Personalizing Multi-Modal Content for a Diverse Audience: A Scalable Deep Learning Approach

Nishant Oli
nishant.oli@glance.com
Glance
Bangalore, India

Aditya Patel
aditya.patel@glance.com
Glance
Bangalore, India

Vishesh Sharma
vishesh.sharma@glance.com
Glance
Bangalore, India

Sai Dinesh Dacharaju
sai.dinesh@glance.com
Glance
Bangalore, India

Sushrut Ikhar
sushrut.ikhar@inmobi.com
InMobi
Bangalore, India

ABSTRACT

In recent years, deep neural networks have not only made an entrance but have flourished in the field of recommendation systems following their immense popularity in speech recognition, natural language processing and computer vision. While this holds true in the context of traditional recommendation problems with static, homogeneous inventory and also in the context of pure news-based recommendation, the industry at large is perpetually pursuing scalable, economical and tailor-made solutions for unique problems. In this paper, we discuss one such deep-learning based solution to the problem of recommending heterogeneous content, in that the content may vary based on factors such as lifetime, format and language. The proposed system generates a multi-modal feature space for representing and learning user-content relationship, taking its inspiration from the setting of traditional collaborative filtering algorithms. With an aim to focus on implementation and industry-based applications, we also address the problem of serving DNN-based model predictions in live environment efficiently. To that end, we share our system design and learnings that have yielded 15x throughput improvement while making 150K predictions per core per second. The performance of this system in live environment is encouraging, with improvements of 2.3% and 8.1% in click-through rate (CTR) and total duration respectively as compared to the existing system.

1 INTRODUCTION

Internet coverage has witnessed staggering growth in the past decade with more than half of the world's population now online. This growth has caused a surge in creation, consumption and variety of content. Advancements in the fields of neural machine translation, speech recognition and natural language processing have helped on-board diverse groups of first time internet users from developing countries. Most of these users prefer consuming content in their respective vernacular languages while adopting the internet as their new source for content consumption. Personalization of users' content feeds thus becomes inevitable when we try to solve this problem of information overload for a diverse target audience. It helps in improving user experience which is instrumental in increasing overall product engagement.

Glance[1] is a content discovery platform with a current reach of 100Mn daily active users (DAU). In order to cater to wide-ranging interests of these users, content on Glance is varied in terms of lifetime - ephemeral or timeless, format - article or video, mode - image or text, and language - English, Hindi and Bahasa among others.

Figure 1: A preview of Glance: As can be seen, a glance card consists of Image & Text with an embedded call-to-action (CTA)

The context of this paper, although inspired from our work at Glance, can be generalized as building a recommendation engine which addresses a combination of following challenges:

- Ephemeral item inventory: Frequently changing, low lifetime content
- Multi-modal content: Content varying in format and language

[1]Glance: https://glance.com.

- Recommendation at scale: Model and serving architecture capable of serving millions of users

After considering various approaches in existing literature, which we cover in detail in the Related Work section, we noticed that they solved only a subset of the challenges highlighted above. The need for a more generalised approach to address our additional requirements and our efforts in the said direction led to the work in this paper. Firstly, the usage of traditional collaborative filtering methods was ruled out due to the ephemeral nature of the content along with the vast user base which made model training difficult. Secondly, while news recommendation systems solve the problem for ever- changing content they do so by considering single modality of content which is not the case at Glance. Lastly, click-through-rate prediction models in several contexts require static user based meta information like demographics and location which is not available in our case owing to our philosophy of having a stateless user experience. In this paper, we propose a deep learning based solution to build a model that can leverage the effectiveness of collaborative filtering while simultaneously scoring newly generated content. In our endeavour to emphasize on execution and focus on industry based applications, we also present the lessons we learnt while taking a DNN-based model to production in System Deployment Section.

The core of our approach is:

- A Multi-Modal Encoder that produces domain-specific text embeddings by making use of a BERT model we fine-tuned for our use case. Fine tuning BERT for domain specific use cases is discussed in Our Approach section.
- A neural network architecture called *FM-Lite* that combines cross-domain user interests and item features jointly optimizing user's prediction behavior
- A system architecture that meets strict service level agreement (SLA) requirements by using a distributed prediction service powered by caching and optimized operations to deploy neural network models in real world

In Dataset & Results section, we present offline performance metrics in comparison to other well-known neural network based architectures, followed by our model's performance on live traffic.

2 RELATED WORK

Traditional recommendation algorithms are broadly divided into three categories: Collaborative Filtering [1],[2],[3], Content Filtering [4],[5],[6] and Hybrid [7],[8]. While content filtering aims to build user and item portraits based on explicit item attributes, collaborative filtering works without any prior information about an individual user or item. The latter uses interaction data of all users and items in the defined world to derive intrinsic patterns in interests. It helps discover new interests of a user that might fall outside the subspace defined by their historical interactions. In recent years, collaborative filtering via matrix factorization has become the default choice to build recommendation systems [1],[3],[9]. However, it is seen to perform well when you have enough interactions, which are harder to come by in ever-changing content. Content filtering on the other hand tends to be more robust against popularity bias and the cold start problem but limits the scope of recommendation owing to its high dependency on explicit item attributes. When run for too long, pure content filtering based approach causes user fatigue due to predictable recommendations. Hybrid filtering combines content and collaborative filtering, to capitalize on the benefits and curtail the disadvantages of these two approaches.

Considering the ever changing, quickly expiring nature of our content, we took to news recommendation use cases for initial inspiration. Content filtering and Collaborative filtering are usually combined to build models that make sense of user interactions [10],[11], [12] & [13]. As mentioned above, regular collaborative filtering methods fail to deliver with cold items and limited interactions. Okura et al. [13] solved this using an auto-encoder to build item embeddings for every article based on the word statistics in the documents. The embeddings of items that constitute a user's historical preferences are then fed to a Recurrent Neural Network (RNN) to obtain that user's embedding. Because of ephemeral nature, hybrid filtering techniques have also been employed to build recommendation systems for news content. Moerland et al. [14] use semantic similarity instead of Bag-of-Words (BOW) to rank items.

With neural networks adoption accelerated by GPUs, more and more organizations have started experimenting with neural nets in the field of recommendations. Currently, most companies such as Netflix [15], Airbnb [16], MSN [17] are using neural nets to power their recommendation systems. He et al. [18] used neural nets to introduce non-linearity in collaborative filtering via two-way interactions called Neural Collaborative filtering (NCF). Few works such as Efficient Extreme Classification by YouTube [19] and Wide and Deep [20] by Google use neural network models to treat recommendation as a classification problem. In Wide and Deep [20], Cheng et al. combine a wide network with a deep channel using a multi-layer perceptron. The former is used to memorize interactions while the latter models more complex interactions effectively.

We also came across literature that addresses multi-modality. In their work, Elkahky et al. [21], make use of a multi-view deep neural network architecture in order to understand users' interests more comprehensively. A user's interests sourced from different domains are jointly mapped to a semantically richer user latent feature vector that is more compact. By doing so, in addition to getting a richer user embedding, they also improve recommendation quality across involved domains.

3 PROBLEM FORMULATION

On Glance, a user interacts with a content card (called a glance card) by clicking on it, referred to as a 'peeking' action on the lock screen. While the content of a glance card can vary from news to featured content in multiple languages and categories, the format of this card can be image or video based with some text embedded in either case for title and summary. Based on lifespan, glance cards can be positioned in a wide spectrum ranging from ephemeral to perennial, with the majority expiring within 48 hours. A glance card, as can be seen in Figure 1, has multiple components like an image, title text, summary text and a call-to-action button (play button in case of videos).

To define the problem we define our interaction data as $\mathbf{X}$, where each $\mathbf{x} \in \mathbf{X}$, has $(\mathbf{u}, \mathbf{g}, y, \mathbf{d})$, where $\mathbf{u}$ and $\mathbf{g}$ are user and glance pair, with y being a binary variable with value 1 or 0, if there is a peek or

no peek respectively and d being the time spent on a glance card by the user. The pair **u** and **g** are part of set of users **U** with |**U**| = n and a set of items (glance cards) which the user has interacted in past **G** with |**G**| = m. We formulate the problem by getting a probability of click, for a given user and glance and ranking them according to the probability.

$$p(y = 1|\mathbf{u}, \mathbf{g}) = f(\mathbf{u}, \mathbf{g}) \quad (1)$$

The challenge with solving this problem lies with the short shelf life (expires in hours) of glances, comprehending the context of the text & image, multi-modal content (text and images) in the glance card and the absence of static user features (like demographic information, user social connections etc)

Furthermore, on the serving side, we had to come up with a system that could cater to user requests with predictions while honouring the latency limits (up to 50ms) and high traffic (with traffic at peak hours surging up to 1.2M prediction requests per second) so that the user experience is not compromised.

4 OUR APPROACH

We solved the problem by dividing the model into 3 parts, Multi-Modal Encoder or Glance Encoder, User Encoder and the Prediction Model. *Multi-Modal Encoder or Glance Encoder* encodes each glance to an embedding. Since the glance-card is short-lived in nature, we rely totally on the content (text, image, and meta-information) to get the embedding. *User Encoder* uses the glance embedding of all glances interacted by the user, and creates an embedding for a user from historical interactions (28 days) of the user. Finally, the *Prediction Model* predicts the probability of a user peeking a glance, given their embedding obtained from the respective encoders.

4.1 Multi-Modal Encoder

Multi-Modal Encoder or *Glance Encoder* has three main components, Meta Features Encoder, Image Encoder, and Text Encoder. Glance Embedding has all the set of features for the item, which, in our case, is a glance card. To represent these features for the glance, we treat various features based on how accurately they represent the Glance's theme. We append the embedding from three encoders viz - text, image and meta-features to obtain the glance embedding.

4.1.1 Text Encoder. In recent years we have seen many advancements in the field of Natural language processing (NLP) to represent words/sentences in the form of embedding. For our use case we represented the text into a continuous space by evaluating various existing methods in our downstream task, we explored Word2Vec[22], Universal Sentence Encoder (USE) [23] and BERT[24]. From our experiments mentioned in Table 1 we found that although the available sentence embedding perform well for standard tasks, they did not perform well for our task. The short and direct nature of text in Glance from a diverse set of categories required us to fine tune to improve the feature representation and thus performance.

BERT: Bidirectional Encoder Representations from Transformers [24] is a deep bidirectional model that is pre-trained in a semi supervised fashion for a masked language model task and next sentence prediction task, conditioning on both the left and the right context of the text in all layers. The pre-trained model can be fine-tuned to most of the language tasks and create state of art models.

- Masked Language Model: In this task, authors mask a fraction of the input tokens randomly, and then predict the masked tokens.
- Next Sentence Prediction: In this task two sentences, namely A and B are provided as input to the model. 50% of the time sentence B is made the next sentence for A while the other time there is a random allotment. Finally the CLS [24] token's output is taken as the label for the next sentence.

Fine-Tuning BERT. Fine-Tuning BERT[2] is a method that yields better results for downstream and domain specific tasks, as shown by Sun et al. [25] Lee et al. [26] & Beltagy et al [27].

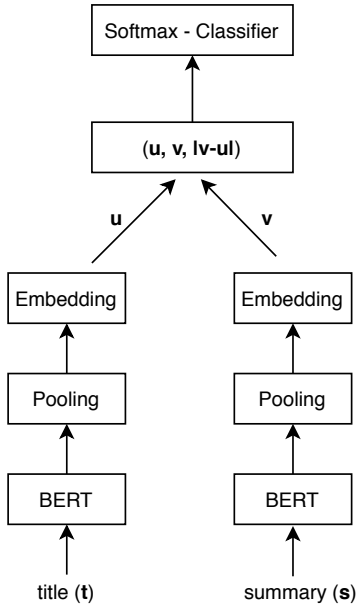

Figure 2: Framework for fine-tuning the BERT embedding for glance's data. The title (t) & summary (s) are passed as the input, which transforms to u & v before passing it to loss and then a softmax layer.

We first pre-process and translate all the non English text to English text using Neural Machine Translation (NMT) [28]. We decided to translate the non-English text to English due to the lack of a pre-trained BERT on non-English indic languages such as Tamil, Telugu and Kannada. Post this, to improve the quality of the textual embedding that we use as a feature in our recommendation models, we fine-tuned the BERT base model on our own glance dataset [29]. We followed the work done by Reimers et al [30], which uses a Siamese Network Structure [31] to fine-tune the BERT model. The fine-tuning architecture as shown in Figure 2 follows the presence of a pooling layer on top of the BERT model, followed by a loss function which in our case was mean absolute error and then a softmax function. The final hyper-parameters found after experimentation were that of batch size 16 with Adam optimizer and a learning rate of $2e^{-5}$ along with a linear learning rate warm up over 15% of the data. The pooling logic we used was Mean

[2]Github Source Code: https://github.com/vishesh60/bert_finetuning.

Pooling with a training set of 40K sentence pairs which was fixed after checking the relation between loss and the number of sentence pairs as seen in Figure 3.

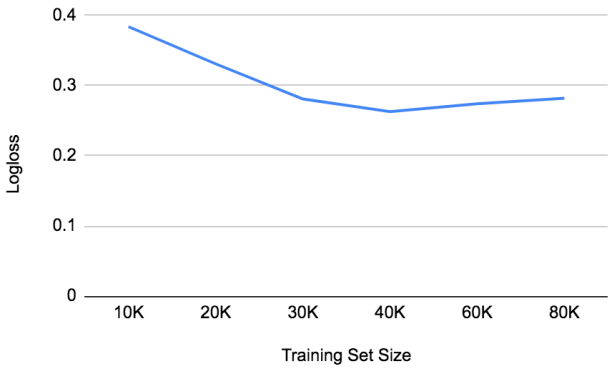

Figure 3: Relation between loss and number of training data examples for BERT finetuning

We concatenate the sentence embeddings **u** and **v**, (obtained by passing title (**t**) and summary (**s**) through BERT and pooling as shown in Figure 2) with the element-wise difference |**v** - **u**| and multiply it with the trainable weight **W** as shown below:

$$P(\mathbf{y} = 1|\ \mathbf{u},\ \mathbf{v}) = \sigma(\mathbf{W}(\mathbf{u},\ \mathbf{v}\ , |\mathbf{v} - \mathbf{u}|)) \tag{2}$$

We use Glance text data to train and validate the model. To avoid manual tagging of the dataset for fine-tuning, we followed a format wherein a title (**t**) and summary (**s**) of a particular glance was marked with a supervised label 1, while for label 0 we picked glances from different categories to ensure true negatives. We did this as within the same category, there were chances of two glances having higher semantic similarity which can lead to introduction of false negatives in the training dataset. We evaluate using 3 pooling strategies: mean-pool of all the output vectors, max-pool of all the output vectors, and output of the [CLS] token.

Table 1: Offline Experimentation of various embeddings and BERT fine-tuning with different pooling strategies with their log loss

Embedding	Before Finetuning	After Finetuning
Word2Vec [22]	0.477	-
USE [23]	0.328	-
BERT Base mean	0.446	0.315
BERT Base cls	0.457	0.263
BERT Base max	0.599	0.298

The BERT Base fine-tuned model with the pooling from CLS token is chosen for creating the embedding as it performs the best in our downstream task Table 1. We use this method to generate the text embeddings by appending the title and summary, $\mathbf{g_{TF}}$.

4.1.2 Image Encoder. To create the image features for the glance, we bring it into the same semantic space as the text features, so that we can jointly optimize for the different characteristics such as the Text/Image available per glance. To create the image embedding we utilized the Neural Image Caption Generation model by Xu et al [32]. The caption also aids in capturing the aesthetics and abstract nature of the images (Example: an article about a stack of notes in a business news). The generated caption is then passed through the Text Encoder module to generate the embedding $\mathbf{g_{IF}}$.

4.1.3 Meta-Features Encoder. Meta Features such as the language, modality, type of the item, dominant colors, tags etc. are one-hot encoded and appended to form the meta feature embedding $\mathbf{g_{MF}}$.

$$\text{So, } \mathbf{g}_i \ \forall\ \mathbf{G} \ \text{ we have a } \ \mathbf{g}_i^F = [\mathbf{g}_i^{MF}\ \mathbf{g}_i^{IF}\ \ \mathbf{g}_i^{TF}] \tag{3}$$

4.2 User Encoder

User Encoder, encodes every user to an embedding space, based on their historical interactions with the glances. It uses the content (text and image) of all the peeked glances along with their affinity towards each of the meta feature of the glance to get a unique embedding such that users with similar interest come closer in the embedding space and the users with dissimilar interests will be farther away. Since we have stateless system i.e. there are no static user information, we generate all the user features from 3 sections. We divide the user encoder into 3 different sections which are finally appended together to get a user embedding.

4.2.1 Text & Image Embedding ($\mathbf{u}_{TF}, \mathbf{u}_{IF}$). We use the *Text Embedding* ($\mathbf{g}_{TF}$) of all glances from the user's positive interaction history (i.e. interaction with a peek) and pass them through a mean pooling to get the User's Text Embedding.

Similarly, we use the *Image Embedding* ($\mathbf{g}_{IF}$) of all glances from the user's positive interaction history (i.e. interaction with a peek) and pass them through a mean pooling to get the User's Image Embedding.

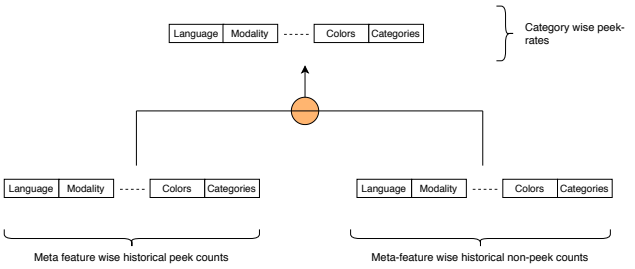

Figure 4: Building the User Meta Feature Embedding: Using the historical data of a user, we build a set of user-features that gives the user peek-rates

4.2.2 Meta-features Embedding ($\mathbf{u}_{MF}$). We use the *meta-feature embedding* of all the glances the user interacted with, to build the user interest profile as shown in Figure 4. The meta features for users are obtained by passing the positive interactions to the mean pooling layer, normalizing each one of the pooled values with the output from the mean pooling from all the interactions. This

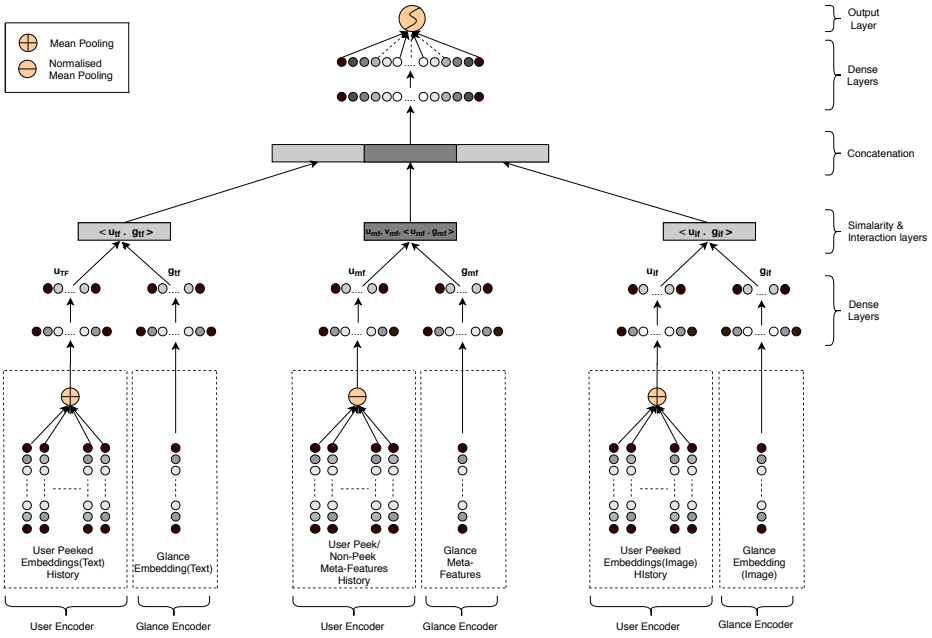

Figure 5: FM-Lite Architecture. Historical interactions (peeked and non-peeked) of glances are pre-processed to obtain the User and Glance Features using User and Glance Encoder.

normalization is equivalent to user's peeking rate for the given categorical feature value.

These features built from the user's past history of peek and non-peek, forms the core set of features that defines the interest of a user for specific type of content like the interest of user towards a specific language, or modality like article or video, or category like sports, food,.. and likewise for other set of features. This embedding maps each user into a meta-feature space, where each value in the embedding represents the user's peek-rate(interest) for that meta-feature, hence giving each user a unique embedding altogether.

Point to be noted is users with similar peeking history might have similar text and image embedding, but they might have different user embeddings since their meta-features embedding is not similar as it considers both peek and non-peek history.

$$\text{So, } \mathbf{u}_i \ \forall \ \mathbf{U} \ \text{ we have a } \ \mathbf{u}_i^F = [\mathbf{u}_i^{MF} \ \mathbf{u}_i^{IF} \ \ \mathbf{u}_i^{TF}] \tag{4}$$

4.3 FM-Lite

We get the embedding for a user i and glance j as $\mathbf{u_i^F}$ from the *User Encoder* and $\mathbf{g_j^F}$ from the *Glance Encoder* and every such pair has a label y = 0 or 1, (where y = 1 means a peek and y = 0, otherwise) coming from the training data of interactions. We train the model (f) on user embedding and item embedding for all such pairs to predict the probability of user clicking on the item.

$$P(y = 1|\mathbf{u_i^F}, \mathbf{g_j^F}) = f(\mathbf{u_i^F}, \mathbf{g_j^F}) \tag{5}$$

To predict this probability, we use a multiple input single output architecture as shown in Figure 5, where we provide the various types of inputs to the model separately and train all the different sections in a joint fashion. The input sections takes the different types of inputs separately viz the text ($\mathbf{u_i^{TF}}, \mathbf{g_j^{TF}}$), image ($\mathbf{u_i^{IF}}, \mathbf{g_j^{IF}}$) and other meta features ($\mathbf{u_i^{MF}}, \mathbf{g_j^{MF}}$). All these three inputs are first reduced to a lower dimension (($\mathbf{a}_u^{IF}, \mathbf{a}_g^{IF}$), ($\mathbf{a}_u^{MF}, \mathbf{a}_g^{MF}$) & ($\mathbf{a}_u^{TF}\ \mathbf{a}_g^{TF}$)) using dense layers. This brings input (user and item) to a common feature space and also aids in reducing latency of the system as seen in Table 2, the parameters used are almost 4x lesser than the Wide & Deep Network [20].

After reducing the 3 inputs sections to lower dimensions (for both user and glance embeddings), we jointly trained the model for all 3 sub-spaces with three different functions as seen in Eq. 6. It is different from taking an ensemble of 3 different models since here gradient flows from output to input jointly. Handling the sections differently, helps model to have content-based recommendation capability (from text and images) and collaborative-based capability by using Factorization Machine layers.

$$P(y = 1|\mathbf{u}_i^F, \mathbf{g}_j^F) = h\Big(g_1\big(\mathbf{a}_u^{IF}, \mathbf{a}_g^{IF}\big),\ g_2\big(\mathbf{a}_u^{MF}, \mathbf{a}_g^{IF}\big),\ g_3\big(\mathbf{a}_u^{TF}, \mathbf{a}_g^{TF}\big)\Big) \tag{6}$$

The text and the image section of the model are build on past positive interactions and attempts to find an item from the candidate glances giving a content-filtering capability to the model. We use an inner product (defined by $\langle\ .\ \rangle$) to find the most similar glance from user history, giving $\mathbf{s_{IF}}$ for image-based features and $\mathbf{s_{TF}}$ for text based features.

$$\mathbf{s_{IF}} = \langle\ \mathbf{a}_u^{IF}\ .\ \mathbf{a}_g^{IF}\ \rangle \tag{7}$$

$$\mathbf{s_{TF}} = \langle\ \mathbf{a}_u^{TF}\ .\ \mathbf{a}_g^{TF}\ \rangle \tag{8}$$

The meta-feature embedding of the user and item are passed to a different section to get $\mathbf{s_{MF}}$, drawing its inspiration from the

DeepFM architecture, where an intermediate layer is used to get the 1st-degree and the 2nd-degree cross features. Contrary to traditional collaborative filtering systems, which work on a user and item level, we use meta-features ($\mathbf{a}_u^{MF}, \mathbf{a}_g^{MF}$), so we do not need to train for each and every user. This reduces the training time of the model, hence we call it *FM-Lite*, a computationally lighter version of the factorization machines.

$$\mathbf{s_{MF}} = [\mathbf{a}_u^{MF}\ ,\ \mathbf{a}_g^{MF}\ ,\ \langle\ \mathbf{a}_u^{MF}\ .\ \mathbf{a}_g^{MF}\ \rangle] \tag{9}$$

All these outputs from different sections ($\mathbf{s_{IF}}, \mathbf{s_{TF}}$ and $\mathbf{s_{MF}}$) are concatenated together, and which is then fed to one common logistic loss function for joint training. We backpropagate the gradients from the label to the input using Stochastic Batch Gradient Descent Algorithm with Adam Optimizer, with learning rate = 0.001.

5 DATASET & RESULTS

5.1 Dataset

To build the dataset, we take the user's interaction data for 35 days. We take the interaction data for the first 28 days to build the *user embedding* and *glance embedding* and use the later 7 days for interactions to train on. This was done to mimic the online setting of the model.

To build our train and validation data, we split the training data, by dividing the users into 80:20 split. Training Data has interactions from all user from the 80 split, and the Validation set has it from the users from 20 splits. This gives model capability to predict for the users it was not trained for hence we can reduce the training data by not training for all the users.

5.2 Offline Evaluations

As part of offline evaluations, we test the model on interactions of a single day. We evaluate the model's performance against following models.

Baseline: We compute cosine similarity between the user features and glance features and use it as a baseline model.

Linear: We concatenate the user features and glance features, and pass it through a Logistic Regression Model.

MLP: We concatenate and take inner product of the user features and glance features, and pass it through a Dense ReLU Layers.

Wide & Deep: We concatenate the user features and glance features, and pass it through a Wide & Deep Model.

5.2.1 Metrics. For offline experimentation, we use a popular and widely accepted metric, Mean Average Precision MAP@k and a custom metric Mean Peek Rate Ratio MPRR@k. The motivation behind using this custom metric is to check, if the content rated high by the model (let us say top k content) has a higher rate of engagement than the overall rate of engagement for the users. This gives us a relative measure of liking of content by the user. We define both the metric:

Mean Average Precision: MAP is the average $AP@N$ metric over all $|U|$ = n users. Average Precision ($AP@N$) is defined as Precision@k across all recommended item.

$$MAP@k = \frac{1}{n}\sum_{\mathbf{u}_i \in \mathbf{U}} \frac{1}{m}\sum_{i=1}^{k} P(i).rel(k) \tag{10}$$

Table 2: Comparison of various models

Model	MAP@25	MAP@50	MPRR@25	MPRR@50	Parameters
Baseline	0.406	0.368	1.075	1.026	-
Linear	0.437	0.433	1.112	1.044	1.2 k
Dense	0.485	0.458	1.233	1.061	11.5 Mn
Wide & Deep	0.491	0.465	1.238	1.062	8.8 Mn
FM-Lite	0.504	0.479	1.261	1.067	2.4 Mn

Mean Peek Rate Ratio: We define the custom metric Peek Rate Ratio (PRR) for a user as the ratio of peek-rate of the top k prediction for the user to the peek-rate of all the predictions for the same user. Mean Peak Rate Ratio (MPRR) is the average Peek Rate Ratio (PRR) across all users U where $|U| = n$.

$$MPRR@k = \frac{1}{n}\sum_{u_i \in \mathbf{U}} \frac{PR(k)_{u_i}}{PR_{u_i}} \tag{11}$$

where $PR(k)_{u_i}$ is the peek rate of top k prediction and PR_{u_i} is the overall peek-rate for the i^{th} user

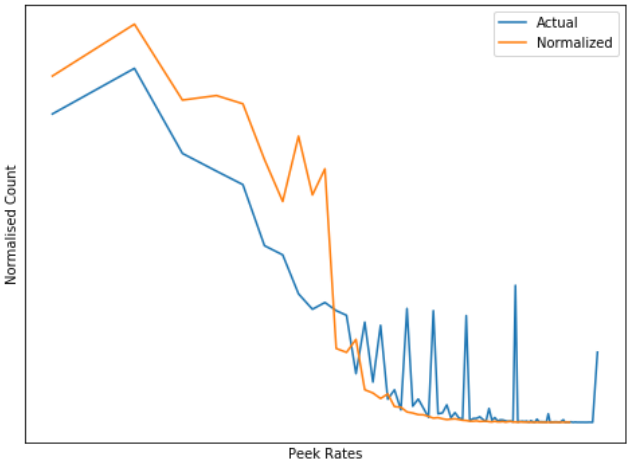

Figure 6: Comparison of histogram of peek rates before and after normalisation.

5.2.2 Normalizing Users. We see that many users, because of the lower number of interactions have relatively higher peek-rates than the users with a greater number of interactions, this is due to data insufficiency of the new users or less active user. To normalize this, we use the beta distribution's inverse cumulative density function to get a normalizing value for a user. We saw an improvement in our offline results when we normalize the user's peek rates. We can see in Figure 6 the histogram of peek rate of users before and after the normalization. Normalization reduces high peek rates of irregular users. For a user $\mathbf{u}_i$ with peeks = a and no peeks impression = b, we define n as

$$n = 1\ -\ \left(F^{-1}(p = 0.975|a,b)\ -\ F^{-1}(p = 0.025|a,b)\right) \tag{12}$$

$$\text{where,}\quad p = f(x|\ a,b) = \frac{1}{\mathbf{B}(a,b)}\int_0^x y^{a-1}(1-t)^{b-1}dt$$

Table 3: Comparison of the Recency Experiments, with or without the decay function.

Model	MAP@50
No-Decay	0.478
Decay	0.481

where $\mathbf{B}(a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$ and $\Gamma(z)$ is a Gamma function. The beta function, **B**, is a normalization constant to ensure that the total probability is 1.

5.2.3 Negative Sub-sampling. The data is an unbalanced, with peek to non-peeks ratio equal to 1:99. We sampled the negative classes with various fractions and found that sub-sampling the negative class to 2:1 makes the model perform best.

5.2.4 Exponential Decay. We also experimented with recency of the label by weighing more on recent interactions, by incorporating a decay function. We define the recency for an interaction, as

$$d(t) = \exp\left(\frac{(t - t_p)}{T}\right) \tag{13}$$

where, t = is the day of interaction, t_p is the present day and T = 28, that depends upon how long we look back while creating the user profile. We saw an increase in the offline results as seen in Table 3

We evaluated the model on two metrics MAP and MPRR. Along with that we experimented with few changes like normalising the peek-rates, various sub-sampling fractions and introducing a recency weight and incorporated them into the final model evaluated. We can see in Table 2, that our model outperforms all the other model we experimented against, and we evaluate the model in an online setting in next section.

5.3 Online Evaluations

The objective of the platform is to provide personalized content. We base the online performance of the model on two metrics, the Average peek rate or Click-through-rate(CTR) across users **U**, where |**U**| = n and Average Duration Spent across users on the platform. Some statistics are shown in Table 4.

$$\text{Average Peek Rate} = \frac{1}{n}\sum_{u_i \in \mathbf{U}} \frac{\text{peeks}_{u_i}}{\text{impression}_{u_i}} \tag{14}$$

$$\text{Average Duration} = \frac{1}{n}\sum_{u_i \in \mathbf{U}} \text{duration}_{u_i} \tag{15}$$

Table 4: Statistics of the data for random 5% users

Data	Interactions (M)	Peek (M)
User Profile (28 days)	600	6
Training (7 days) [80 split]	158	1.5
Validation (7 days) [20 split]	38.7	0.38
Testing (1 day)	20.5	0.195

To measure the performance of our model online, we run two models, the Baseline Model and the Proposed Model in separate 5 buckets (5% of the traffic) and compare on the above-mentioned metrics. We choose Baseline Model, since it serves as a good baseline for serving and latency requirements for the deployment. As we see from the Figure 7 that our model improves over the baseline with a good margin, improves average duration spent by the user by +8.1% and click-through-rate or peek-rate by +2.3% over the baseline model.

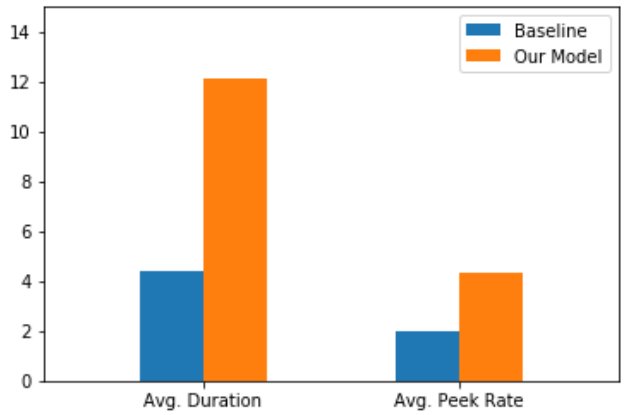

Figure 7: Comparison of the two models we put online, our model outperforms the baseline model for 5% users online

6 SYSTEM DEPLOYMENT

With the ever increasing amount of data during model training phase and accurate predictions with low-latency and high throughput requirements at online serving time, we needed a robust and easy to scale solution to meet our requirements. Existing online prediction serving systems such as Tensorflow Serving[33], Mleap[34], Clipper[35], LASER[36], Velox[37] are not designed to provide an all-together generalized, cost-efficient, highly available, scalable solution with supports for batching, config-driven traffic splitting-and-combination, explore-exploit and A/B testings, and single-click model training, updation and deployment mechanisms. To address this, we have built an in-house solution *Sigma™*, which is an end-to-end ML platform to seamlessly discover, collaborate, experiment, deploy and operate ML models at scale. Sigma is deployed on Azure Cloud using auto-scaling *Azure Kubernetes Service(AKS)*[38].

For training, we are using *Azure Distributed Data Engineering Toolkit* (AZTK)[39] to launch PySpark jobs to generate User Profiles & Glance Profiles in parquet format[40]. These training jobs are scheduled on Azure Batch Scheduler using Low Priority VMSS[41] which are cheap by $1/5^{th}$ of original price of virtual machines. PySpark trained models are serialised with Mleap format & Tensorflow trained models are serialised using SavedModel Bundle APIs[42]; stored on Azure Blob[43] to make it available for online serving.

Figure 8. demonstrates our Online Prediction Service Architecture. Prediction service has two main components :

- Model Controller(MC)
- Leaf Predictor(LP)

MC : MC has two layers- L0 and L1; where L0 layer acts like a Traffic Splitting Layer which forwards the request to the corresponding

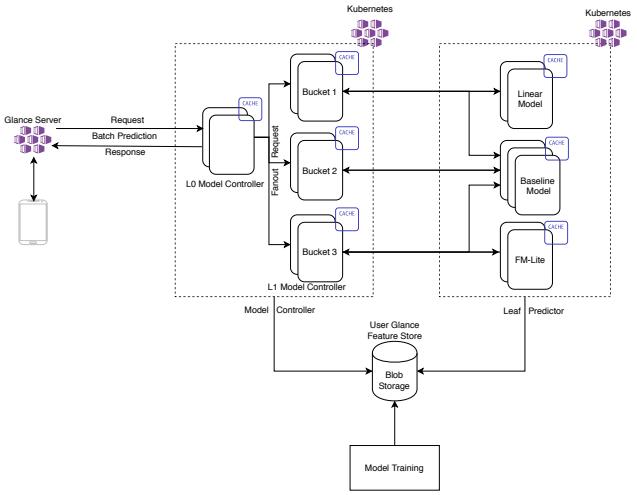

Figure 8: Serving Architecture

L1-shard based on the user-id hash. L1 consists of sharded-MC which is bucketized on basis of hash of user-ids and forwards the request to an appropriate LP; L1-MC splits the batch *parallelly* into chunks of smaller batches, which is determined by how much each LP can handle for the required capped latency requirement.

LP : LP supports serving using various libraries viz. Mleap, Tensorflow Serving. It handles reloading of model bundle and user and glance profiles into in-memory HashMap, as and when they are updated on the Azure Blob Storage. Batch-size for our model is 100 and for baseline model it is 400.

Both MC and LP are stateless Java8 SpringBoot[44] Jetty Applications and are deployed on auto-scaling AKS cluster. We use Apache Thrift[45] over HTTP for fast inter-communication between the services. Each prediction request consists of one user-id and a batch of glance-ids and the response is a list of predicted scores corresponding to each glance-id. Mleap's LeapFrame inherently gives predictions row by row even for a batched Dataset, which caused latency to grow linearly with batch-size; hence we added a Batch-LeapFrame in Mleap library to process requests parallelly. Setting timeout for request from MC to LP to a lower value than the capped latency and having a couple of additional retries for timed-outs requests further helped in lowering p99 latency in case of heavy load on one of the LP pods.

Initially, we tried Dl4j[46] Java Library for online predictions of our model in batch but it didn't scale well in our multi-threaded environment. As we decreased batch-size to a relatively small number, the thread dump logs clearly showed the synchronized blocks in Dl4j APIs as seen in Figure 9. So we shifted to Tensorflow Serving by bringing up Tf-Server in each pod and sending requests using the grpc protobuf client, that gave us throughput gains of 5x.

Using in-memory localised caching and optimized BLAS libraries we could further improve on throughput by 3x. The incoming request mean batch-size to L0 MC is around 1000 glances and the overall p99 latency of the system is 50ms and per pod of 14-cores/56-GB we are able to serve 2,100 ops/sec which translates to 150k predictions per core per second. Moreover, to decrease the production cost of serving system we deployed 2/3rd of total pods on spot vmss in one nodepool[47] and 1/3rd on dedicated vmss in another nodepool of AKS and leveraged kubernetes service label selectors[48] to serve the traffic in this hybrid vmss ecosystem. This helped in saving costs further by 40% without having any performance impact on latency of live serving traffic.

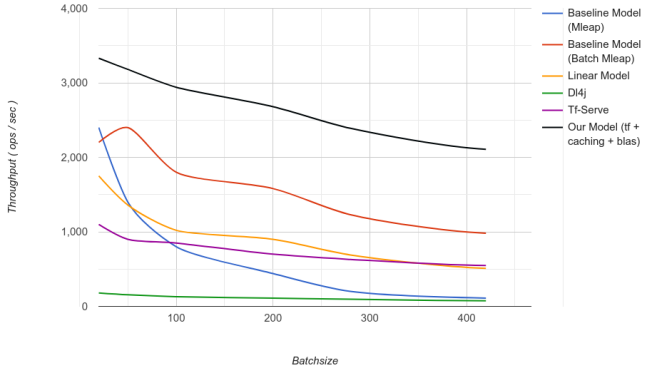

Figure 9: Comparison of variation of throughput with batch-size for multiple models with p99 latency capped at 50ms

7 CONCLUSION

In this paper we present a solution to address the challenges of multi modality of content and the ability to serve a recommendation engine at scale. The paper presents a solution that addresses each of these problems and shares our learnings of the process. We use a neural network architecture called *FM-Lite*, a computationally lighter version of factorization machines that combines cross domain user and item features. Through experiments we demonstrate how the model outperforms various other neural network models not only in performance metrics but also in significant reduction in the number of parameters that it uses. We further showcase and share our learnings on system design practices to deploy a robust, economical and scalable model while adhering to the requirements of low latency and high through-put and deploy this model in the online world on a scale of 100 M Daily active user base.

In future we will focus on iterating on the current architecture to make the recommendation system more effective by exploring cross-domain embedding for vernacular languages and introducing diversity in the feed.

ACKNOWLEDGMENTS

The authors would like to thank Prof. R. Ravi, Carnegie Mellon University for his unique inputs. We also want to thank Glance Engineering & Product teams for all the support in making this endeavour possible.

REFERENCES

[1] Jasson D.M. Rennie and Nathan Srebro. Fast Maximum Margin Matrix Factorization for collaborative prediction. In *ICML 2005 - Proceedings of the 22nd International Conference on Machine Learning*, pages 713–720, New York, New York, USA, 2005. ACM Press. doi: 10.1145/1102351.1102441. URL http://portal.acm.org/citation.cfm?doid=1102351.1102441.

[2] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted Boltzmann Machines for Collaborative Filtering. Technical report.

[3] A Survey of Collaborative Filtering Techniques. URL https://www.hindawi.com/journals/aai/2009/421425/.

[4] Michael J. Pazzani and Daniel Billsus. Content-based recommendation systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 325–341. Springer, Berlin, Heidelberg, 2007. doi: 10.1007/978-3-540-72079-9_10.

[5] Content-based Recommender Systems: State of the Art and Trends. In *Recommender Systems Handbook*, pages 73–105. Springer US, 2011. doi: 10.1007/978-0-387-85820-3_3.

[6] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, jun 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.99. URL http://ieeexplore.ieee.org/document/1423975/.

[7] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002. ISSN 09241868. doi: 10.1023/A:1021240730564. URL http://link.springer.com/10.1023/A:1021240730564.

[8] Gabriele Sottocornola, Fabio Stella, Francesco Canonaco, and Markus Zanker. Towards a deep learning model for hybrid recommendation. In *Proceedings - 2017 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2017*, pages 1260–1264. Association for Computing Machinery, Inc, 2017. ISBN 9781450349512. doi: 10.1145/3106426.3110321.

[9] Shuai Zhang, Lina Yao, Y I Tay, Aixin Sun, and Yi Tay. Deep Learning based Recommender System: A Survey and New Perspectives. Technical report, 2018.

[10] Yuanhua Lv, Taesup Moon, Pranam Kolari, Zhaohui Zheng, Xuanhui Wang, and Yi Chang. Learning to model relatedness for news recommendation. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011*. ACM Press, 2011. ISBN 9781450306324. doi: 10.1145/1963405.1963417. URL http://portal.acm.org/citation.cfm?doid=1963405.1963417.

[11] Michel Capelle, Marnix Moerland, Flavius Frasincar, and Frederik Hogenboom. Semantics-based news recommendation. In *ACM International Conference Proceeding Series*, page 1. ACM Press, 2012. ISBN 9781450309158. doi: 10.1145/2254129.2254163. URL http://dl.acm.org/citation.cfm?doid=2254129.2254163.

[12] Xuejian Wang, Lantao Yu, Kan Ren, Guanyu Tao, Weinan Zhang, Yong Yu, Jun Wang, and † Shanghai. Dynamic Attention Deep Model for Article Recommendation by Learning Human Editors' Demonstration. *KDD*, 17. doi: 10.1145/3097983.3098096.

[13] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. Embedding-based News Recommendation for Millions of Users. *KDD*, 17. doi: 10.1145/3097983.3098108. URL http://dx.doi.org/10.1145/3097983.3098108.

[14] Marnix Moerland, Michel Capelle, Frederik Hogenboom, and Flavius Frasincar. Semantics-based news recommendation with SF-IDF. In *ACM International Conference Proceeding Series*, 2013. ISBN 9781450318501. doi: 10.1145/2479787.2479795.

[15] Xavier Amatriain and Justin Basilico. Recommender Systems in Industry: A Netflix Case Study. In *Recommender Systems Handbook*, pages 385–419. Springer US, Boston, MA, 2015. doi: 10.1007/978-1-4899-7637-6_11. URL http://link.springer.com/10.1007/978-1-4899-7637-6{_}11.

[16] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and et al. Applying deep learning to airbnb search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '19, page 1927–1935, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330658. URL https://doi.org/10.1145/3292500.3330658.

[17] Chuhan Wu, Fangzhao Wu, Mingxiao An, Jianqiang Huang, Yongfeng Huang, and Xing Xie. NPA: Neural News Recommendation with Personalized Attention. 19, 2019. doi: 10.1145/3292500.3330665. URL https://doi.org/10.1145/3292500.3330665.

[18] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural Collaborative Filtering *. 2017. doi: 10.1145/3038912.3052569. URL http://dx.doi.org/10.1145/3038912.3052569.

[19] Paul Covington and Jay Adams. Deep Neural Networks for YouTube Recommendations. doi: 10.1145/2959100.2959190. URL http://dx.doi.org/10.1145/2959100.2959190.

[20] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah Google Inc. Wide & Deep Learning for Recommender Systems. Technical report. URL http://tensorflow.org.

[21] Ali Elkahky, Yang Song, Xiaodong He, and User Modeling. A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems General Terms Keywords User Modeling; Recommendation System; Multi-View Learn-ing; Deep Learning. doi: 10.1145/2736277.2741667. URL http://dx.doi.org/10.1145/2736277.2741667.

[22] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[23] Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder, 2018.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova Google, and A I Language. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Technical report. URL https://github.com/tensorflow/tensor2tensor.

[25] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to Fine-Tune BERT for Text Classification? Technical report.

[26] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, jan 2019. doi: 10.1093/bioinformatics/btz682. URL http://arxiv.org/abs/1901.08746http://dx.doi.org/10.1093/bioinformatics/btz682.

[27] Iz Beltagy, Kyle Lo, and Arman Cohan. SciBERT: A Pretrained Language Model for Scientific Text. pages 3613–3618, mar 2019. URL http://arxiv.org/abs/1903.10676.

[28] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL http://arxiv.org/abs/1609.08144.

[29] Samuel R Bowman, Gabor Angeli, Christopher Potts, Christopher D Manning, and Stanford Linguistics. A large annotated corpus for learning natural language inference. Technical report. URL http://aclweb.org/aclwiki/index.php?

[30] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. pages 3980–3990, aug 2019. URL http://arxiv.org/abs/1908.10084.

[31] Gregory Koch. Siamese Neural Networks for One-Shot Image Recognition. Technical report, 2015.

[32] Zhenghua Xu, Thomas Lukasiewicz, Cheng Chen, Yishu Miao, and Xiangwu Meng. Tag-aware personalized recommendation using a hybrid deep model. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 3196–3202. International Joint Conferences on Artificial Intelligence, 2017. ISBN 9780999241103. doi: 10.24963/ijcai.2017/446.

[33] Tensorflow serving. URL https://www.tensorflow.org/tfx/guide/serving.

[34] GitHub - combust/mleap: MLeap: Deploy Spark Pipelines to Production. URL https://github.com/combust/mleap.

[35] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 613–627, USA, 2017. USENIX Association. ISBN 9781931971379.

[36] Deepak Agarwal, Bo Long, Jonathan Traupman, Doris Xin, and Liang Zhang. Laser: A scalable response prediction platform for online advertising. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, page 173–182, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450323512. doi: 10.1145/2556195.2556252. URL https://doi.org/10.1145/2556195.2556252.

[37] Daniel Crankshaw, Peter Bailis, Joseph E. Gonzalez, Haoyuan Li, Zhao Zhang, Michael J. Franklin, Ali Ghodsi, and Michael I. Jordan. The missing piece in complex analytics: Low latency, scalable model management and serving with velox. *CoRR*, abs/1409.3809, 2014. URL http://arxiv.org/abs/1409.3809.

[38] Azure Kubernetes Service (AKS) | Microsoft Azure. URL https://azure.microsoft.com/en-in/services/kubernetes-service/.

[39] GitHub - Azure/aztk: AZTK powered by Azure Batch: On-demand, Dockerized, Spark Jobs on Azure. URL https://github.com/Azure/aztk.

[40] Apache Parquet. URL https://parquet.apache.org/.

[41] Run workloads on cost-effective low-priority vms - azure batch, . URL https://docs.microsoft.com/en-us/azure/batch/batch-low-pri-vms.

[42] Using the SavedModel format | TensorFlow Core. URL https://www.tensorflow.org/guide/saved_model.

[43] Introduction to Blob (object) storage - Azure Storage | Microsoft Docs, . URL https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction.

[44] GitHub - spring-projects/spring-boot: Spring Boot. URL https://github.com/spring-projects/spring-boot.

[45] Apache thrift - home. URL https://thrift.apache.org/.

[46] DeepLearning4j. URL https://deeplearning4j.org//.

[47] Adding spot nodepool on AKS. URL https://docs.microsoft.com/en-us/azure/aks/spot-node-pool.

[48] Kubernetes Label Selectors. URL https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors.