how the NVIDIA A100 GPU can be used to accelerate the inference prediction of our complex ensembling pipeline, by reducing the runtime from 8 hours and 24 minutes on 8 CPU cores down to 5 minutes and 30 seconds on single GPU, a speed-up of 92x. At the same time, the approximate dollar costs are reduced by 8x-9x when running on cloud.

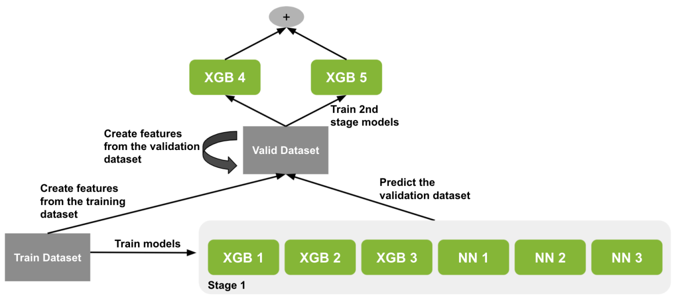
+
XGB 4
XGB 5
Train 2nd stage models
Create features from the validation dataset
Valid Dataset
Create features from the training dataset
Predict the validation dataset
Train Dataset
Train models
XGB 1
XGB 2
XGB 3
NN 1
NN 2
NN 3
Stage 1

Fig. 1. Final solution of creating features from the training dataset, stacking multiple stage 1 models and creating features from the validation dataset.

2 RECSYS 2021 CHALLENGE

The RecSys 2021 challenge follows a similar setup to the previous year. The dataset has the same structure and the goal is to predict the same user engagements: does a user reply, retweet, quote or like a tweet? This year the performance metrics reflect fair recommendations with respect to users' popularity. Five user groups were defined according to authors' follower counts and two metrics were computed individually for each user group: Average Precision (AP) and RCE (Relative-Cross-Entropy). Those metrics are averaged over the five user groups to generate the final score.

Twitter provided 1 billion tweet-user pairs over 4 consecutive weeks. The dataset was sampled so that more interactions are available by engaged with users (author) or engaging users (reader), as visualized in Figure 3 in Appendix A.

The training dataset comprised the first three weeks. The fourth week was randomly split into a validation and test dataset, with the special constraint that each author and each reader had at least one interaction in the training dataset.

The competition added constraints on runtime and computational resources during inference to provide a more realistic production environment. Approximately 15M tweet-user pairs had to be predicted in less than 24 hours on a single CPU core with 64GB memory, resulting in less than 6ms per prediction. Participants had to upload a compressed file with their code, features and models to score their solutions in the cloud. This requirement limited the options of algorithms and model architectures and added constraints on the file size.

The RecSys 2021 Challenge ran from March 22nd until June 23rd. Two weeks before the deadline the hosts provided the validation dataset including the targets, which was officially allowed to be used for training. Releasing the validation dataset changed the competition in a sense, as the training and test dataset have different distributions based on the time split but the validation and test dataset cover the same time period. In this paper we will refer to the setup without the validation dataset as stage 1 of the competition and the final two weeks with the validation dataset as stage 2.

3 DATA PROCESSING

Similar to the RecSys 2020 Challenge, the dataset has a tabular data structure with 20 raw features. Although deep learning can learn from raw unstructured data such as images or texts and new architectures for tabular data have been proposed, feature engineering continues to be a key component for high performing models for tabular machine learning problems. Models can learn better and faster with additional features which are related to the prediction task.

Our pipelines are end-to-end GPU-accelerated, which enabled us to quickly iterate over experiments offline. We were able to develop and experiment with more than 500 different features, using up to 151 features in a single model. The inference hardware constraints added limitations to the feature engineering by constraining the submission size to around 20GB. We had to balance the tradeoff between the increase of accuracy, runtime and size of the submission file. Applying features to the test dataset requires transferring the information in the submission file. As participants did not know which users were present in the test dataset, features needed to be calculated for all users in the training dataset. Our feature engineering and training code for last and this year are available in our GitHub repository [11] [12].

3.1 Feature Engineering

The feature engineering process requires the usage of specific techniques to extract and transform features from raw data, where some domain knowledge is always useful. As this dataset has the same data structure as last year, we highlight some new techniques and features from this year's solution. Target Encoding features were calculated with two additional out-of-fold strategies and instead of manually defining the smoothing function, the model learned it from the statistics. In addition, we leveraged the reader and author id relationship to develop a more detailed user profile.

Target Encoding: The main technique for our solution is Target Encoding, which calculates the conditional probabilities of the interaction given sets of categorical features. We explained the technique and the importance of smoothing and k-fold strategies to avoid overfitting in our last year's paper [16]. This year, we used two alternative out-of-fold strategies to scale it to the large dataset. First, we used leave-one-out strategy, which instead of k-fold uses all examples in the training dataset except for the current one to encode the categorical features.

Second, we used a time-split strategy. As target encoding features can distill information of a large portion of the training dataset, a model such as XGBoost can be trained on a small sample of the training dataset. For one of the models, the training dataset was split into two parts, one of them containing only 20% of the 3rd week. The other, larger part was used to calculate the statistics for the target encoding features, which are applied to the smaller part. For one of the models, instead of calculating the conditional probabilities, we provided raw statistics (sum of positive interactions and count of observations) as features, so that the model can learn a smoothing function.

Reversed Target Encoding: In this dataset each example contains a user id for the author and reader, which are jointly label encoded. So we engineered multiple target encoding features, leveraging features from the author to the reader and vice-versa. For example, the model can learn from these features, if a reader has a high like probability, will his/her tweets be liked by other users?
Other features: Similar to last year, we added multiple other features, such as transforming the raw data, extracting special symbols, characters or text segments or calculating ratios and can be found in our previous paper [16].

4 OUR SOLUTION TO STAGE 1

Our solution to stage 1 was an ensemble of models. In this section, we describe and analyze the models of the first stage before the validation dataset was released. We used gradient boosted trees (XGBoost) and neural networks. Each model predicted the validation and test dataset and their predictions were used as an input to the stage 2 models.

4.1 Tree-Based Models

Gradient boosted trees have excellent performance on tabular machine learning problems. As some features, such as target encoding, capture the information of the full dataset, tree-based models can be trained on a smaller sample of the dataset with only a small drop in accuracy. We developed three diverse model types, each using a different training strategy and a combination of multiple XGBoost models. In common, all model types use target encoding as a main feature engineering technique, sub-samples the dataset for training and train at least one model per target. The model types differ in the strategy used for target encoding and sampling of the training dataset. Bagging multiple XGBoost models per target by training with different random seeds improved each model type, as well as, training an individual XGBoost model per user group based on the authors follower counts. The runtime constraints prevent us from applying these strategies to all models in the final submission. A detailed summary can be found in Table 3 in Appendix B. As XGBoost supports accelerated training with multiple GPUs using dask [7] and dask_cudf [5], it easily scaled to the large dataset and was used in our final submission.

4.2 Deep Learning Models

Many recommendation systems competitions were usually won by tree-based models [9], similar to our success last year using only XGBoost models. In our recent competitions, we relied on deep learning based models [10] [15].

To improve the last years' solution, we designed three diverse deep neural networks models. They all have three components: tweet text module, categorical feature module, and continuous feature module. The output of each module, plus matrix factorization, are concatenated before passing through a few fully connected layers. The diversity of the three models come from network architecture, deep learning framework, and training procedure. Key differences are shown in Table 4 in Appendix B. The architectures are visualized in Figure 4, 5 and 6 in Appendix C.

For the tweet text module, we experimented with BERT type transformer models [8], but their inference speed on a single CPU core is too slow. So we settled on the most basic RNN: single directional, single layer Gated Recurrent Units (GRU) [4] with 48 (or 42) tokens, which are truncated if longer. Single model's performance would be higher with higher GRU dimension or token length, but we sacrificed some single model performance in exchange for more diverse models, which was later proved to be key for stage 2 stacking. In model 1, the 48 token embeddings are mean pooled instead of going through a GRU, making it the fastest model of the three.

Table 1. AP and RCE scores of the validation dataset for different models trained only on training dataset.

Model	Average AP	Average RCE
XGBoost 1	0.315	18.052
XGBoost 2	0.317	17.829
XGBoost 3	0.306	16.678
Neural Network 1	0.318	17.299
Neural Network 2	0.319	17.586
Neural Network 3	0.297	14.035
Ensemble	0.337	20.199

We use Multilingual BERT's pre-trained word embeddings, i.e., take the first layer of the pretrained Multilingual BERT model and freeze it during training. As BERT token ids were given in the training data instead of tweet text, no tokenization is needed during training or inference.

All three models are trained on NVIDIA V100 GPUs. Due to the large user embedding table size, we had to apply a few tricks to make the model fit into 32G GPU memory: (1) we discard low-frequency users from the embedding table, e.g. mapping all the users which appear less than 5 times into the same embedding, (2) use PyTorch sparse embedding layer in model 3, (3) split 20 million users into two 10 million row embedding tables and train two separate and identically structured networks for model 1, (4) AMP mixed precision training.

4.3 Results

Table 1 shows the performance of each XGBoost and neural network model on the validation dataset. The highest RCE with 18.052 of a single model is based on a XGBoost model and the highest AP with 0.319 of a single model is neural network. The ensemble of a weighted average achieved a significantly higher score of 20.199 and 0.337. As outlined in the previous section, each model is designed to be diverse to each other, which empowers the ensemble. The full inference over the test set with an ensemble of six models successfully ran in 18 hours.

5 OUR FINAL SOLUTION TO STAGE 2

After the validation dataset was released, the competition changed to leveraging the additional data it provided. This problem is equivalent to many production systems, where recommendation systems have to be frequently trained as the online service collects new data. Common techniques are to retrain the model from scratch or continuously train the model on the new data. We experimented with both approaches and experienced the best performance by stacking our models. Stacking means learning how to combine multiple machine learning models by training a model using as features the predictions of other trained models. Our stage 2 model can be simpler, such as XGBoost, and require less data to calibrate than the stage 1 models. The stacked models can leverage all information of the training dataset, as we engineered features from the train dataset and applied it to the validation dataset. The stage 1 models' predictions are another source of relevant information from the training dataset. The stage 1 models were added one-by-one using a greedy approach to determine which model is added next and the best performing version is to leverage all six stage 1 models, showing the superiority of the diverse models. We engineered additional features from the validation dataset, such as target encoding and count encoding. For local experiments, the validation dataset is randomly split into two folds with 80% for training and 20% for local validation. Two different stage 2 models are trained equivalent

Table 2. AP and RCE scores for different stage 2 models. Local CV is 20% of the validation dataset, which was not used for training. Final LB is the test dataset and official scores on the leaderboard.

	Local CV		Final LB	
Model	Average AP	Average RCE	Average AP	Average RCE
XGBoost 4	0.372	23.560		
XGBoost 5	0.373	23.690		
Ensemble	0.377	24.054	0.379	24.356

to the strategies of XGBoost 1 and XGBoost 2, where each model was bagged by training three identical copies with random seeds. The final prediction was an ensemble by averaging the stage 2 model predictions. Table 2 summarizes the performance of each stage 2 model and the ensemble. Individual models achieved an RCE of 23.56 and 23.69 and the ensemble increased the score to 23.054. As the final submission was trained on the full validation dataset, the final leaderboard score has an RCE of 24.356. Our solution provides an alternative approach to retraining or continuously training production systems. The stacked model can be trained faster and on less data by leveraging trained models from a previous time period. A new approach could be to train a stacked model with a high frequency to address the latest trends and update the stage 1 models with a low frequency.

6 ACCELERATING INFERENCE WITH A SINGLE GPU

The constraints of a single CPU core with 64GB memory as a realistic production environment significantly limited the feasible solution space. Specialized hardware, such as GPUs, are common practice in the training process. We demonstrated the speed-ups of training our models [16] last year. In order to test the performance of GPUs for inference, we accelerated our complete RecSys2021 solution to predict the validation dataset after the competition ended. For a fair comparison, we rerun our final submission using all CPU cores on the Google Cloud Instance of type e2-highmem-8.

The preprocess consisted of loading the raw TSV files, optimizing variable data types, imputing NaN values and extracting text features, such as users mentioned in the text, special keywords, length of tweets, etc. The open source library RAPIDS.AI cuDF [5] is able to accelerate the complete step including decoding the BERT text tokens and manipulating Strings. The first step took 69 seconds on a GPU instead of 1 hours and 9 minutes on a CPU. The output was consumed by all models.

Feature Engineering contains all further data transformations and depends on the model. Most common types are merging features from the train dataset to the validation dataset, calculate some ratios, extract timestamp features or normalize continuous features for neural network models. We accelerated feature engineering for 8 models from 2 hours and 52 minutes to 1 minute and 59 seconds, a speed up of 87x.

We used 5 XGBoost models of which one could be an ensemble of multiple bags. CuML Forest Inference Library (FIL) [6] is an open source library to accelerate decision trees on GPUs, which is even faster than the native XGBoost library with GPU. FIL achieved a speed up of 168 times, reducing calculating time from 2 hours to 43 seconds.

Finally, we accelerated our neural networks models in TensorFlow [1] and PyTorch [14]. We used NVTabular data loaders [13] and DLPack to keep the data on the GPU memory without any CPU-GPU communication. The default TensorFlow Keras embedding layer uses CPU operations slowing down the inference. We added a custom embedding layer, which alone provides a 10x speed-up and is available in the NVTabular GitHub repository [13]. We reduced the runtime from 2 hours and 22 minutes down to 100 seconds.

Our full pipeline runs in 5 minutes and 30 seconds on a single NVIDIA A100 GPU with 40GB memory, resulting in a average compute time of 0.022 ms per example. The GPU-accelerated pipeline is 257x faster than our final submission on a single CPU core and 92x faster than a CPU with eight cores. Figure 2 summarizes the speed ups per step in comparison to eight core CPUs on the left side. The complete details can be found in Table 5 in Appendix D and in Appendix E. Accelerating inference with a NVIDIA A100 GPU reduces the costs by 88% in comparison to using e2-highmem-8 with eight CPUs as visualized in Figure 2 right. As an alternative, larger models with higher accuracy would be able to run in the latency requirement of 6ms per example by using GPU acceleration.

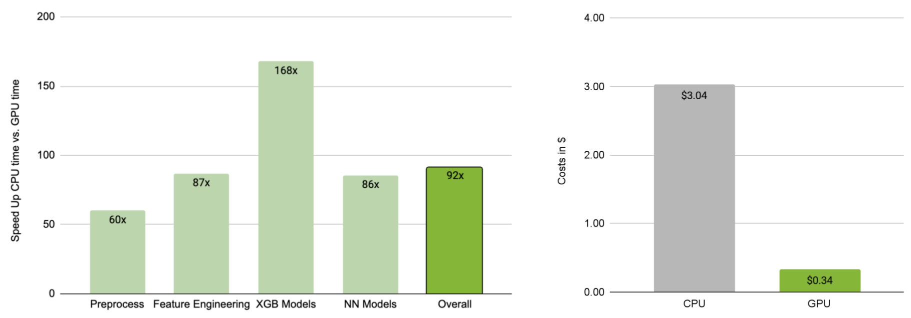

Fig. 2. On the left side, the barchart visualizes the speed-ups per pipeline step and overall. It compares the runtime of a 8-core CPU with 64GB memory and a NVIDIA A100 GPU with 40GB memory. The right side compares the costs of running a e2-highmem-8 for 8 hours and 24 minutes and running an a2-highgpu-1g for 5 minutes and 30 seconds.

7 CONCLUSION

In this paper we presented our 1st place solution for the RecSys2021 competition. In addition to XGBoost models, we proposed three different neural network architectures. Our stage 1 ensemble of three XGBoost and three neural networks showed the advantages of combining diverse models and demonstrated the feasibility to deploy them given the latency and compute constraints. As production systems often rely on a single model, we think that our ensemble will facilitate rethinking about ensembles in production. In stage 2, we trained a stacked model, which leveraged existing models and required less data to train. This technique can be further evaluated as an alternative to current approaches of retraining or fine tuning models with new data. Finally, we demonstrated the benefits of GPU-accelerated inference by using the open source libraries Forrest Inference Library, NVTabular, RAPIDS cuDF, PyTorch and TensorFlow. In our experiment, we reduced the runtime from 23 hours and 40 minutes on a single CPU or 8 hours and 24 minutes on a 8-core CPU to 5 minutes and 30 seconds on a single NVIDIA A100 GPU, resulting in a speed up of 257x and 92x. Simultaneously, infrastructure costs will be reduced by 8-9x. This result should start the paradigm shift to GPU-accelerated inference, which enables larger, more accurate recommender system models. We published our final solution [12] and hope it is useful for the community to build, scale and deploy recommender systems with GPUs.

ACKNOWLEDGMENTS

The authors wish to thank our colleagues on the NVIDIA Merlin, KGMON and RAPIDS.AI teams for their support in this competition.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[2] 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-Preserving Recommender Systems Challenge on Twitter's Home Timeline. (2020).

[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[4] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Y. Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. (06 2014). https://doi.org/10.3115/v1/D14-1179

[5] cudf 2021. *cuDF - GPU DataFrames*. Retrieved July 20, 2021 from https://github.com/rapidsai/cudf

[6] cuml 2021. *cuML - GPU Machine Learning Algorithms*. Retrieved July 20, 2021 from https://github.com/rapidsai/cuml

[7] dask 2021. *Dask: Scalable analytics in Python*. Retrieved July 20th, 2020 from https://dask.org/

[8] J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

[9] Dietmar Jannach, Gabriel de Souza P. Moreira, and Even Oldridge. 2020. Why Are Deep Learning Models Not Consistently Winning Recommender Systems Competitions Yet? A Position Paper. In *Proceedings of the Recommender Systems Challenge 2020* (Virtual Event, Brazil) *(RecSysChallenge '20)*. Association for Computing Machinery, New York, NY, USA, 44–49. https://doi.org/10.1145/3415959.3416001

[10] Gabriel de Souza Pereira Moreira, Sara Rabhi, Ronay Ak, Md Yasin Kabir, and Even Oldridge. 2021. Transformers with multi-modal features and post-fusion context for e-commerce session-based recommendation. In *Proceedings of the Fifth SIGIR Workshop on eCommerce (SIGIR eCom'21)*.

[11] NVIDIA RAPIDS.AI 2020 2021. *NVIDIA RAPIDS.AI's final model for RecSys 2020 Challenge*. Retrieved July 20th, 2020 from https://github.com/rapidsai/deeplearning/tree/master/RecSys2020

[12] NVIDIA RAPIDS.AI 2021 2021. *NVIDIA RAPIDS.AI's final model for RecSys 2021 Challenge*. Retrieved July 20th, 2020 from https://github.com/NVIDIA-Merlin/competitions/tree/main/RecSys2021_Challenge

[13] NVTabular 2021. *NVTabular*. Retrieved July 20, 2021 from https://github.com/NVIDIA/NVTabular/

[14] Adam Paszke, S. Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, N. Gimelshein, L. Antiga, Alban Desmaison, Andreas Köpf, E. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*.

[15] Benedikt Schifferer, Chris Deotte, Jean-Francois Puget, Gabriel de Souza Pereira Moreira, Gilberto Titericz, Jiwei Liu, and Ronay Ak. 2021. Using Deep Learning to Win the Booking.com WSDM WebTour21 Challenge on Sequential Recommendations, In Proceedings of the Workshop on Web Tourism (WebTour) co-located with the 14th ACM International WSDM Conference (WSDM 2021). *CEUR Workshop Proceedings* Vol-2855.

[16] 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

[17] Twitter 2021. *Twitter RecSys Challenge 2021*. Retrieved July 20th, 2021 from https://recsys-twitter.com/

A INTERACTIONS OF USERS

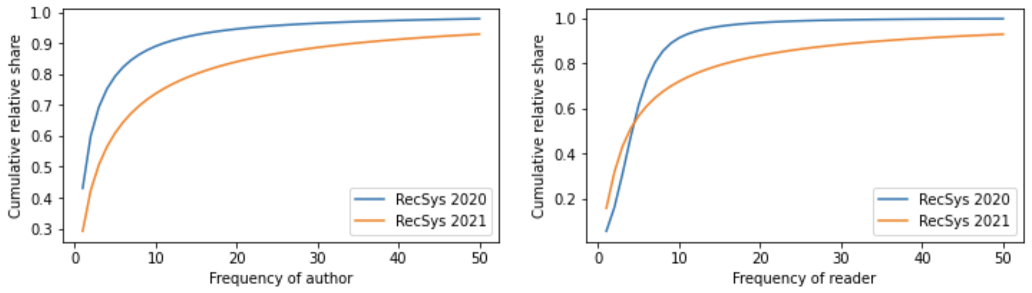

Fig. 3. Number of interactions for author (left) and reader (right).

B OVERVIEW MODEL CONFIGURATIONS

Table 3. Overview of the configuration for the three different XGBoost model types.

	XGBoost 1	XGBoost 2	XGBoost 3
Target Encoding Strategy	5-Fold	Leave-One-Out	Time-based split
Number of Features	70	Between 77-79	69
Number of Trees	250	Between 450-1000	1000
Trained per user group	No	No	Yes
Number of Bags	3	1	1
Number of Models	12	4	20
Training data	150M rows that contains users present in both week 3 and a prior week	Week 2 and Week 3 (depending on the target)	20% of 3rd week

Table 4. Overview of the configuration for the three different neural networks.

	Neural Network 1	Neural Network 2	Neural Network 3
Word Embeddings	Random Initialized	Pretrained BERT multilingual	Pretrained BERT multilingual
Token Lengths	48	42	48
Text Token Layer	Mean Pooling	GRU with dim 64	GRU with dim 128
Framework	Tensorflow	Pytorch	Pytorch
Continuous Features	27	41	36
Using TE Features	No	Yes	No
User Embeddings	author and reader jointly embedded at dim=96	reader only, dim=64	author and reader jointly embedded at dim=128
Frequency Threshold	5	7	3
Matrix Factorization	author x reader; reader x tweet	reader x tweet	author x reader
Model Size	7.4 G	3.9 G	5.4 G
Sparse Embedding Layer	No	No	Yes
Split into two models	Yes	No	No
Training time	20 hours (for 2 NNs) on 4xV100	48 hours on 1xV100	87 hours on 1xV100
Training data	all data	week 1-2	all data
# Epochs	2	2	3

C NEURAL NETWORK ARCHITECTURES

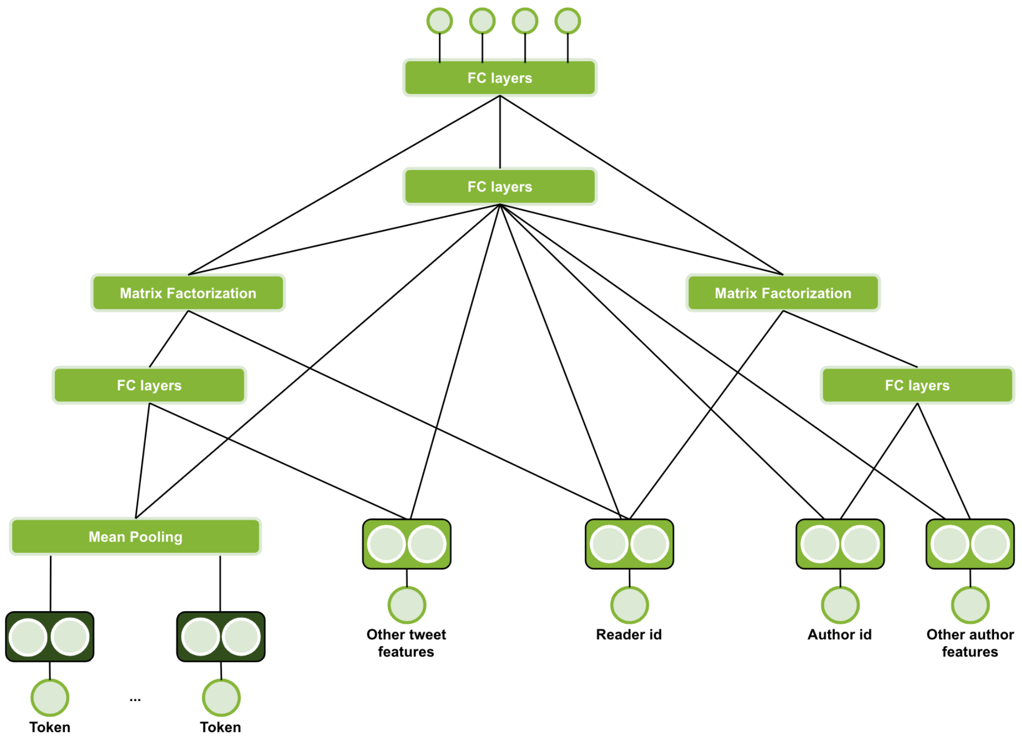

Fig. 4. Visualization of neural network 1. FC layers can be multiple fully connected layers.

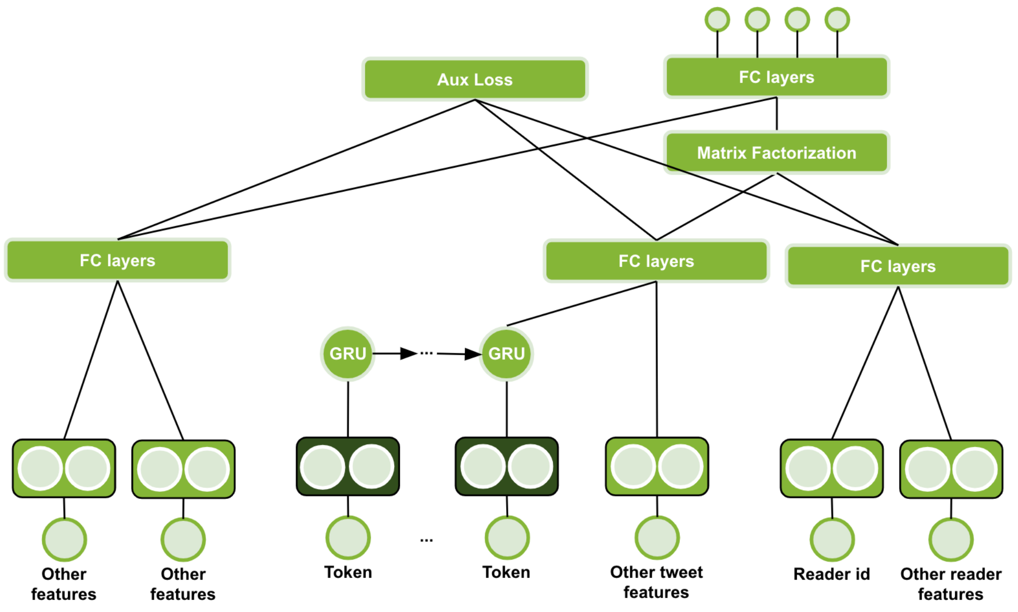

Fig. 5. Visualization of neural network 2. FC layers can be multiple fully connected layers.

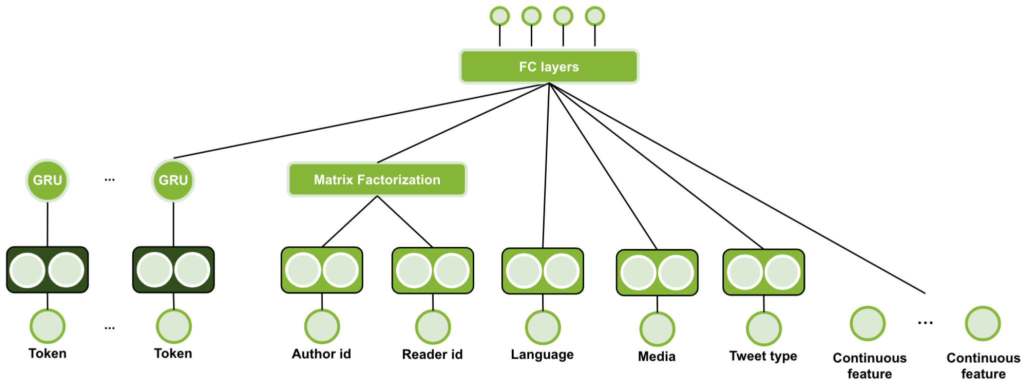

Fig. 6. Visualization of neural network 3. FC layers can be multiple fully connected layers.

D COMPUTATION TIME FOR INFERENCE

Table 5. Runtime in seconds per step and total for the full pipeline, containing 5 XGBoost models and 3 neural network models.

	8 core CPU	NVIDIA A100 GPU	Speed Up
Preprocess	4155	69	60
Feature Engineering	10334	119	87
XGB Models	7193	43	168
NN Models	8540	100	86
Overall	30223	330	92

E COST CALCULATION FOR INFERENCE

CPU Costs: The competition required to predict the test dataset in 24h given a single CPU core with 64 GB memory. As Google Cloud Platform does not offer single CPU core with 64 GB memory, we selected the smallest instance providing 64 GB memory. The instance type e2-highmem-8 provides 8 CPU cores with 64 GB memory and costs $0.36/h. Cost of larger instances scales linearly with the computational resource. As pipelines does normally not scale linearly with more computational resource, e2-highmem-8 should provide the best cost efficiency for CPU instances. Our final submission ran in 8 hours and 24 minutes on a e2-highmem-8 without limiting the CPUs or memory. The costs for a e2-highmem-8 for 8 hours and 24 minutes are $3.

GPU Costs: After the competition ended, we wanted to compare the speed up and cost efficiency of GPU-accelerated inference. We converted our full pipeline by using open source libraries to run it on a single NVIDIA A100 GPU with 40GB memory. The execution time for the same pipeline is 5 minutes and 30 seconds. Google Cloud Platform offers NVIDIA A100 with 40GB memory with the instance type a2-highgpu-1g for $3.675/h. The costs for a a2-highgpu-1g instance for 5 minutes and 30 seconds are $0.34.