Dynamic Heterogeneous Graph Neural Network for Real-time Event Prediction

Wenjuan Luo[1], Han Zhang[1], Xiaodi Yang[1], Lin Bo[1], Xiaoqing Yang[1]
Zang Li[1], Xiaohu Qie[2], Jieping Ye[1]
[1]AI Labs, Didi Chuxing, Beijing, China
[2]Didi Chuxing, Beijing, China
{luowenjuan,arsenezhang,yangxiaodi_i,bolinlin,yangxiaoqing,lizang,tiger.qie,yejieping}@didiglobal.com

ABSTRACT

Customer response prediction is critical in many industrial applications such as online advertising and recommendations. In particular, the challenge is greater for ride-hailing platforms such as Uber and DiDi, because the response prediction models need to consider historical and real-time event information in the physical environment, such as surrounding traffic and supply and demand conditions. In this paper, we propose to use dynamically constructed heterogeneous graph for each ongoing event to encode the attributes of the event and its surroundings. In addition, we propose a multi-layer graph neural network model to learn the impact of historical actions and the surrounding environment on the current events, and generate an effective event representation to improve the accuracy of the response model. We investigate this framework to two practical applications on the DiDi platform. Offline and online experiments show that the framework can significantly improve prediction performance. The framework has been deployed in the online production environment and serves tens of millions of event prediction requests every day.

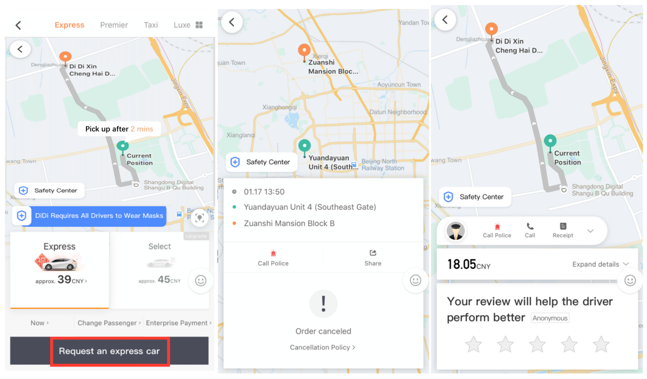
CCS CONCEPTS

• **Information systems** → **Spatial-temporal systems**; *Data mining*; • **Computing methodologies** → **Neural networks**.

KEYWORDS

Real-time event embedding; Heterogeneous graph neural networks; Dynamic graph embedding

ACM Reference Format:
Wenjuan Luo[1], Han Zhang[1], Xiaodi Yang[1], Lin Bo[1], Xiaoqing Yang[1] and Zang Li[1], Xiaohu Qie[2], Jieping Ye[1]. 2020. Dynamic Heterogeneous Graph Neural Network for Real-time Event Prediction. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403373



(a) PreView and Request (b) Cancel_Order (c) Finish_Order

Figure 1: Illustration of Ride-hailing Events

1 INTRODUCTION

The development of ride-hailing platforms makes people's travel much more convenient. In ride-hailing platforms such as Uber and DiDi, once a user provides the pick-up and drop-off locations, the ride-hailing APP pops up a PreView page to show the route, service type and estimated price (Figure 1a). We use the term, **PreView**, to refer to the event in which a PreView page is viewed by a user. The user can then decide whether to click the Request button to request a ride or not. Once the user makes a request, the platform will assign an express or premier car according to the user's choice. We name the event triggered by the user as **Request**. Before the driver arrives at the pick-up location, the passenger can cancel the order if his/her plan has changed as depicted in Figure 1b. This event is called **Cancel_Order**. Otherwise, the passenger will successfully arrive at the destination and pay for the trip as shown in Figure 1c, which is named as **Finish_Order**.

In the advertising field, it is crucial to determine which impressions to bid on and the bid amount by predicting the probability of user response such as click and conversion given an ad-impression. Similarly in a ride-hailing platform, the prediction of the PreView event conversion rate is of great importance for down-stream strategies such as supply and demand regulation, transportation scheduling, carpool route matching, etc. In this paper, we study how to model a PreView event and estimate the probability of a passenger clicking the Request button.

Figure 2 illustrates the major events involved in one passenger's ride-hailing decision process, where the colored circles denote various user actions. Specifically, in this paper, we utilize the word POI (Point Of Interest) to denote all the possible pick-up and drop-off points in a map. As shown in the figure, different user actions occur simultaneously at different POIs.

Typically, whether or not the passenger clicks the Request button is affected by many factors, such as the spending habits of the user, the weather condition and the supply and demand conditions at that time. While some explicit characteristics (e.g. the distance between current location and pick-up location, weather and time) can be extracted from various data sources, other intrinsic characteristics, such as the willingness to wait and pay, and satisfaction with the route are much harder to extract. One solution is to introduce proxy signals from historical and contemporary observations such as historical transaction-related actions, and real-time events that occur in the surrounding environment.

For example, a passenger may have completed multiple orders before current PreView; we can use these historical signals for capturing implicit features such as user preferences on service types, willingness to pay and so on. Specifically, the passenger is more willing to issue the request on the PreViews similar to those that have been converted to Requests. At the same time, negative signals can also be extracted from those PreViews that have not been converted. To be able to calculate similarities between PreViews, we propose to use embeddings, low-dimensional vector representations learned from historical data. In addition, we also want our embeddings to capture the supply and demand conditions in the surrounding area at that time. To this end, we propose to investigate events that occurred simultaneously in the vicinity. For example, if a lot of demands around the region have not been fulfilled, then the current supplies and demands are unbalanced; if there are many cancellations (Cancel_Order) around, then the road conditions are more congested, or the expected waiting time of arrival is long. Obviously, these historical data and real-time observations can be used as input factors for the prediction model.

The above-mentioned events, whether historical or ongoing, have various relationships with the current event to be analyzed, and the degree of relevance to the current event is also different. It is natural to use heterogeneous graphs to represent these events and relationships. One key advantage of heterogeneous graph representations is that graph embedding [5, 8, 18, 23, 25, 29, 31] can be used to encode and represent the semantics in these graphs. In recent years, graph embedding has achieved great success in many fields [28] [7]. For example, embedding items, users or preferences has been used to improve CTR predictions, searching or ranking models [19, 30, 32, 33]. However, it is still challenging to embed real-time events in dynamically changing heterogeneous graphs:

1 **For each new event the system needs to dynamically construct a graph around the event.** Each time a new event occurs, it is necessary to construct a graph dynamically by collecting the historical events of the corresponding passengers and the events that are happening in the surrounding area.
2 **The entities and relationships in the graph are heterogeneous.** There exist different types of events (e.g., PreView, Request, etc.) and relationships (e.g., the same transaction, the same passenger, the same origin, etc.) between events.
3 **Different entities and relationships have different effects on the events of interest.** For example, historical events happened at similar origin and destination are more relevant to the current event. And events within the same transaction are highly relevant to each other.
4 **Large-scale real-time event modeling.** With tens of millions of events being generated every day, the system needs to be able to construct a graph for each event in real time and generate corresponding representations efficiently.

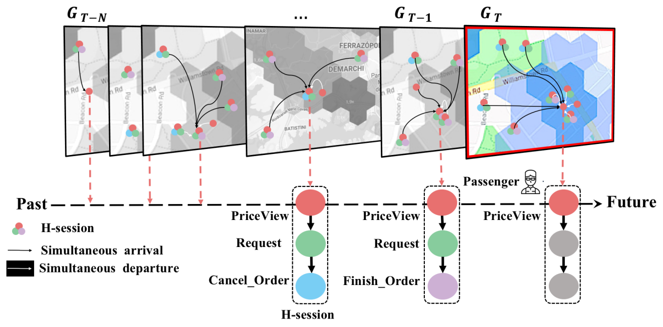


Figure 2: Typical Ride-hailing Event Flows: Red, green, blue and purple nodes represent PreView, Request, Cancel_Order and Finish_Order, respectively. A passenger may have multiple transactions on the platform, involving different events. In addition, at the same time as an event, there may be many h-sessions and events happening in the surrounding area.

Instead of embedding items during the training phase, we propose a novel learning framework to generate event representations in real time so that we can capture live changes in user behavior and the surrounding environment. The embedding of each entity is generated in an inductive manner based on graph neural networks (GNNs) [13]. Note that entities discussed in this paper include events, items, user actions, etc. Specifically, our method mainly consists of the following steps: 1. construct a dynamic heterogeneous graph for each event; 2. generate event embeddings using our proposed embedding algorithm for heterogeneous graphs; 3. make real-time predictions based on entity embeddings. Before we get into the details, we introduce a concept called heterogeneous session (denoted as h-session). An h-session is a collection of chronologically related heterogeneous events for the same transaction. In our case, in a ride-hailing transaction, the user may trigger certain events like Request, Finish_Order and Cancel_Order after the PreView event is issued. These events, which belong to the same h-session, describe a user's complete transaction behavior on the ride-hailing platform.

Based on the workflow of constructing heterogeneous graphs for events, we propose a novel graph learning algorithm called Real-time Event Graph Neural Network (REGNN) to generate event embedding. We summarize our main contributions as follows:

1 **Propose a workflow for constructing a heterogeneous graph centered on the event for each real-time event**

Table 1: PreView Event Notations

Symbols	Definitions and Descriptions
p	a passenger entity
o	an entity for pick-up (origin)
d	an entity for drop-off (destination)
$HetG_{x,T}$	Heterogeneous Graph at time T, about entity x , e.g. p, o, d
y_T	label of event at time T
P_T	PreView event at time T
R_T	Request event at time T
C_T	Cancel_Order event at time T
F_T	Finish_Order event at time T
E_T	dynamic graph embedding for event at time T
α	the attention weight of the embedding
θ	the model parameters
$\bigoplus$	the concatenation of two vectors
$\circ$	the Hadamard product of two matrices

that needs to be predicted. The process explores events from various perspectives, including temporal and spatial dimensions.

2 **Propose an attention based GNN model to embed the dynamically generated graph for each event.** A novel multi-layered graph structure and different graph attention [26] operators at each level are proposed to encode events and their relationships.

3 **Experiments show that a recurrent neural network can further capture more temporal dependencies among events.** Additionally, in our online implementation, past event representations are stored, and then will be used when computing the temporal dependencies for subsequent events.

4 **By applying this framework to two practical applications (see SUPPLEMENT) on the DiDi platform, we show that the framework is widely applicable to various prediction problems of real-time events.** The framework has been deployed in DiDi's online production environment and serves tens of millions of event prediction requests every day.

2 PROBLEM FORMULATION

In this section, we introduce the definition of PreView related concepts and use dynamic heterogeneous graphs to formalize the problem of event embedding and event sequence learning. Specifically, a heterogeneous graph is dynamically created for each real-time event that needs to be predicted. The graph contains the events in related h-sessions and other related entities. The edges of the graph represent various complex relationships between the vertices in the graph, such as sequential relationships, spatial position relationships, and other logical relationships, etc. Table 1 summarizes all notations used in the paper.

2.1 PreView Conversion Prediction

As described in [31], a graph is modeled as $G = (V_G, E_G, O_V, R_E)$ with a node mapping function $V_G \rightarrow O_V$ and an edge mapping function $E_G \rightarrow R_E$, where each node in V_G belongs to a type in O_V, and each edge in E_G belongs to a type in R_E. G is a homogeneous graph if the number of node types $|O_V| = 1$ and the number of edge types $|R_E| = 1$. In a heterogeneous graph,the number of node types or the number of edge types is greater than one.

Definition 2.1 (PreView Conversion Prediction). Given a PreView event $P_T = (p, o, d, T)$ at time T, from origin o to destination d, our goal is to estimate the probability y_T of the user p to trigger event *Request* by embedding a series of dynamic heterogeneous networks in history $[G_{P_T}, G_{P_{T-1}}, \ldots, G_{P_{T-N+1}}]$ where G_{P_t} denotes the dynamic heterogeneous graph for event P_t, for $t = T - N + 1, ..., T$.

Given a formulated event P, G_P consists of different kinds of events or items. Our framework aims to embed the heterogeneous graph into a low dimensional vector $E \in \mathbb{R}^{dim}$, where dim is the embedding dimension size. In particular, the embedding network needs to learn the following function:

$$\mathcal{F}(G_P) \rightarrow E \tag{1}$$

Given a time series of events, with embeddings obtained from Equation (1), the top layer in our model aims to learn a model $\mathcal{G}_\theta$ with parameters θ as follows:

$$\mathcal{G}_\theta \ : \ [E_{T-N+1}, ..., E_T] \quad \rightarrow \quad y_T \tag{2}$$

where y_T is prediction target for the event at timestamp T, E_t denotes the embedding vector for the event at timestamp t, and N indicates the sequence length of the time series.

3 METHODOLOGY

In the following, we introduce the proposed framework for real-time event embedding. The core idea is to analyze the various attributes of the event, and based on these attributes, we then explore other events that may have an impact or have similar semantics. In our case, for simplicity, we only consider the most relevant attributes of a PreView event: passenger, timestamp, origin and destination. According to our previous discussion, from the passenger perspective we can obtain information from the past behavior events; and from the two perspectives of the origin and destination, we can obtain spatial representation by observing and synthesizing the event information around these two locations. Specifically, our real-time event embedding workflow is shown in Figure 3.

- First, Given the current PreView event $P_T = (p, o, d, T)$, we can generate a heterogeneous graph according to the following process:
  - Passenger perspective: we select the passenger's latest N_p PreView events within one week before time T, together with their corresponding Request events (if any), Finish_Order events (if any), Cancel_Order events (if any). We create corresponding neighbor nodes in the graph for these events. This subgraph about passenger p is denoted as $HetG_{p,T}$.
  - Origin and destination perspective: from all the PreView events happening simultaneously, we select PreView events who share the same origin of P_T within x minutes (the time length) before time T, together with their corresponding Request, Finish_Order and Cancel_Order events (if any). These events are added to the graph as an origin point

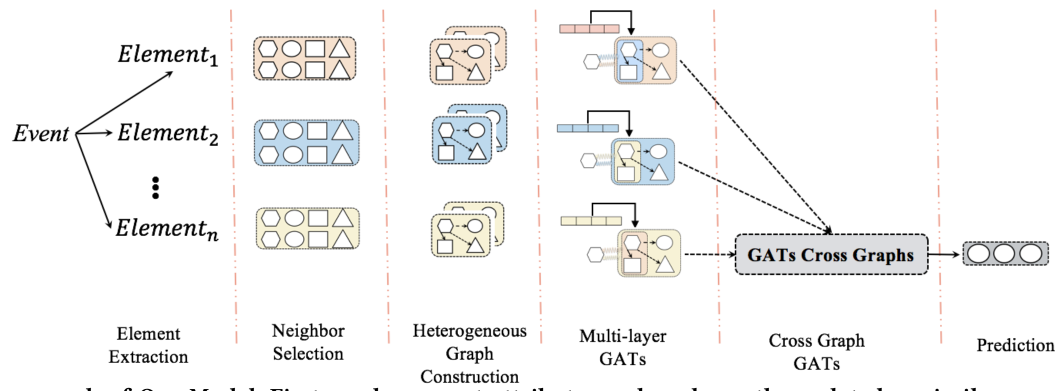


Figure 3: Framework of Our Model. First, analyze event attributes and explore other related or similar events. Afterwards, heterogeneous subgraphs are built carefully and then GATs at different levels are performed for down-stream tasks.

subgraph $HetG_{o,T}$. On the other hand, those PreViews of which the destination is P_T's origin point within x minutes (the time length) before time T, together with their corresponding Request, Finish_Order and Cancel_Order events (if any) form the destination subgraph $HetG_{d,T}$.
– Note that when serving online, to aggregate the spatiotemporal information of historical PreViews, we learn the hidden state of the historical event sequence through RNN and save the hidden state in a carefully designed key-value store, so that the next sequence of event sequences can be quickly predicted and updated.

- Second, according to the relationship between these events and the current event P_T, add the corresponding types of edges. For example, the relationship that belongs to the same h-session, the sequence relationship between the h-sessions, etc.
- Afterwards, with the constructed heterogeneous subgraphs, REGNN is performed to generate P_T's real-time event embedding. The details will be introduced in Section 3.2.
- Finally, the generated event embedding is used as input features for the down-stream prediction tasks.

Figure 4 shows the detail of the PreView event modeling. In general, compared to previous GraphSAGE [10] like methods, our framework is a comprehensive multi-layer model, in which the bottom three layers are composed of graph attentions [26] (for heterogeneous neighborhood feature aggregation inside each h-session and across different kinds of h-sessions), and the top layer is composed of GRUs [12] for sequential structure learning.

3.1 Dynamic Heterogeneous Graph Construction

In our framework, the key idea of generating embedding for PreView is to aggregate information from basic features (i.e., estimated time of arrival, estimated price, etc.) and P_T's neighbors.

3.1.1 Dynamic PreView Graph Generation. As described in Figure 2, the embedding of a PreView event P_T is influenced by three different heterogeneous subgraphs. Formally, the dynamic heterogeneous graph G_{P_T} for P_T is given as follows,

$$G_{P_T} = HetG_{p,T} + HetG_{o,T} + HetG_{d,T} \tag{3}$$

where $HetG_{p,T}$, $HetG_{o,T}$ and $HetG_{d,T}$ are passenger subgraph, origin subgraph and destination subgraph, and + denotes the graph join operators, defined as follows: Suppose $G = G_1 + G_2, G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, G has the node set $V_1 \cup V_2$, and the edge set $E_1 \cup E_2$. $HetG_{p,T}$, $HetG_{o,T}$ and $HetG_{d,T}$ are constructed by the following procedure:

Graph construction inside h-session (intra-session) First, we connect events in the same session together to form a subgraph. For example, a successful transaction includes events PreView, Request and Finish_Order. For more efficient graph calculations and subsequent snapshots, we can treat this h-session subgraph as a virtual node and explore the relationships between these virtual nodes.

Graph construction across h-sessions (inter-session) Moreover, to analyze the impact from previous h-sessions to the target PreView, edges are connected between previous h-Sessions to the target PreView. However, h-sessions in different subgraphs play different roles in analyzing the status of the target PreView, so the types of edges are different, i.e. "h-session$_p \xrightarrow{p} P_T$", "h-session$_o \xrightarrow{o} P_T$", "h-session$_d \xrightarrow{d} P_T$".

3.2 PreView Event Graph Embedding

As illustrated in Figure 4, P_T represents the PreView event at time T. We build graphs around P_T with the most recent N h-sessions. Besides, following the methods discussed earlier, we collect events that occurred within the last 10 minutes (P_T) around the origin and destination points and build an origin and a destination subgraph.

Specifically, we utilize different graph operators for 3 different levels to perform dynamic graph embedding by following the sequential and semantic relationships between events.

GATs inside h-session: Within one h-session depicted in Figure 4, we perform GATs over vertices to learn knowledge from heterogeneous events. Specifically, for one h-session, the operation GAT_I mechanism is described in Figure 5, and formulated as

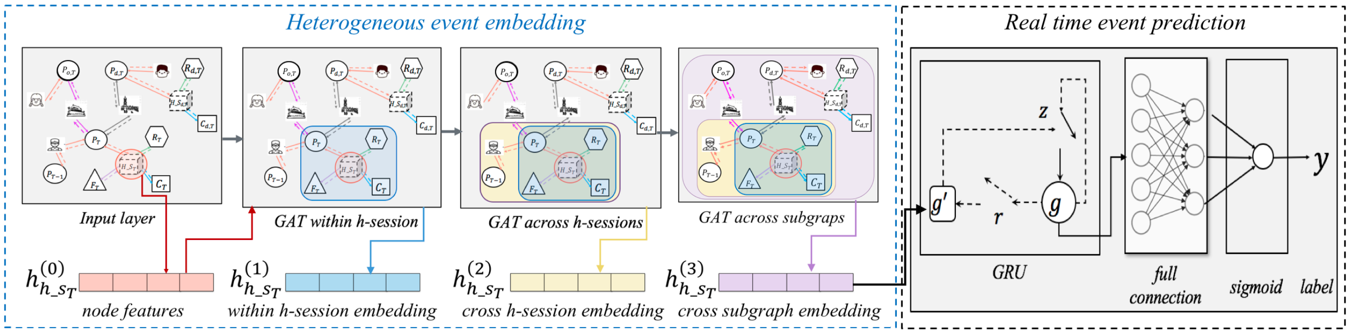


Figure 4: Architecture of the PreView Event Embedding. Vertices of different shapes represent heterogeneous events and edges of different colors are heterogeneous edges. H_S_T denotes an h-session at timestamp T consisting of P_T, R_T, F_T, C_T. For H_S_T, graph operators at different granularities (GAT within h-session, GAT across h-sessions within the same subgraph and GAT across subgraphs) are performed to update H_S_T's hidden states, shown as $h^{(\cdot)}_{h\_s_T}$.

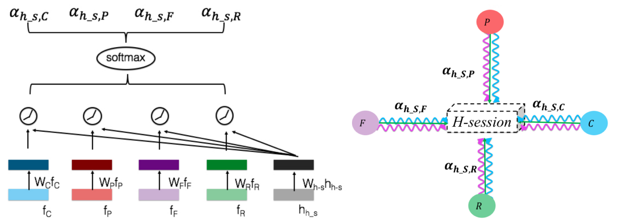


Figure 5: GAT within h-session, f_i denotes the node features for event i with i standing for P, R, F and C. H-session embedding is updated through a multi-head attention based on P, R, F and C.

Equation (4).

$$h^{(1)}_{h\_s} = \oplus^{K}_{k=1}\sigma\Big(\sum_{i\in O_V}\alpha^{k}_{h\_s,i}W^{k}_{i}f_i\Big)$$
$$\alpha^{k}_{h\_s,i} = \frac{\sigma(W^k\cdot[W^k_h h^{(0)}_{h\_s}\oplus W^k_i f_i])}{\sum_{j\in O_V}\sigma(W^k\cdot[W^k_h h^{(0)}_{h\_s}\oplus W^k_j f_j])} \quad (4)$$

where $\oplus$ denotes concatenation, O_V is the set of different types of events within the same h-session, K is the total number of heads, $h^{(1)}_{h\_s}$ denotes hidden states of h-session after GAT inside h-sessions, and $h^{(0)}_{h\_s}$ is the initial state of the h-session, which is initialized by the node features of PreView events. $\alpha^{k}_{h\_s,i}$ is the attention weight of the k-th head computed through the attention mechanism between h_s and event i, W^k is the attention matrix in the k-th head. As initial event node features are heterogeneous in nature, W^k_h, W^k_i respectively work as transformation matrices of h-session and event i (i.e., P, R, F, C) with initial node features as f_i in the k-th head.

GATs across h-sessions: These operators perform attention based aggregation across h-sessions on the graph. For h-sessions in different subgraphs, the graph attention works as follows:

$$h^{(2)}_{h\_s_{T,p}} = GAT_p(h^{(1)}_{h\_s_{T-t,p}}), \forall t, 0 <= t <= N_p$$
$$h^{(2)}_{h\_s_{T,o}} = GAT_o(h^{(1)}_{h\_s_{T-t,o}}), \forall t, 0 <= t <= N_o \quad (5)$$
$$h^{(2)}_{h\_s_{T,d}} = GAT_d(h^{(1)}_{h\_s_{T-t,d}}), \forall t, 0 <= t <= N_d$$

where N_p, N_o and N_d denote the number of different timestamps in passenger subgraph, origin subgraph and destination subgraph, respectively. Moreover, $h^{(1)}_{h\_s_{T-t,p}}$, $h^{(1)}_{h\_s_{T-t,o}}$ and $h^{(1)}_{h\_s_{T-t,d}}$ are hidden states after GAT inside the h-session at time $T-t$ of passenger subgraph, origin subgraph and destination subgraph. As indicated in the equation, different GATs are trained for different subgraphs. Note that subscript t starts from 0, hence self attention is also employed in the GATs across h-sessions. We formulate the attention mechanism within passenger graph GAT_p as Equation (6), while the attentions for other subgraphs (GAT_o, GAT_d) are computed similarly.

$$h^{(2)}_{h\_s_{T,p}} = \oplus^{K_p}_{k=1}\sum^{N_p}_{t=0}\alpha^{k}_{t,T}W^{p,k}_{l}h^{(1)}_{h\_s_{T-t,p}}$$
$$\alpha^{k}_{t,T} = \frac{\sigma(W^{p,k}\cdot[W^{p,k}_l h^{(1)}_{h\_s_{T,p}}\oplus W^{p,k}_l h^{(1)}_{h\_s_{T-t,p}}])}{\sum^{N_p}_{j=0}\sigma(W^{p,k}\cdot[W^{p,k}_l h^{(1)}_{h\_s_{T,p}}\oplus W^{p,k}_l h^{(1)}_{h\_s_{T-j,p}}])} \quad (6)$$

where $h^{(1)}_{h\_s_{T-t,p}}$ denotes the hidden state of h-session after GAT inside h-session at time $T-t$ for subgraph of passenger p, and $h^{(2)}_{h\_s_{T,p}}$ aggregates information of all its neighbors within subgraph of passenger p. K_p is the number of heads in GAT_p. $W^{p,k}$ denotes the attention matrix for the k-th head in the subgraph, while $W^{p,k}_l$ denotes the linear transformation matrix before attention.

GATs across subgraphs: Afterwards, the global attention is carried out as :

$$h^{g}_{P_T} = h^{(3)}_{h\_s_T} = GAT_g(h^{(2)}_{h\_s_{T,p}}, h^{(2)}_{h\_s_{T,o}}, h^{(2)}_{h\_s_{T,d}}) \quad (7)$$

where $h^{(3)}_{h\_s_T}$ is the final event embedding for h-session at time T, and the final event embedding of PreView P_T after GAT cross subgraphs $h^{g}_{P_T}$ is updated accordingly. GAT_g aggregates information at the global level for heterogeneous subgraphs. Concretely, the attention mechanism for global attention GAT_g for PreView event is designed as:

$$h^{(3)}_{h\_s_T} = \oplus_{k=1}^{K_g} \sum_{i \in O_G} \alpha^k_{i,T} W_l^{g,k} h^{(2)}_{h\_s_{T,i}}$$
$$\alpha^k_{i,T} = \frac{\sigma(W^{g,k} \cdot [W_l^{g,k} h^{(2)}_{h\_s_{T,p}} \oplus W_l^{g,k} h^{(2)}_{h\_s_{T,i}}])}{\sum_{j \in O_G} \sigma(W^{g,k} \cdot [W_l^{g,k} h^{(2)}_{h\_s_{T,p}} \oplus W_l^{g,k} h^{(2)}_{h\_s_{T,i}}])} \tag{8}$$

where O_G denotes the set of different kinds of heterogeneous subgraphs, K_g is the number of heads in global attention, $W^{g,k}$ denotes the global attention matrix of head k, and $W_l^{g,k}$ is the transformation matrix in global attention. As indicated in the equation, all the other subgraphs compute their attentions with respect to the passenger graph. Furthermore, self attention is also conducted to the passenger graph.

3.3 Event Time-series Embedding

We leverage the recurrent neural networks (RNNs) [11] to model the temporal dependency between user past PreView events. Here we utilize the GRU to learn the sequential embeddings as follows,

$$\begin{aligned} z_T &= \sigma(W_z \cdot E_T \oplus U_z \cdot h_{T-1}) \\ r_T &= \sigma(W_r \cdot E_T \oplus U_r \cdot h_{T-1}) \\ \tilde{h_T} &= tanh(W \cdot E_T \oplus U(r_T \circ h_{T-1})) \\ h_T &= (1 - z_T) \circ h_{T-1} + z_T \circ \tilde{h_T} \end{aligned} \tag{9}$$

where E_T is the final embedding of event after global attention at time T i.e., $h^g_{P_T}$ in Equation (7), and $\oplus$ denotes concatenation, $\circ$ means Hadamard product, $h_T \in \mathbb{R}^{dim}$ is the output hidden state at time T, while W, U are parameters to be learned.

3.4 Objective And Model Training

Given the embeddings generated in the previous sections, we minimize the following loss function for the PreView conversion rate prediction problem:

$$L = -\sum_{u \in U} \sum_{t=1}^{T} y^u_T log(\sigma(\theta \cdot h^u_T)) - (1 - y^u_T) log(1 - \sigma(\theta \cdot h^u_T)) + \lambda ||\theta||_2 \tag{10}$$

where y^u_T denotes the label for user u at time T, and h^u_T denotes the embedding for user u at time T as Equation (9), σ denotes the sigmoid function, θ represents the parameters to learn and $\lambda||\theta||_2$ is the $L2$ regularization. Note that in our framework, the loss functions are optimized using gradient descent.

4 EXPERIMENTS

In this section, we evaluate our framework using real-world datasets. We first show offline experiment results and then present online performance of our production framework. In addition, we conduct extensive experiments to compare the influence of different subgraphs and also analyze hyper parameters. Besides, we generalize our methodology to another real world problem in Section A.

4.1 Offline Experiment Results

In this section, we present the details of offline experiments compared to SOTA baselines.

4.1.1 Dataset And Experimental Setup. Our training dataset is constructed through a random selection of real-world PreView events with conversion labels from online log of one week, and test dataset is randomly selected in the same way of the following week. We provide the details of the dataset in Table 2.

Experiment Setup: We use Adam [14] with learning rate at 0.001 for all models. The numbers of heads in multi-head attentions are all set to 3. The number for hidden dimension of GRU in our method is set to 100. The dimensions of DNN Layers are set to $128 \times 64 \times 32$ before the output layer. For PreView task, we take AUC(Area Under ROC Curve) and ACC(Classification ACCuracy) as the metric for performance of models.

Table 2: Datasets Description For PreView Event Conversion Prediction

dataset	TrainSize	TestSize	FeatureSize	Classes
PreView	652266	298649	64	2

4.1.2 Competitors. We compare our framework to the state-of-the-art CTR methods (DIN [33], DIEN [32], MIMN [19]) in the scenario of sequential user behavior modeling, as well as a state-of-the-art session-based GNN (SRGNN [30]) method for recommendation. Here we denote our framework as REGNN.

DNN utlizes a multi-layer DNN for prediction. It uses sum pooling to integrate users' historical embeddings. **DIN** is an early work for CTR which applies attention mechanism to combine the representation of users' sequential behaviors and target item. **DIEN** integrates GRU with attention mechanism to capture the evolution of users' interests and proposes an auxiliary loss from negative sampling for better embedding learning. Specifically, we omit the trick of negative sampling as MIMN[19]. **MIMN** uses a memory-based framework because of improvements on NTM (Neural Turing Machine [6]) for user's interest learning. **SRGNN** proposes a session-based GNN to model session information and generate recommendations, which is the SOTA GNN-based method for session based recommendation.

To fit our task to the above competitors, there exist two problems: I) These methods aims to embed homogeneous items and II) these methods adopt item ids to generate item embedding through neural networks, thus they can not generate new item embedding in the inference stage. To solve these problems, we utilize the node features of PreView events as item embedding in these methods, and the passenger's historical PreView sequence works as user behavior sequence in these models.

Table 3: Performance Of Different Methods On Preview Event Prediction

Method	AUC(mean ± std)	ACC(mean ± std)
DNN	0.6854 ± 0.0008	0.7025 ± 0.0003
DIN	0.6817 ± 0.0001	0.7017 ± 0.0003
DIEN	0.6769 ± 0.0005	0.7027 ± 0.0003
MIMN	0.6895 ± 0.0002	0.7024 ± 0.0002
SRGNN	0.6901 ± 0.0008	0.7028 ± 0.0003
REGNN	**0.7684± 0.0006**	**0.7369± 0.0006**

4.1.3 Offline Performance. Specifically, we give the experimental results in Table 3, from which we can observe: **REGNN achieves the best results on AUC and ACC.** This experiment shows that our dynamic heterogeneous graph embedding framework can effectively help the prediction model improve accuracy. As REGNN incorporates different kinds of user events together with the complex relationships between events, various hidden information (e.g., traffic condition) is efficiently encoded into the model. In general, our model obtains 8.30% increments in AUC for PreView dataset compared to the DNN baseline. This verifies the effectiveness of our framework.

SRGNN achieves better performance than DNN, DIN, DIEN and MIMN. SRGNN adopts GNN for session graph embedding, which construct a homogeneous graph within one session and utilizes GRU for time-series prediction. The better performance of SRGNN than the rest of baselines indicates that SRGNN better fits the PreView event embedding task compared to CTR methods. However, our REGNN differs from SRGNN in three aspects: First, SRGNN generates homogeneous graphs for each session, yet REGNN constructs heterogeneous graphs for each event. Second, SRGNN only models the sequential information as edges for session graphs, in addition, REGNN still utilizes the heterogeneous edges between different types of events and events from different subgraphs. Third, the GNN in SRGNN is a one-layer GCN while REGNN learns event embedding through multi-layer GATs.

4.1.4 Ablation Study. In this section, we study the influence of different modules in our framework. As shown in Table 4, different subgraphs bring different degrees of improvement to the performance of our model. For PreView event conversion prediction task, compared to POI subgraphs, the Passenger subgraph embedding plays a more important role, which is in line with our intuition. Compared with the surrounding real-time traffic, together with supply and demand condition, the similar response of users to similar events may be a more important factor for our prediction task, i.e., similar users of similar habits may share similar request patterns. Meanwhile, as shown in the table, the lack of the *o subgraph* (origin subgraph) contributing a 0.8% decrement to our model indicates that passengers at the same time and space may make similar request decisions. Another observation is that the origin subgraph (*o subgraph*) contributes more information than the destination subgraph (*d subgraph*). This is also in line with our expectations, because the origin subgraph describes the traffic condition at the same time and space, while the destination subgraph depicts the potential supply and demand in the near future. Hence, the origin subgraph is better in reflecting the demand for ride-hailing in the surrounding area. Nevertheless, the destination subgraph can also capture some hidden useful information and help improve the model.

4.2 Online Results

Before deploying to production, we conduct offline training procedure for PreView dataset based on online logs of one week, and testify the model on online logs of the next week. When serving online, our model utilizes Redis [21] to store the last 7-day events, and generates embeddings at real-time for down-stream tasks.

Table 4: Influence Of Different Types Of Graphs On PreView Dataset

Model	AUC	Decrement
REGNN(with all subgraphs)	**0.7684**	-
REGNN(*w/o o subgraph*)	0.7599	0.0085
REGNN(*w/o d subgraph*)	0.7624	0.0060
REGNN(*w/o p subgraph*)	0.7098	**0.0586**

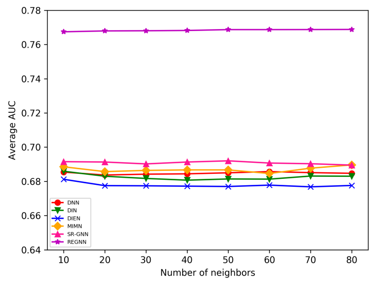
Regular Offline Training: Our framework is scalable and can be used for big data training. Before deploying to production, we randomly select 20 million PreView events within 1 week, and generate their corresponding heterogeneous subgraphs from the historical data in previous 7 days before the PreView events, i.e., we use 2-week log for heterogeneous graph construction. Specifically, we preprocess the passenger historical behaviors, and the historical information of POIs. As mentioned before, historical heterogeneous subgraphs consist of the latest 30 PreView events, and their corresponding follow-up events, i.e., Request events, Cancel_Order events and Finish_Order events. In light of offline evaluation results, we compare the performance on PreView event conversion prediction in the following week after the training week.

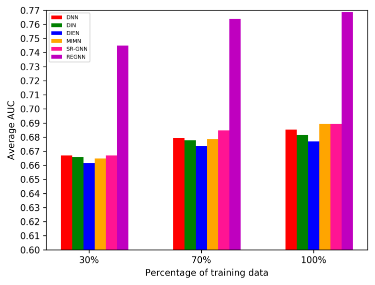
Implementation For Online Serving: The online embedding service is event driven. As mentioned previously, we design different Key-Value data storage for different subgraphs and maintain the ride-hailing neighbor events in Redis. For each passenger, we store his latest 30 Preview, Request, Cancel_Order and Finish_Order events, respectively. Our online system subscribes a series of Kafka topics about user events. Once an event message happened, the user's dynamic heterogeneous graph is updated immediately. Based on the dynamic graph, our embedding model updates the user's embedding subsequently. To maintain different graphs for passengers, we utilize 1TB storage for online serving.

4.2.1 PreView Event Embedding Online Results. Our framework is already deployed online and significantly improves online prediction results. Specifically, for the PreView event conversion prediction, our online forecasting framework is a Gradient Boosting Decision Tree base model [24], and utilizes our REGNN embedding results as model input features. The online improvements are very significant: First, REGNN features obtain very high ranking among all the features (**4 out of top 10 most important features**). Second, online AB tests show that all the generated embeddings of REGNN bring a **3.5%** improvement for the online AUC. As our event embedding is generated in an inductive way, the new event embeddings are generated based on its dynamic subgraphs, and there is no need to worry about the problem of vector space transformation, which is often faced by traditional transductive embedding methods. As a result, our real-time PreView behavior embedding service is already deployed in production and serves for down-stream tasks such as conversion prediction and scheduling strategies.

4.3 Model parameter evaluation and analysis

In this section, we analyze the impact of different model parameters as follows.


(a) Number of Neighbors


(b) Sample Rate

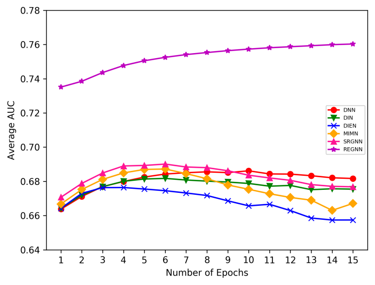

(c) Epoch Number

Figure 6: Influence of Hyper Parameters

4.3.1 *Number Of Neighbors.* Figure 6a illustrates the results of different numbers of neighbors of our framework compared to baselines. We vary the number of neighbors from 10 to 80, our method obtains significant and stable improvements over other baselines, which testifies the robustness of our model. For online serving, we select 30 neighbors for each PreView for performance and storage trade-off.

4.3.2 *Size Of Training Data.* Moreover, we vary the percentage of training data in the PreView Dataset. The results are shown in Figure 6b. We randomly select 30%, 70% and 100% of training data and evaluate the model performance on all the test data. All experiments show improvements of the REGNN model over others. It has also proved the robustness of the REGNN for embedding events.

4.3.3 *Number Of Training Epochs.* In addition, we vary the number of training epochs to compare different methods. The results are shown in Figure 6c. We restrict the epoch number from 1 to 15 for all models for fair comparison. The number of offline training epochs for REGNN is set as 100, and the performance of REGNN continues to grow and the best performance for REGNN is obtained at epoch number 55.

4.3.4 *Modeling Units For Time Series.* Here we compare the effectiveness of different modeling units in time series of events embedding as Table 5, where M-GRU, Relu-GRU and M-Relu-GRU are variants of GRUs proposed in [20]. Specifically, M-GRU deletes the reset gate in traditional GRU, Relu-GRU changes the *tanh* activation function to Relu, and M-Relu-GRU combines the modification in M-GRU and Relu-GRU. LSTM [11] is another kind of RNN for sequential modeling. As shown in the table, among all the different kinds of sequential modeling units, GRU fits REGNN the best in our framework. Moreover, we also investigate modeling the time-series of events as graphs, and aggregate the event embeddings through different graph neural networks such as Pooling, GAT and GCN. The comparison results are given in Table 5. The experimental results shows RNN outperforms other modeling units. It is in line with our expectation that RNNs (Sequential Modeling units) can further capture the order and dependencies between events.

Table 5: Results Of Different Modeling Units On PreView Dataset

Units	AUC	Units	AUC
GRU	**0.7684**	LSTM	0.7623
M-GRU	0.7657	LSTM+Attention	0.7585
Relu-GRU	0.7630	GAT	0.7584
GRU+Attention	0.7629	GCN	0.7566
M-Relu-GRU	0.7616	Pooling	0.7593

4.4 Reproducibility Details

For offline training, we employ Keras [4] to implement REGNN and further conduct it on a server with GPU machines. The drop-out ratio is set to 0.2 before each layer in our implementation, and batch normalization and layer normalization are performed before each layer in the model. The input graph for each Preview event is formatted as TFRecords, where the passenger subgraph, the origin subgraph and the destination subgraph are stored as adjacent matrix of events, and each event is described as a vector of node features. We describe the training algorithm in detail in Section A. When serving online, our system utilizes a Flink [1] system to: I) first receive all the passenger events, and II) extract the node features from events, III) connect the events to its neighbors, then IV) update the heterogeneous dynamic passenger graphs, origin graphs and destination graphs stored in a Redis server, and V) finally generate real-time event embedding. The expiration time of keys in Redis is set to 7 days. When deployed online for AB testing on several cities, the QPS of our online system is 200, and the real time embedding latency is less than 20ms at 99%.

5 RELATED WORK

Session-Based Recommendation SRGNN [30] created a directed graph for the connections of all sessions according to chronological relationships, and used GCN to learn item embedding for recommendations. Moreover, DGRec (Dynamic Graph Recommendation) [27] and RNN-Session [2] also conduct session based recommendations, however [2] could not deal with new items, and [27] focus on homogeneous social graphs.

Graph Neural Network Method GNNs [22] have gained a lot of attention in recent years. GCN (graph convolutional network [15]) uses operations on full graph Laplacian, which is designed in a transductive setting for semi-supervised learning. GraphSAGE [10] extents the GCN framework to the inductive setting and uses different neural networks like LSTM to aggregate neighbors' information. GAT (Graph Attention Networks [26]) applies self-attention mechanism to measure and combine different neighbors impacts. For heterogeneous graph, HetGNN (Heterogeneous Graph Neural Network [31]) proposes using type-based neighbors aggregator and an attention mechanism for heterogeneous types combination.

Deep CTR Method Most deep models pay attention to the interaction between features and follow the framework of Embedding and Multi-Layer Perceptron (MLP). Wide&Deep [3] combines features from wide linear representation and deep neural networks. DeepFM [9] imposes a factorization machine as the wide module in Wide&Deep to enhance the power of expression. Users' historical behavior data reveals users' dynamic interest and has been proven effective for CTR prediction. DIN (Deep Interest Network [33]) proposes attention mechanism to capture users' interest. DIEN (Deep Interest Evolution Network [32]) uses GRU refined with attention mechanism and an auxiliary loss to supervise the learning of interest evolution. MIMN (Multi-channel user Interest Memory Network [19]) proposes a method based on memory network to capture users' long sequential behavior data.

6 CONCLUSIONS

In this paper, we propose a novel framework for real-time event embedding by constructing dynamic heterogeneous graphs. Our framework learns the embedding of new events in an inductive way, which aggregates neighbor information from heterogeneous graph consisting of different types of events with comprehensive relationships. Through carefully designed GATs at different levels, our framework captures the semantic and structural information between events. In addition, after exhaustive experiments, we found that RNNs are more suitable for capturing order and dependencies between events. Our proposed framework is general and can be applied to model different types of events (we provide another application of the proposed framework in the supplement). After a lot of AB experiments and analysis, our model has been deployed in a production environment and serves tens of millions of customers every day.

REFERENCES

[1] Apache. 2003. Apache Flink: Stateful Computations over Data Streams. https://flink.apache.org/.

[2] Hidasi B, Karatzoglou A, and et al. Baltrunas L. 2015. Session-based Recommendation with Recurrent Neural Networks. *arXiv preprint arXiv:1511.069397* (2015).

[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 7–10.

[4] François Chollet et al. 2018. Keras: The python deep learning library. *Astrophysics Source Code Library* (2018).

[5] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 135–144.

[6] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401* (2014).

[7] Mihajlo Grbovic. 2017. Search Ranking And Personalization at Airbnb. In *the Eleventh ACM Conference*.

[8] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[9] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* (2017).

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[12] Chung J, Gulcehre C, Cho K, and et al. 2015. Gated feedback recurrent neural networks. In *Proceedings of International Conference on Machine Learning*. 2067–2075.

[13] Zhou J, Cui G, and Zhang Z. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018). arXiv:1812.08434 http://arxiv.org/abs/1812.08434

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[16] Yucheng Lin, Xiaoqing Yang, Zang Li, and Jieping Ye. 2019. AHINE: Adaptive Heterogeneous Information Network Embedding. *arXiv preprint arXiv:1909.01087* (2019).

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[18] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[19] Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Practice on Long Sequential User Behavior Modeling for Click-Through Rate Prediction. *arXiv preprint arXiv:1905.09248* (2019).

[20] Mirco Ravanelli, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio. 2017. Improving speech recognition by revising gated recurrent units. *arXiv preprint arXiv:1710.00641* (2017).

[21] Salvatore Sanfilippo. 2019. Redis: an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. https://redis.io/.

[22] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.

[23] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and S Yu Philip. 2018. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 31, 2 (2018), 357–370.

[24] Chen T and Guestrin C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 785–794.

[25] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[27] Song W, Xiao Z, and et al Wang Y. 2019. Session-based social recommendation via dynamic graph attention networks. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 555–563.

[28] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 839–848.

[29] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous Graph Attention Network. In *The World Wide Web Conference*. ACM, 2022–2032.

[30] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 346–353.

[31] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. 2019. Heterogeneous Graph Neural Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 793–803.

[32] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 5941–5948.

[33] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1059–1068.

A SUPPLEMENT

A.1 Pseudo Code Of REGNN Training Procedure

The pseudo code of REGNN training procedure is described in Algorithm 1. Dynamic heterogeneous graphs for events are constructed before training. Specifically, for PreView events, three different heterogeneous graphs are constructed.

Algorithm 1 Training Procedure of REGNN

```
1: Input: dynamic heterogeneous graphs for events, event labels
   $y$, iteration times $iter$, training batch size $b$, number of training
   data $num$
2: Output: model parameters $\theta$ of REGNN
3: set $i = 0$
4: repeat
5:     set $j = 0$
6:     repeat
7:         Sample $b$ dynamic heterogeneous events with labels and
           perform batch normalization.
8:         Generate event embeddings for each event according to
           Equation (4,5,6,7).
9:         Compute batch loss according to Equation (10).
10:        Compute gradient and update model parameters $\theta$.
11:        set $j = j + b$
12:    until $j >= num$
13:    set $i = i + 1$
14: until $i >= iter$
15: return model parameters $\theta$
```

A.2 Model Generalization To Dialogue Event Prediction

We introduce another application at DiDi that uses the proposed real-time event embedding framework , which is the customer question prediction in the context of the chatbot customer service as Figure 7. The figure illustrates the question prediction of a chatbot, which predicts the user's most likely clicks or questions based on the user's previous behavior, not only the user's interactions with the chatbot, but also the previous transaction behavior (e.g., placing an order). For example, a passenger has just finished a ride-hailing order, she may want to consult about "cost" or "lost_items" related questions. The user will also process a series of interactions with the chatbot, including inputting text, clicking on the questions or answers pushed by the chatbot. We also divide these interactions into several h-sessions which are separated by the event of "Click_Question" because each click means the user's intent is satisfied partially. In order to embedding the user's current behavior, we also construct the user's dynamic heterogeneous event graph based on the user's previous N-day historical behaviors.

Dynamic Dialogue Graph Generation Specifically, the architecture for dialogue event prediction is show in Figure 8. Moreover, we give the dialogue event notations as Table 6. Similar to the PreView event graph, for a Click_Question event CQ_T at time T, the dynamic dialogue graph of CQ_T is built as described in Figure 7.

Table 6: Dialogue Event Notations

Symbols	Definitions and Descriptions
O_T	Order event at time T
CQ_T	Click_Question event at time T
I_T	Input_Text event at time T
$DiaG_T$	heterogeneous Dialogue Graph at time T

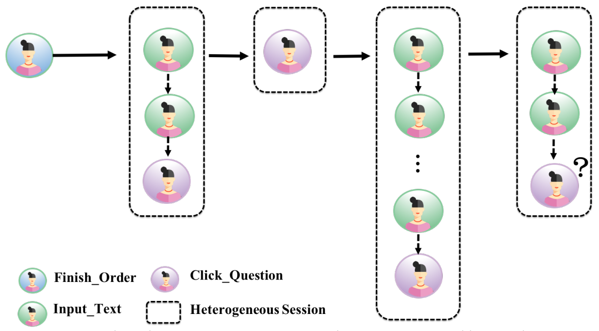


Figure 7: Chatbot Interaction Flow. Typically, when a customer finished a disputed order, she may interact with the chatbot through different actions such as Input_Text and Click_Question.

Considering that all text inputting in an h-session

$$DiaG_T = HetG_{c,T} \tag{11}$$

where $DiaG_T$ represents the dialogue heterogeneous event graph at time T, consisting of one heterogeneous subgraph about the customer c. Note that for the dialogue response prediction task, we only build one heterogeneous subgraph.

Dialogue Event Graph Embedding Similarly, as depicted in Figure 8, given the heterogeneous graph $DiaG_T$ of dialogue at time T, an h-session in $DiaG_T$ consists of the following events, Order O_T, Click_Question event CQ_T and all the Input_Text events occur between timestamp $T-1$ and T, denoted as SI_T. The GATs inside h-session is computed as:

$$h^{(1)}_{CQ_T} = GAT_I(h^{(0)}_{O_T}, h^{(0)}_{CQ_T}, h^{(0)}_{I_i}, \forall I_i \in SI_T) \tag{12}$$

where $h^{(1)}_{CQ_T}$ is the embedding after GATs inside h-sessions. Afterwards, the cross h-session GAT is:

$$h^{(2)}_{CQ_T} = GAT_c(h^{(1)}_{CQ_t}), \forall t, T-M <= t <= T \tag{13}$$

where GAT_c denotes cross h-session GATs operations for updated Click_Order events, and the attention mechanism operator works in the same way as that in the PreView event embedding.

Moreover, the GATs within h-session and cross h-session for dialogue graphs are computed in the same way as those in the PreView event embedding. Since we only build one heterogeneous event graph for dialogue event, thus there is no global GATs across subgraphs, E_T is the final embedding of event CQ_T.

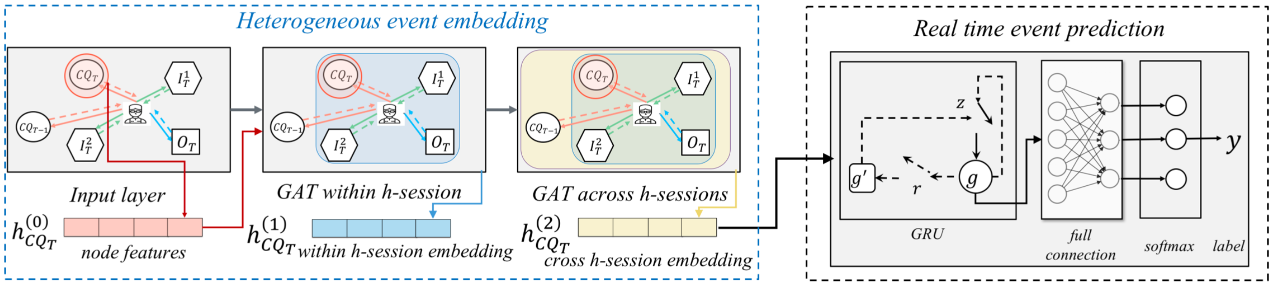

Heterogeneous event embedding
Real time event prediction
Input layer
GAT within h-session
GAT across h-sessions
$h_{CQ_T}^{(0)}$ node features
$h_{CQ_T}^{(1)}$ within h-session embedding
$h_{CQ_T}^{(2)}$ cross h-session embedding
GRU
full connection
softmax
label
y

Figure 8: Heterogeneous Graph Embedding of Dialogue Event. One heterogeneous interaction graph is built for Click_Question event embedding, and GATs at different levels are utilized.

Specifically, for dialogue response prediction, the objective is as follows:

$$L = -\sum_{u \in U}\sum_{t=1}^{T}\sum_{k=1}^{N_C} i_{u,t}^{k}\ logP_{u,t}(y=k) + \lambda||\theta||_2$$
$$P_{u,t}(y=k) = \frac{exp(\theta_k \cdot h_T^u)}{\sum_{j=1}^{N_C} exp(\theta_j \cdot h_T^u)} \tag{14}$$

where $i_{u,t}^{k}$ is an indicator function, if the target class of user u and time t is k, then $i_{u,t}^{k}$ is set to 1, else is set to 0. N_C is the total number of classes in a multi-classification problem. $P_{u,t}(y=k)$ computes the probability of u at time t for label k in a softmax manner, θ represents the parameters to learn, and $\lambda||\theta||_2$ is the $L2$ regularization of parameters. Obviously, we can change the loss function to RMSE error to learn regression problems for other kind of prediction tasks.

A.3 Experimental Results On Dialogue

Dataset For training dialogue event embedding, we adopt customer dialogue log of 1 week for dialogue heterogeneous graph construction, and adopt no more than 10 recent events. We randomly select 177325 samples as training set and 34126 samples as test set. The feature size for each sample is 201 and the number of classes is 403.

Content Features For an event(actually a heterogeneous graph) in the dialogue flow, it contains basic features of order, Click_Question embedding, Input_Text embedding, and neighbors from it's latest events. We apply AHINE(Adaptive Heterogeneous Information Network Embedding [16] to get Click_Question embeddings and it is a network embedding method for heterogeneous graph which models the edges by deep learning. And we use word2vec [17] for Input_Text embddings.

Experiment Setup The number of neighbors in REGNN for for dialogue dataset is 10 (including last 10 Click_Question events and their correponding Input_Text events). And the number of our pre-trained embedding dimension is 30. Similar to the setup in PreView, we use Adam [14] with learning rate at 0.001 for all models. The number of hidden dimension of GRU in our method is set to 100. The DNN Layer is set to 128 × 64 × 32 before the output layer.

Baseline Settings For PreView the input is sequences of node features of PreViews. For dialogue, the input is the sequence of pre-trained Click_Question and Input_Text embeddings. We use the same input features and embeddings for all baselines. For SRGNN, the origin output layer's dimension is determined by size of items, in our framework it's changed to fully connected layer with dimension of classification numbers, followed by a softmax layer.

Software & Hardware We employ Keras[4] to implement REGNN and further conduct it on a server with GPU machines.

Offline Performance The offline results are given at Table 7, from which we can see REGNN significantly outperforms other baseline methods.

Table 7: Performance Of Different Methods On Dialogue Dataset

Method	Accuracy(mean ± std)
DNN	0.2749 ± 0.0056
DIN	0.2765± 0.0071
DIEN	0.2833± 0.0027
MIMN	0.2855± 0.0035
SRGNN	0.3249± 0.0209
REGNN	**0.3783** ± **0.0023**

Online Performance Note that our framework is also deployed online for customer service. Specifically, our online prediction model for customer dialogue response prediction is a DNN based model and the REGNN embedding works as input features of the model. The online effectiveness of REGNN is obtained through online AB test. The AB test results show that: the click through rate of the questions increases 8.3%, and the rate of problems solved improves 1.7%. Consequently, the rate of customer turning to manual services drops 1.4% (the lower the better), which indicates better service of our intelligent chatbot service. Since the number of customers turning to manual help is reduced, the labor cost of customer service is also greatly reduced. As a matter of fact, our dialogue event embedding service helps the prediction of user question selection and customer tagging service, etc.

Discussion Similar to the online PreView embedding architecture, the online architecture of Dialogue event embedding also consists of two parts: I) Dynamic Graph Construction and II) real-time embedding. Once a customer clicked a question, the event was immediately added to the customer graph and connected to its previous heterogeneous neighbors. Afterwards, real-time embedding service updates the event embedding. The whole online service is event driven, making it latency free for real-time down-stream prediction. Future work includes adding more events for embedding (such as customer comments, customer complaints, etc.), and subgraphs (e.g., complained user graph).