Real-world ML use cases

Piero Molino

Agenda

- NLP models for Customer Support
- Model retraining strategies
- Gran Neural Networks for dish/restaurant recommendation
- Learning from recommender system deployment
- Lessons learned from real-world data collection

UBER

Customer Obsession Ticket Assistant

Improving Uber Customer Support with Natural Language Processing and Deep Learning

Piero Molino | AI Labs
Huaixiu Zheng | Applied Machine Learning
Yi-Chia Wang | Applied Machine Learning

Main Takeaways

COTA v1: classical NLP + ML models

- Faster and more accurate customer care experience
- Million $ of saving while retaining customer satisfaction

COTA v2: deep learning models

- Experiments with various deep learning architectures
- 20-30% performance boost compared to classical models

COTA Blog Post and followup, KDD paper

Secure | https://eng.uber.com/cota/

Uber Engineering Updates: email address SUBSCRIBE

UBER Engineering

Search Articles Facebook Twitter Join the Team Uber Open Source

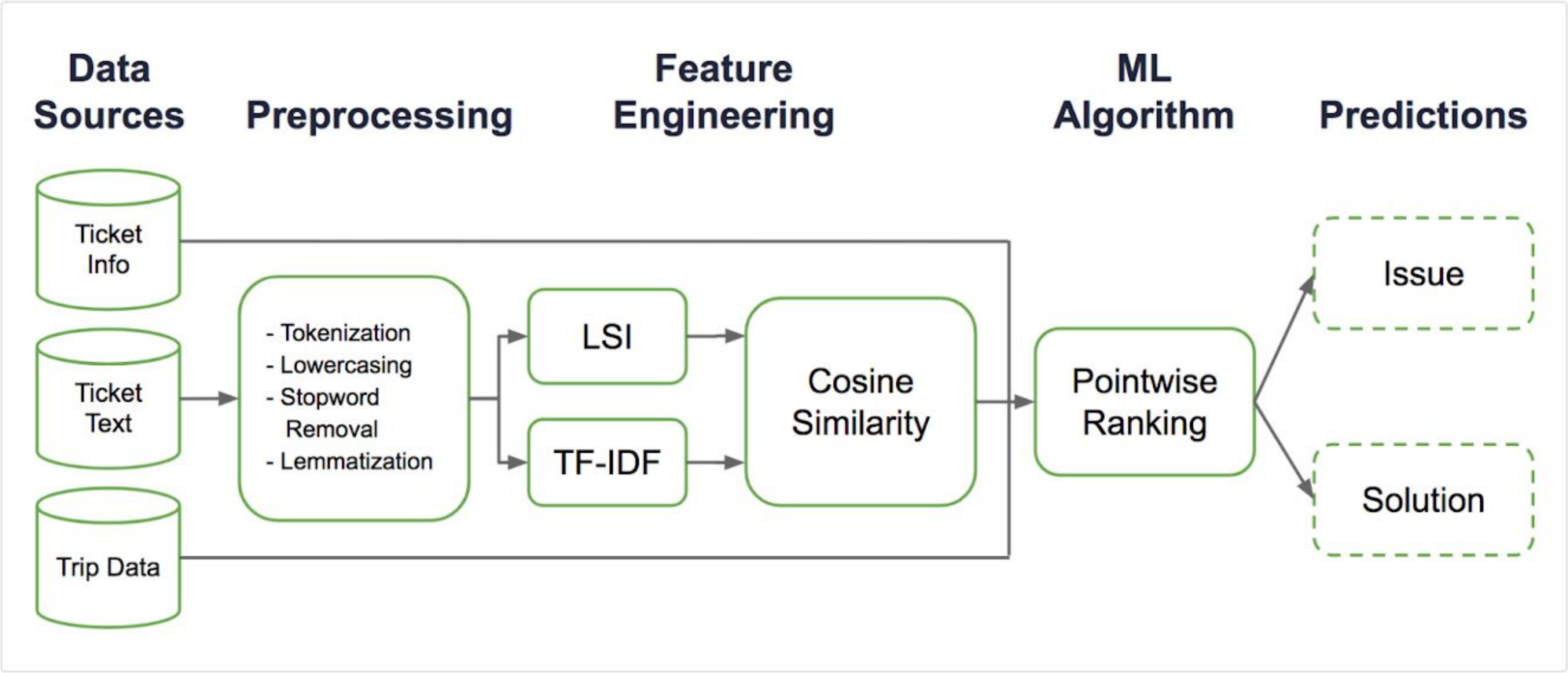
CATEGORIES

- Architecture
- AI
- Uber Data
- Open Source
- Mobile
- General Engineering
- Team Profile
- Culture

COTA: Improving Uber Customer Care with NLP & Machine Learning

By Huaixiu Zheng, Yi-Chia Wang, & Piero Molino

January 3, 2018



Agenda

- Motivation and Solution
  - Complexity of Customer support @Uber
- COTA v1: Traditional ML / NLP Models
  - Multi-class Classification vs Ranking
- COTA v2: Deep Learning Models
  - Deep learning architectures
- COTA v1 vs COTA v2

Agenda

Motivation and Solution

- **Complexity of Customer support @Uber**

COTA v1: Traditional ML / NLP Models

- Multi-class Classification vs Ranking

COTA v2: Deep Learning Models

- Deep learning architectures

COTA v1 vs COTA v2

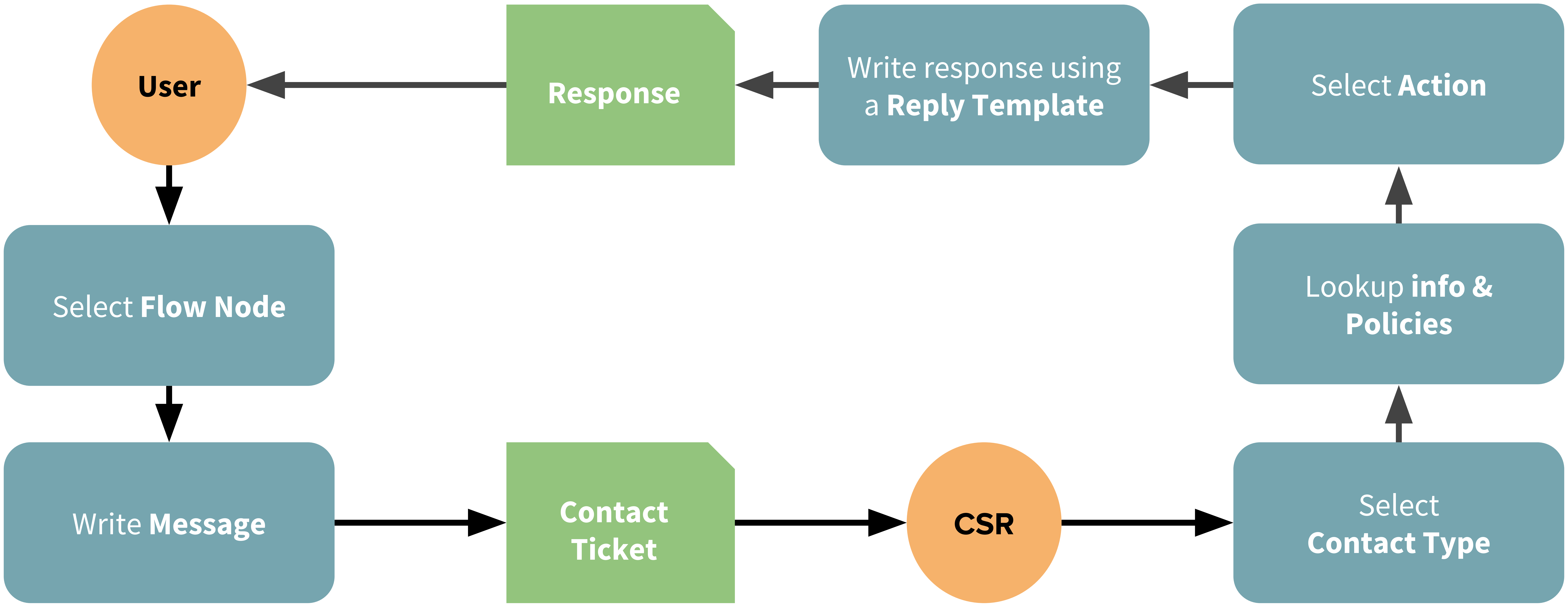
What is the challenge?

As Uber grows, so does our volume of support tickets

Millions of tickets from riders / drivers / eaters **per week**

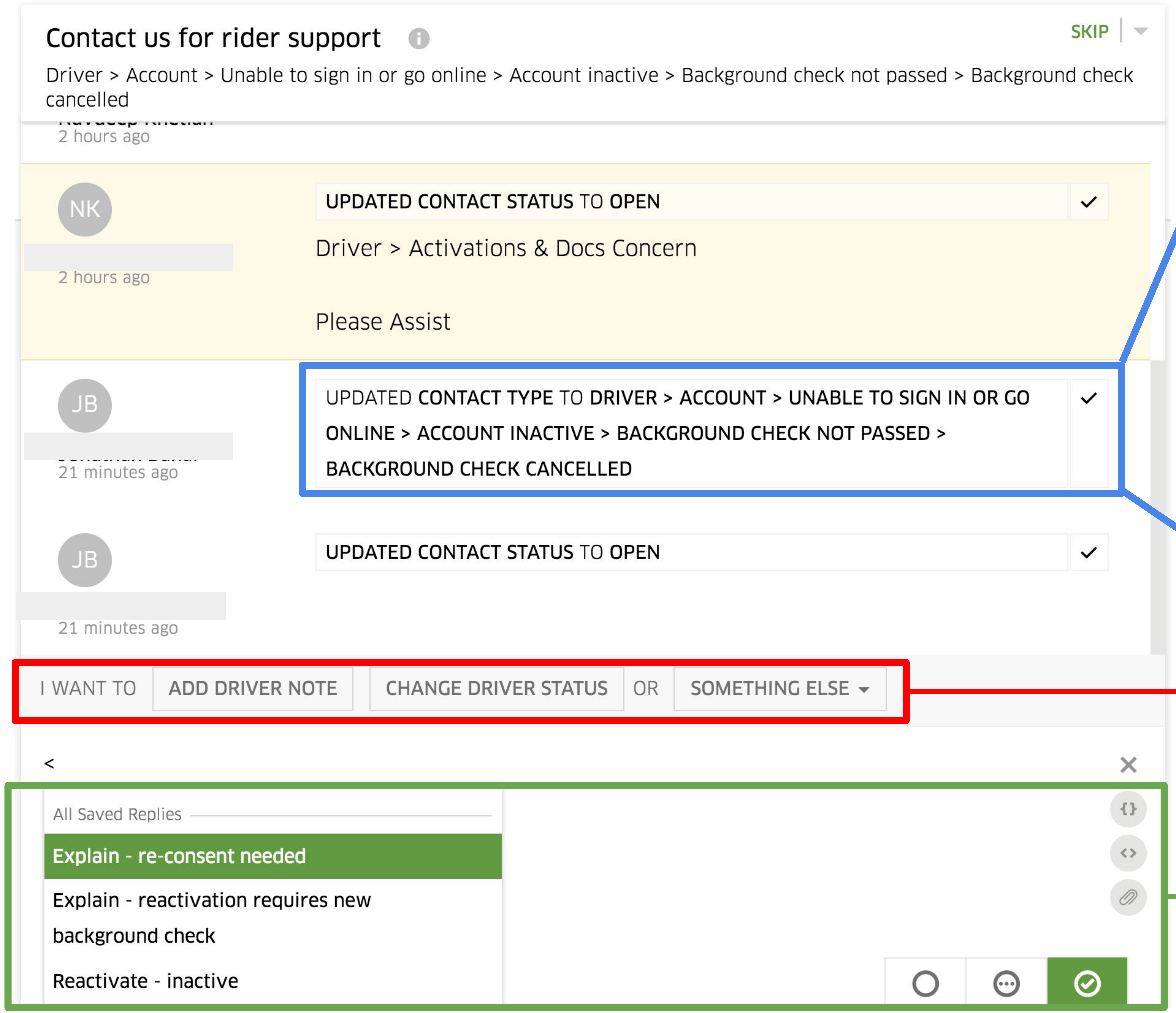
Thousands of different types of issues users may encounter

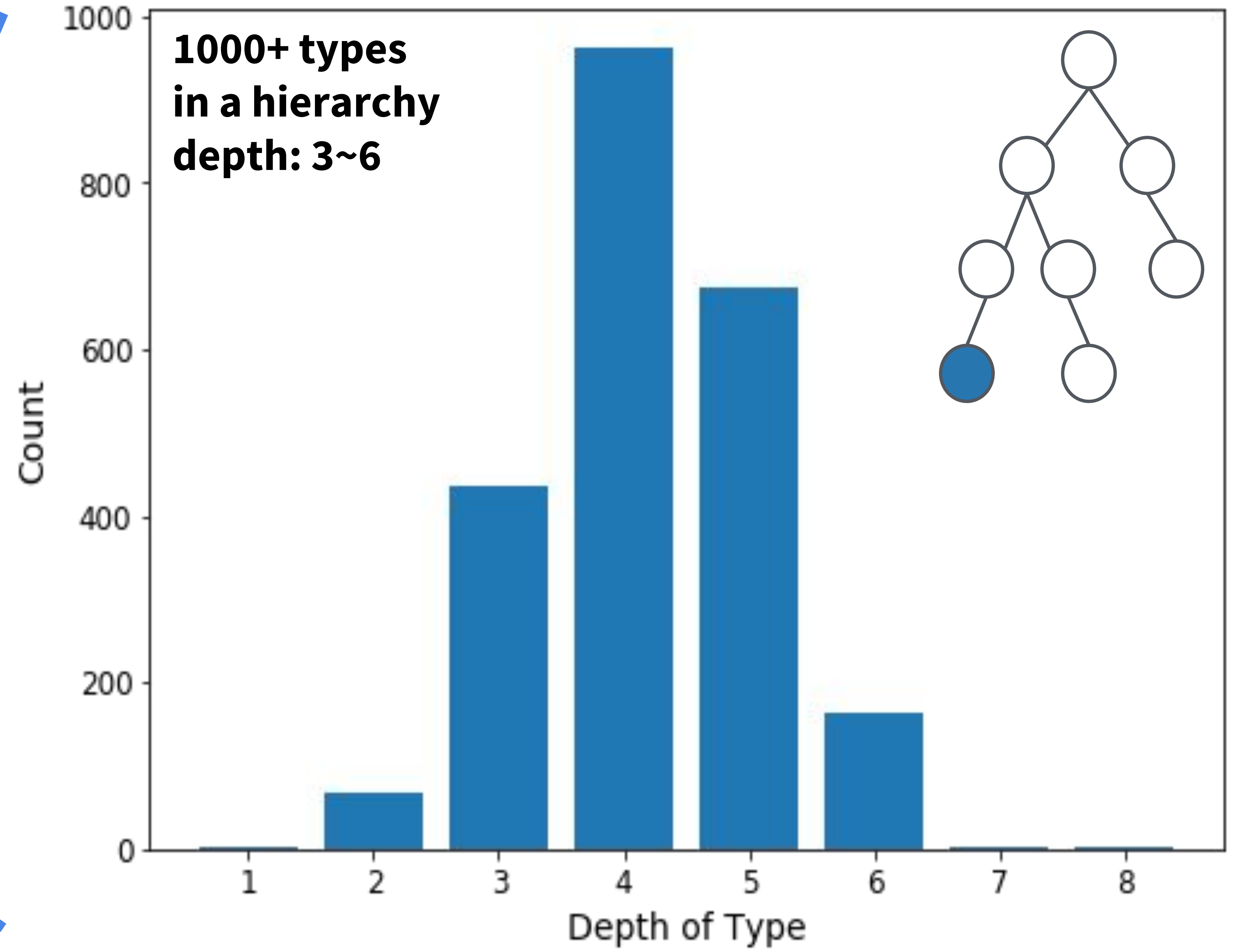
Uber Support Platform



What is the challenge?

And it is not easy to solve a ticket





10+ actions (adjust fare, add appeasement, …)

1000+ reply templates

Agenda

Motivation and Solution

- Complexity of Customer support @Uber

COTA v1: Traditional ML / NLP Models

- **Multi-class Classification vs Ranking**

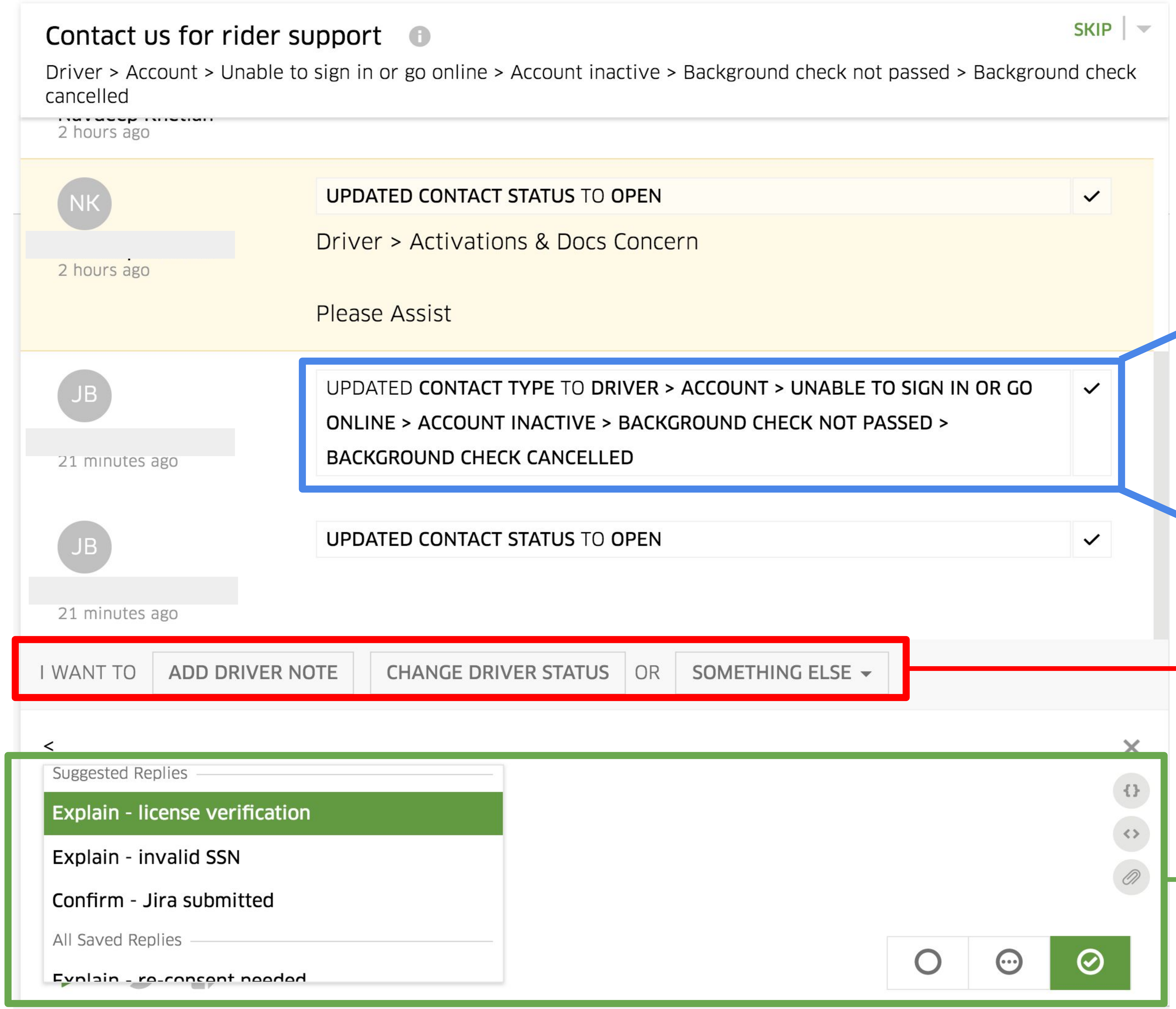
COTA v2: Deep Learning Models

- Deep learning architectures

COTA v1 vs COTA v2

COTA v1: Suggested Resolution

Machine learning models recommending the 3 most relevant solutions



SUGGESTED CONTACT TYPES

Driver > Account > Unable to sign in or go online > Account inactive

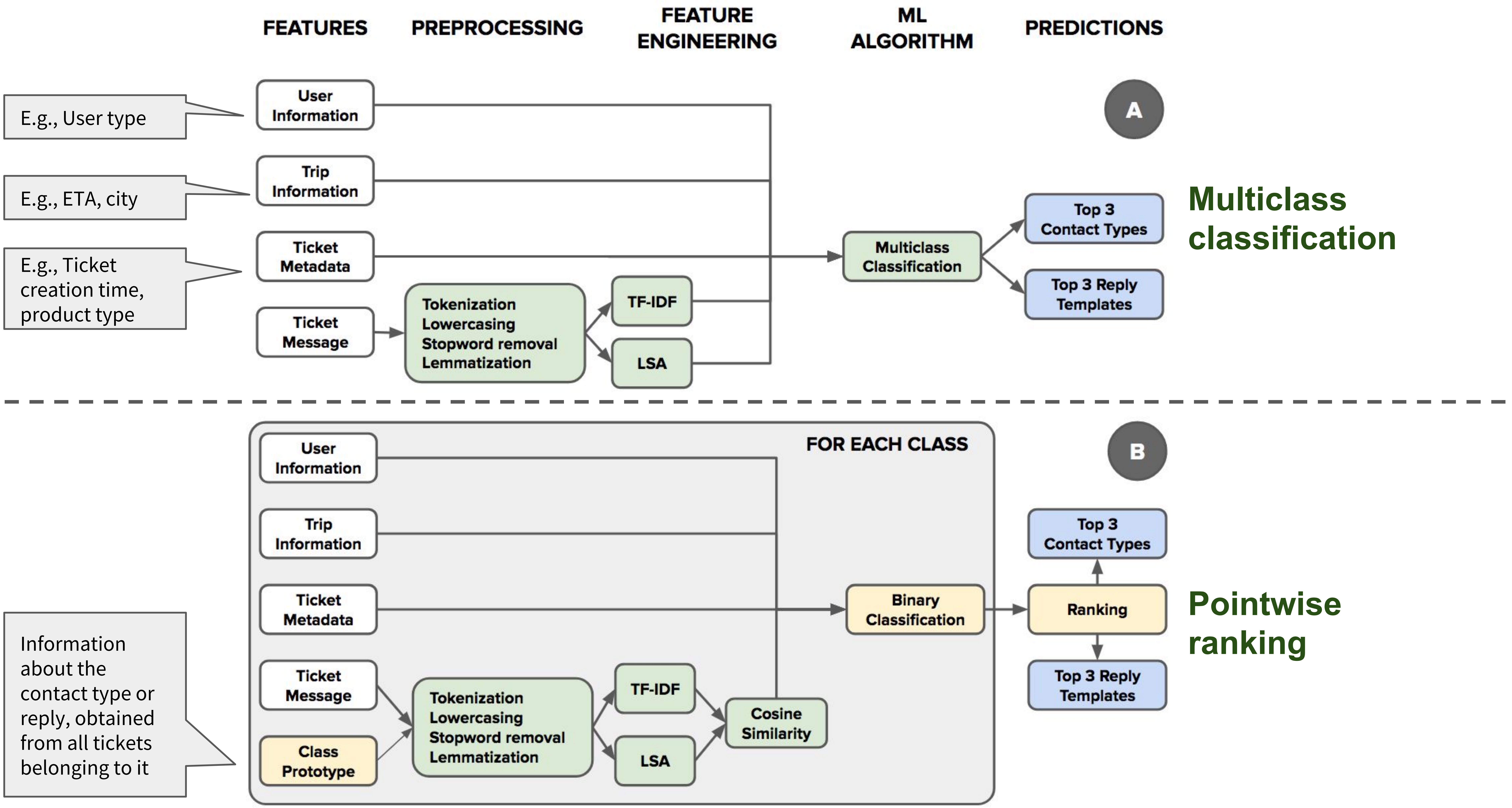
Driver > Account > Profile > Unsubscribe > SMS or Text

Driver > Account > Vehicles > Edit vehicle class

Reorder actions in relevance

Surface top-3 most-relevant reply templates

COTA v1 Model Pipeline



From Classification to Ranking

Multi-class Classification

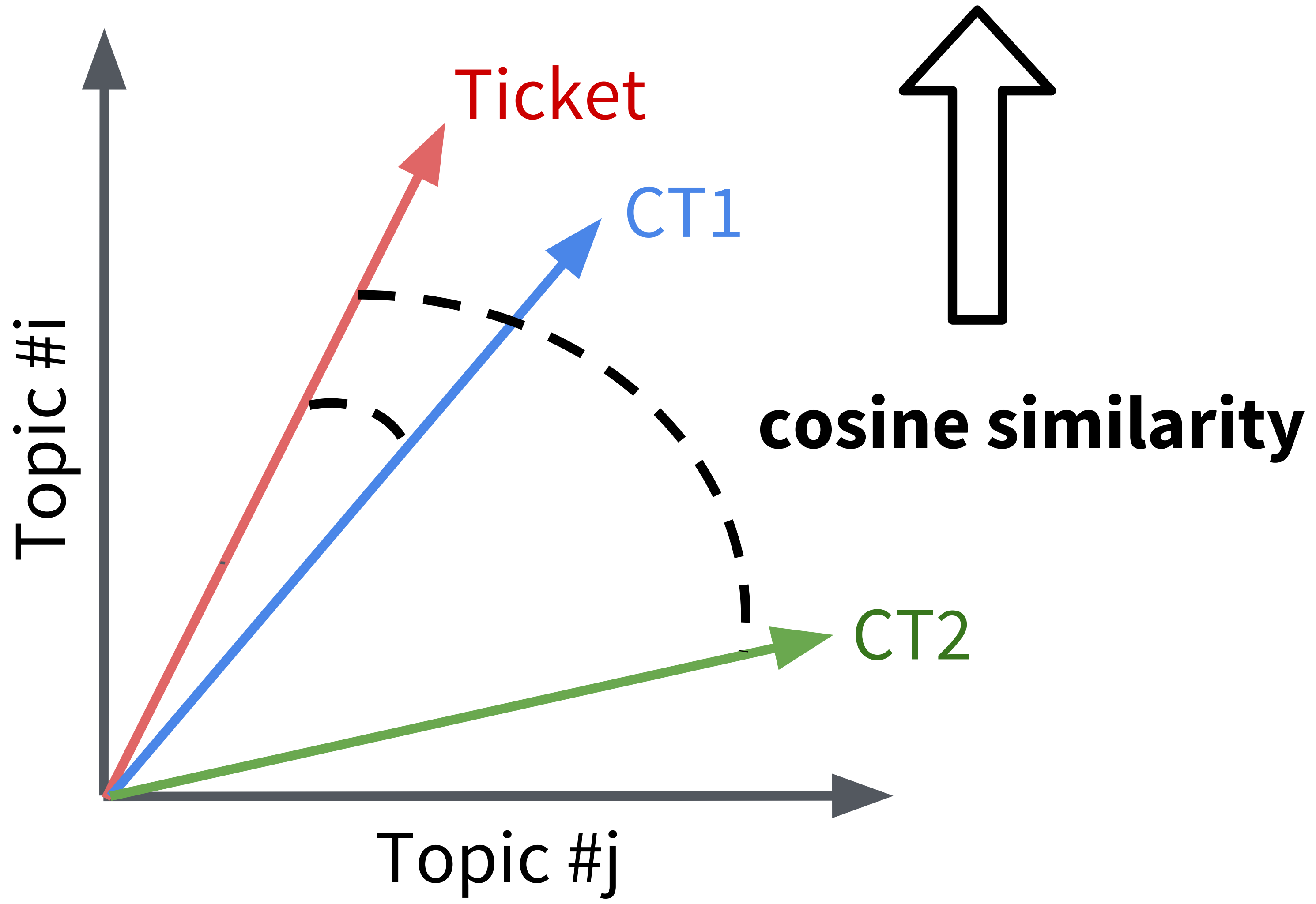
Tickets Features	Label (CT1, CT2)
t1 features	CT1
t2 features	CT2

Pointwise Ranking

Tickets Features	Type Features	Sim (t, CT)	Label (0, 1)
t1 features	CT1 features	0.8	1
t1 features	CT2 features	0.1	0
t2 features	CT1 features	0.2	0
t2 features	CT2 features	0.7	1

Ranking allows us to include **features of candidate types** and **similarity features** between a ticket and a candidate type

Model: **Random Forest** with hyperparameters optimized through **grid search**



Performance Comparison

6% absolute (10% relative) improvement

Hits@3: any of the top 3 suggestions is selected by CSRs

Agenda

Motivation and Solution

- Complexity of Customer support @Uber

COTA v1: Traditional ML / NLP Models

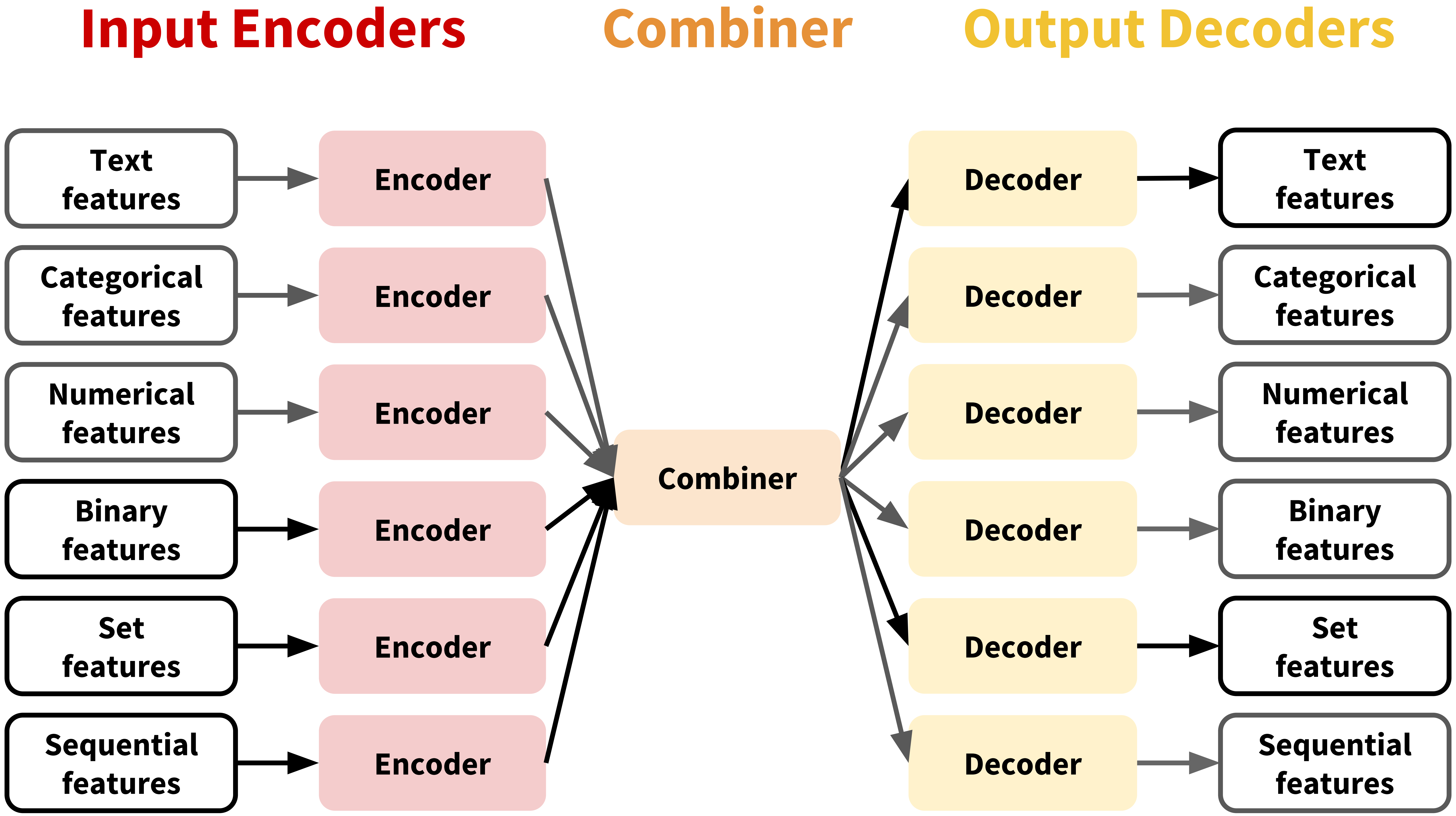
- Multi-class Classification vs Ranking

COTA v2: Deep Learning Models

- **Deep learning architectures**

COTA v1 vs COTA v2

COTA v2: Deep Learning Architecture



Generic architecture, **reusable** in many different applications.
We are considering open-sourcing it!

Documentation
http://ludwig.ai

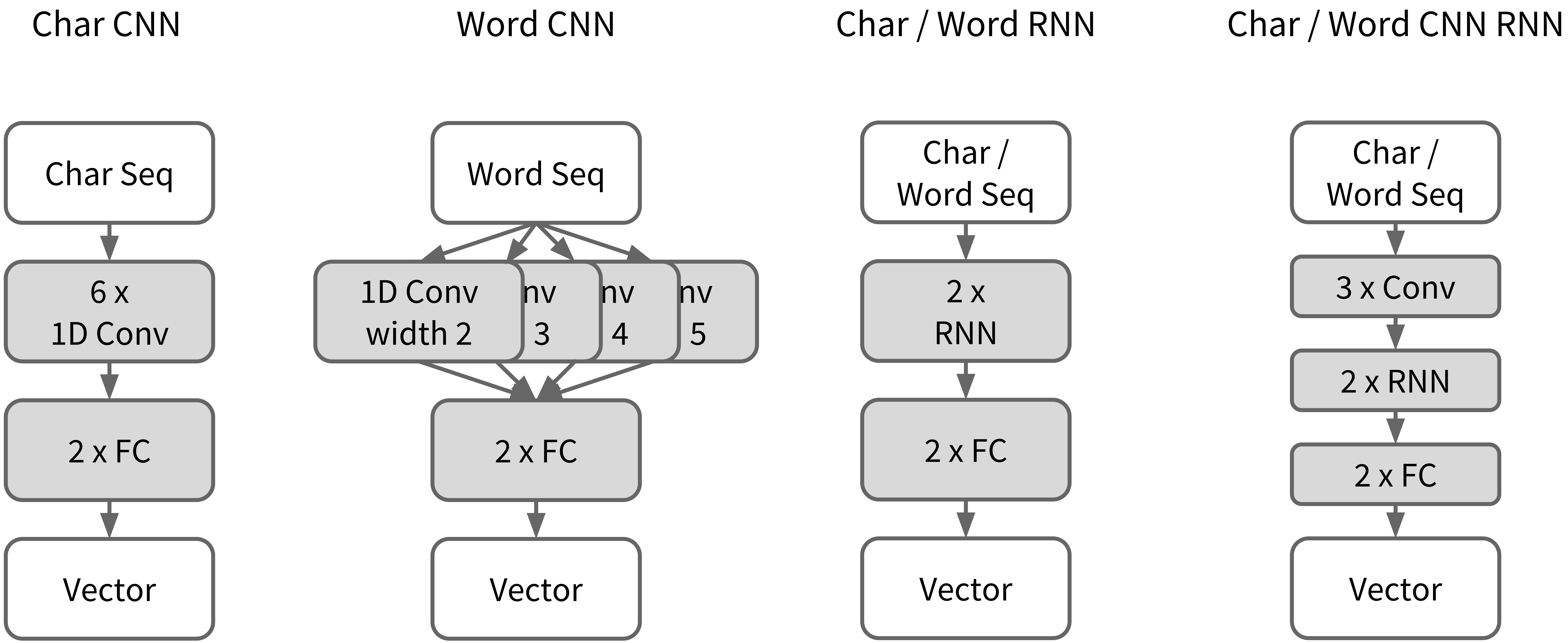
Repository
http://github.com/uber/ludwig

Blogpost
http://eng.uber.com/introducing-ludwig
http://eng.uber.com/ludwig-v0-2/
http://eng.uber.com/ludwig-v0-3/

White paper
https://arxiv.org/abs/1909.07930

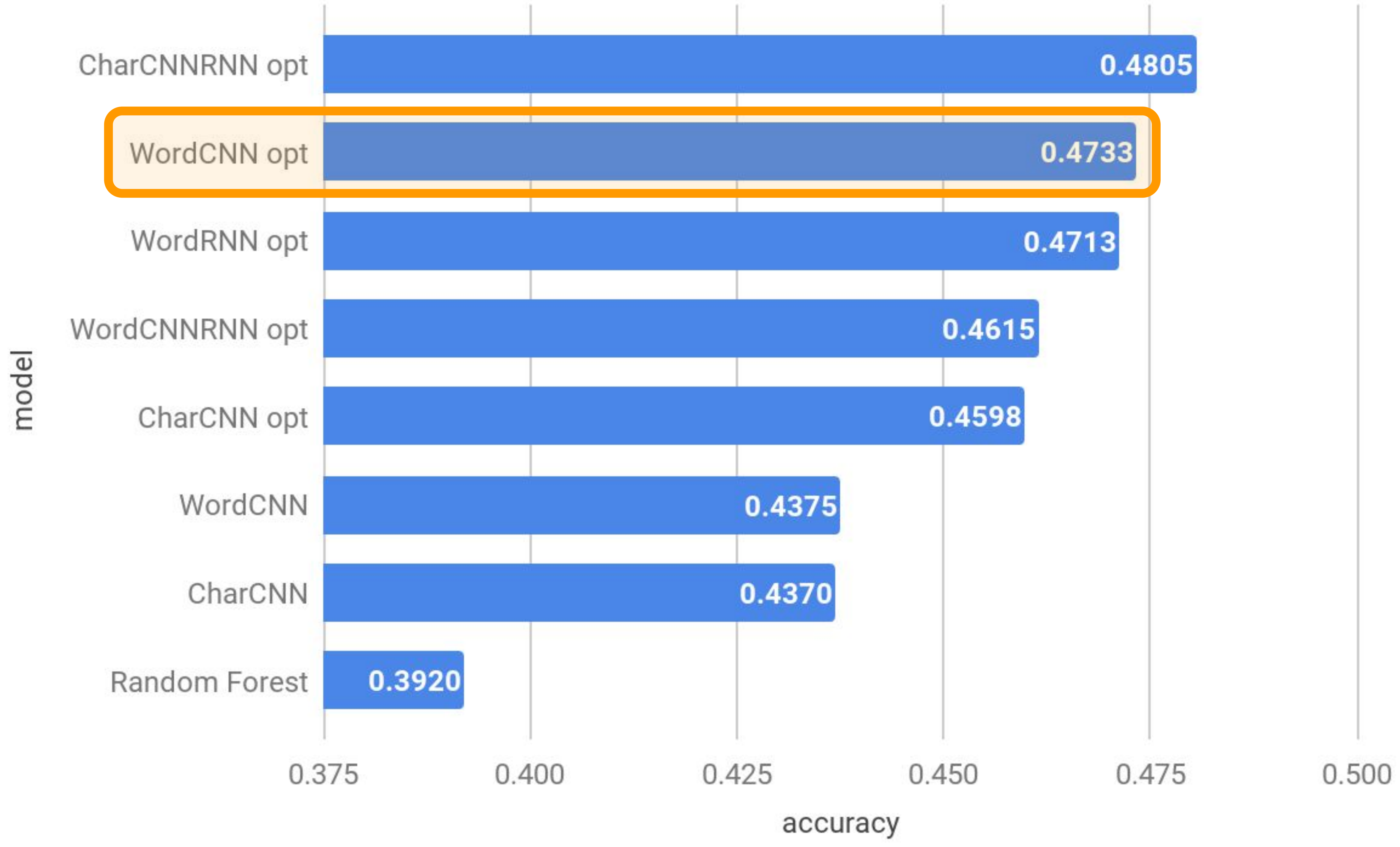
Key contributors
Travis Addair
Yaroslav Dudin
Sai Sumanth Miryala
Jim Thompson
Avanika Narayan
Ivaylo Stefanov
John Wahba
Doug Blank
Patrick von Platen
Carlo Grisetti
Chris Van Pelt
Boris Dayma

COTA v2: Text Encoding Models



Which text encoder?

Hyperparameter search for contact type classification



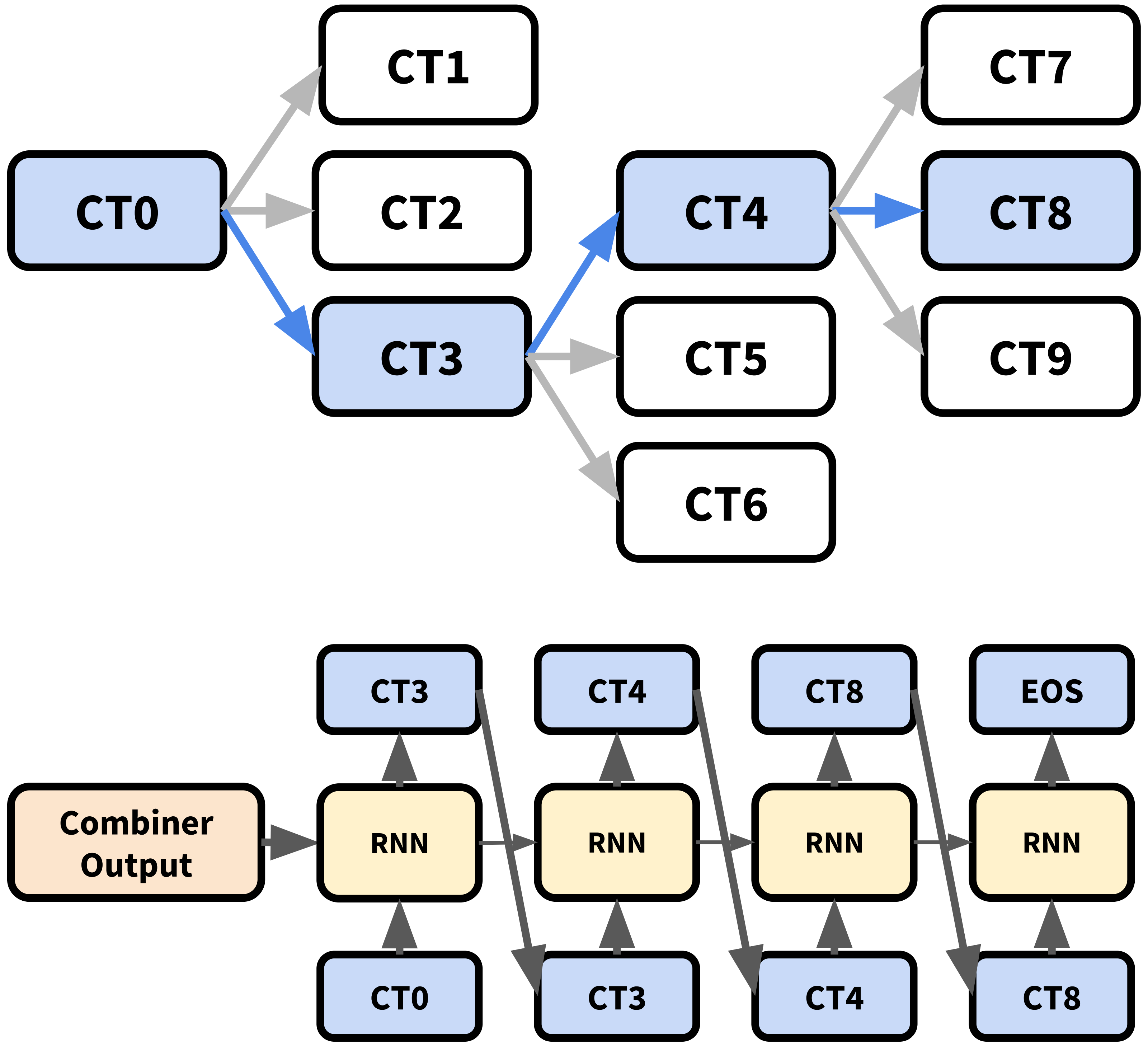
Model	Validation accuracy	Training time per epoch in minutes
CharCNNRNN opt	0.4805	35
WordCNN opt	**0.4733**	**4**
WordRNN opt	0.4713	17
WordCNNRNN opt	0.4615	12
CharCNN opt	0.4598	5

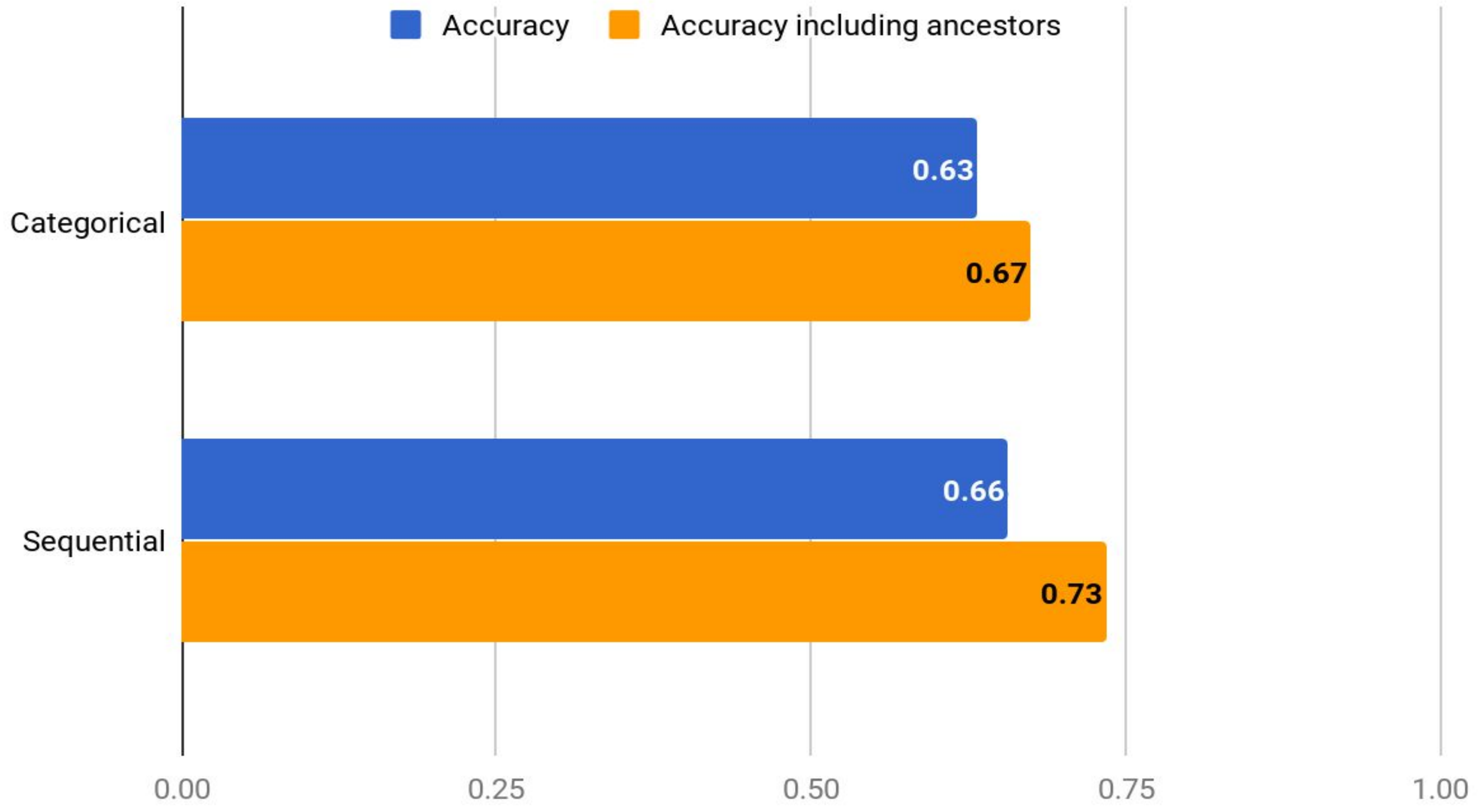
WordCNN is the **best compromise** between **performance** and **speed**

20%+ over Random Forest used in COTA v1 and **~10x** faster than CharCNNRNN

Sequence Model for Type Selection

Predict the sequence of nodes instead of leaf node





Example: **Driver** > **Trips** > **Pickup and drop-off issues** > **Cancellation Fee** > **Driver Cancelled**

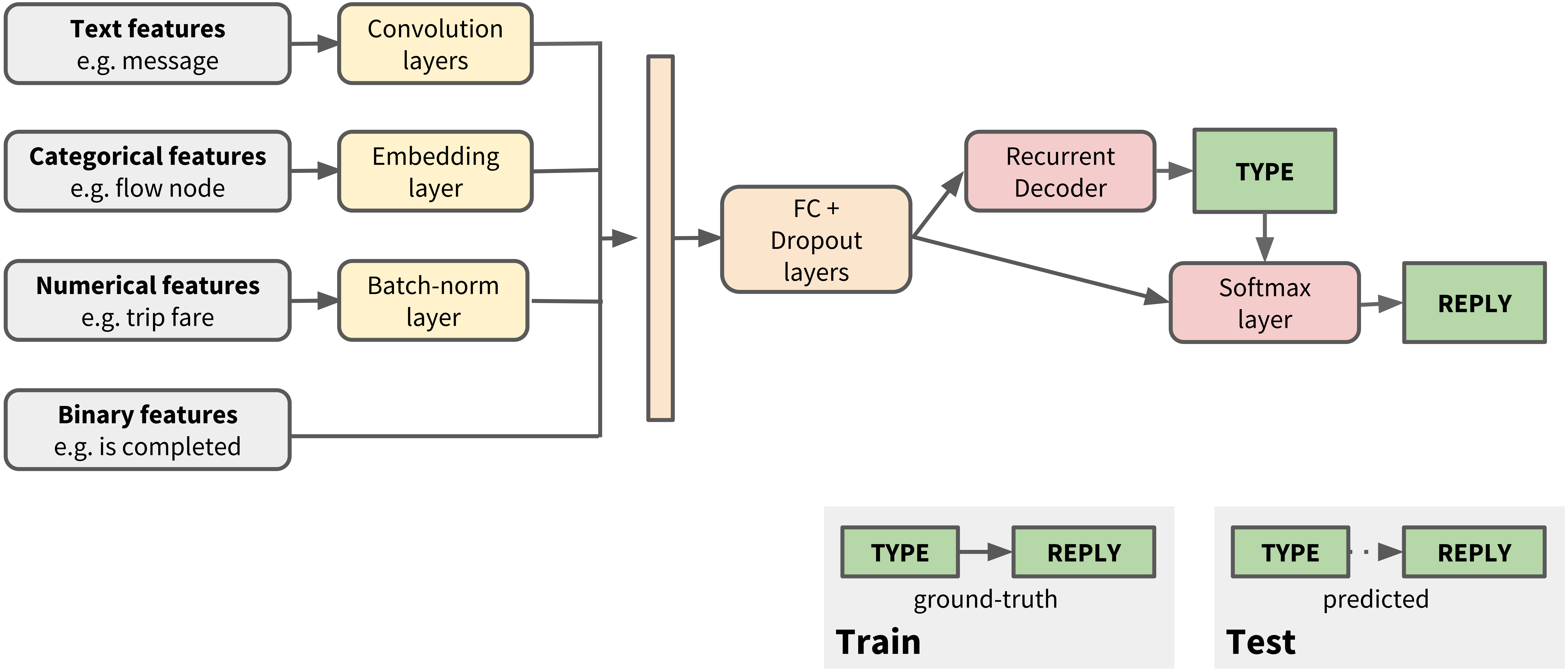
Use a Recurrent Decoder to predict **sequences of nodes** in the contact type tree

Pick the last class before <eos> as prediction

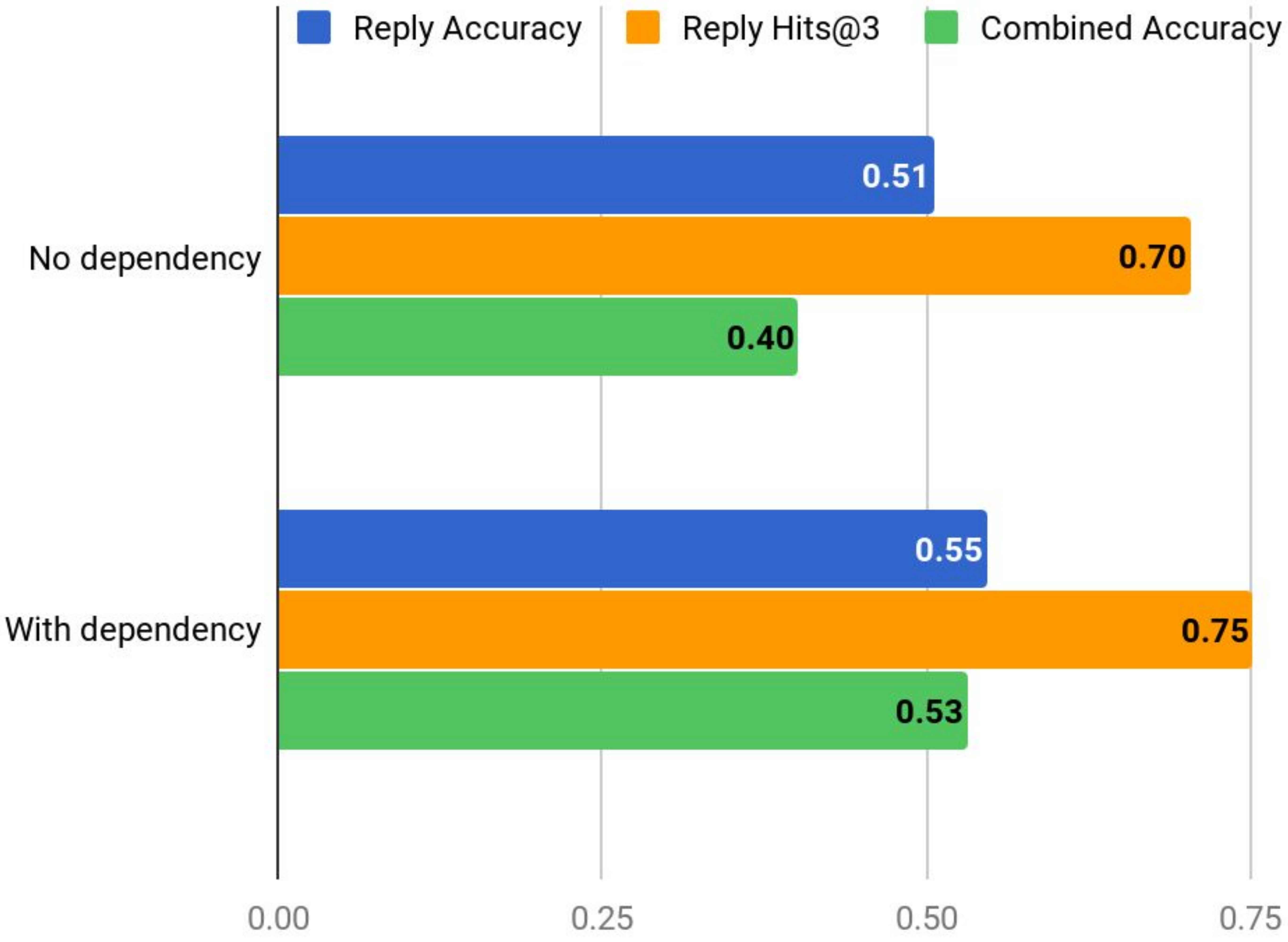
Model makes **more reasonable mistakes**

Final Architecture

Multi-task sequential learning



Effect of Adding Dependencies Between Tasks



Adding the dependency from Type to Reply **improves accuracy**

It also improves a lot the **coherence** between the two models, **increasing combined accuracy** consistently

Combined accuracy computed requiring both Type and Reply model to be **correct at the same time**

Outline

Motivation and Solution

- Complexity of Customer support @Uber

COTA v1: Traditional ML / NLP Models

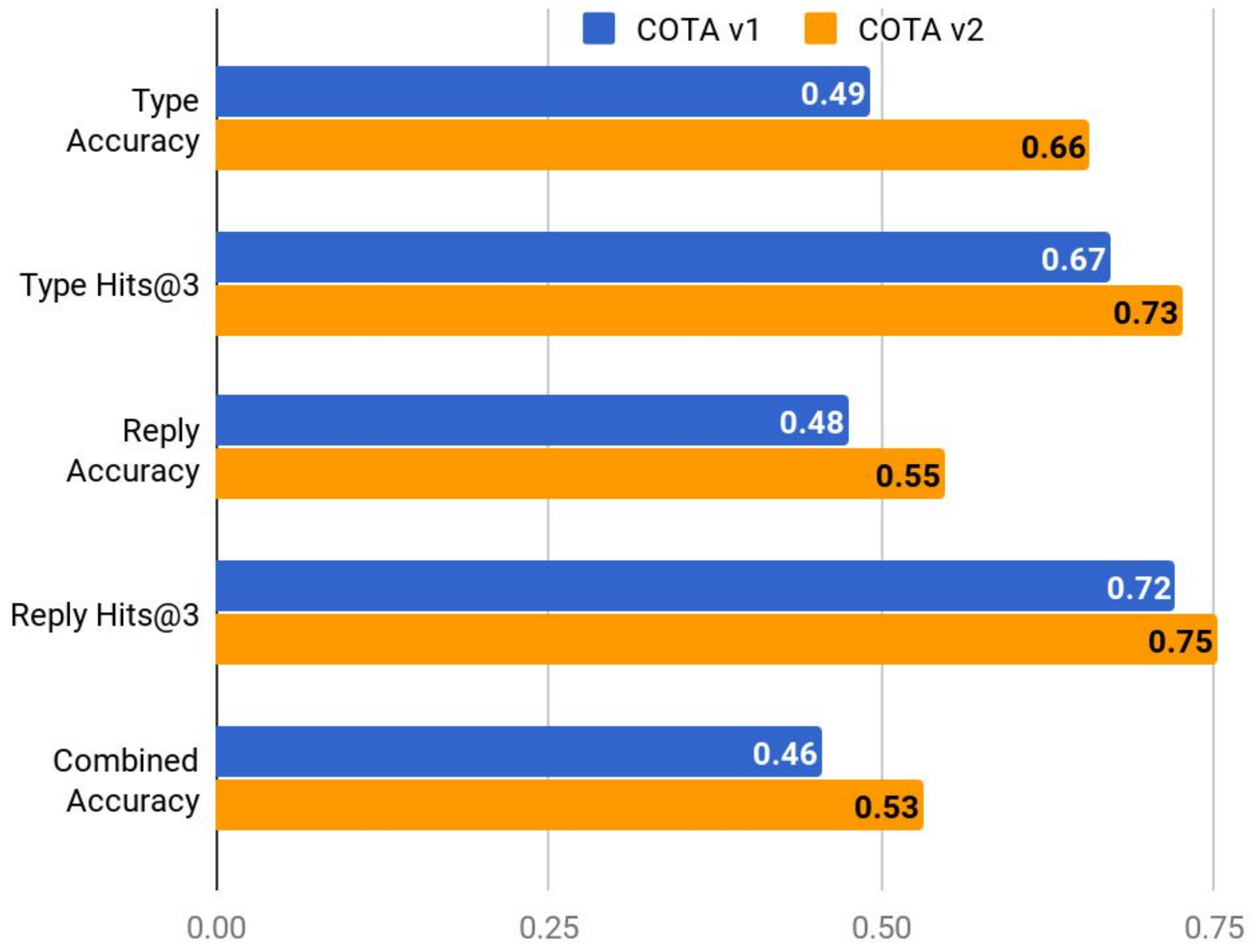
- Multi-class Classification vs Ranking

COTA v2: Deep Learning Models

- Deep learning architectures

COTA v1 vs COTA v2

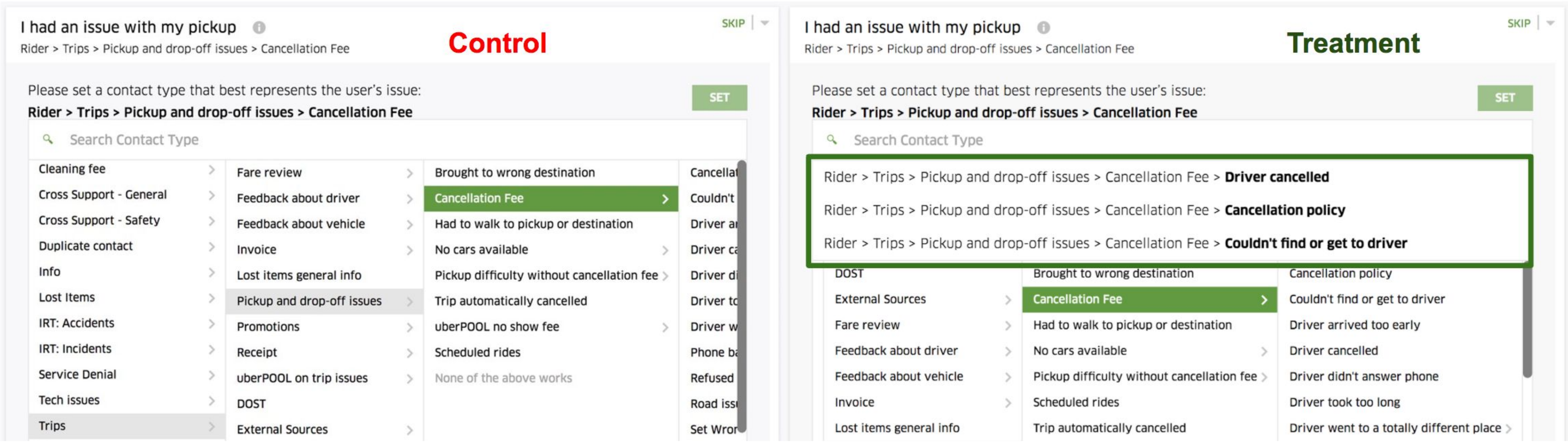
COTA v1 vs. COTA v2 offline comparison



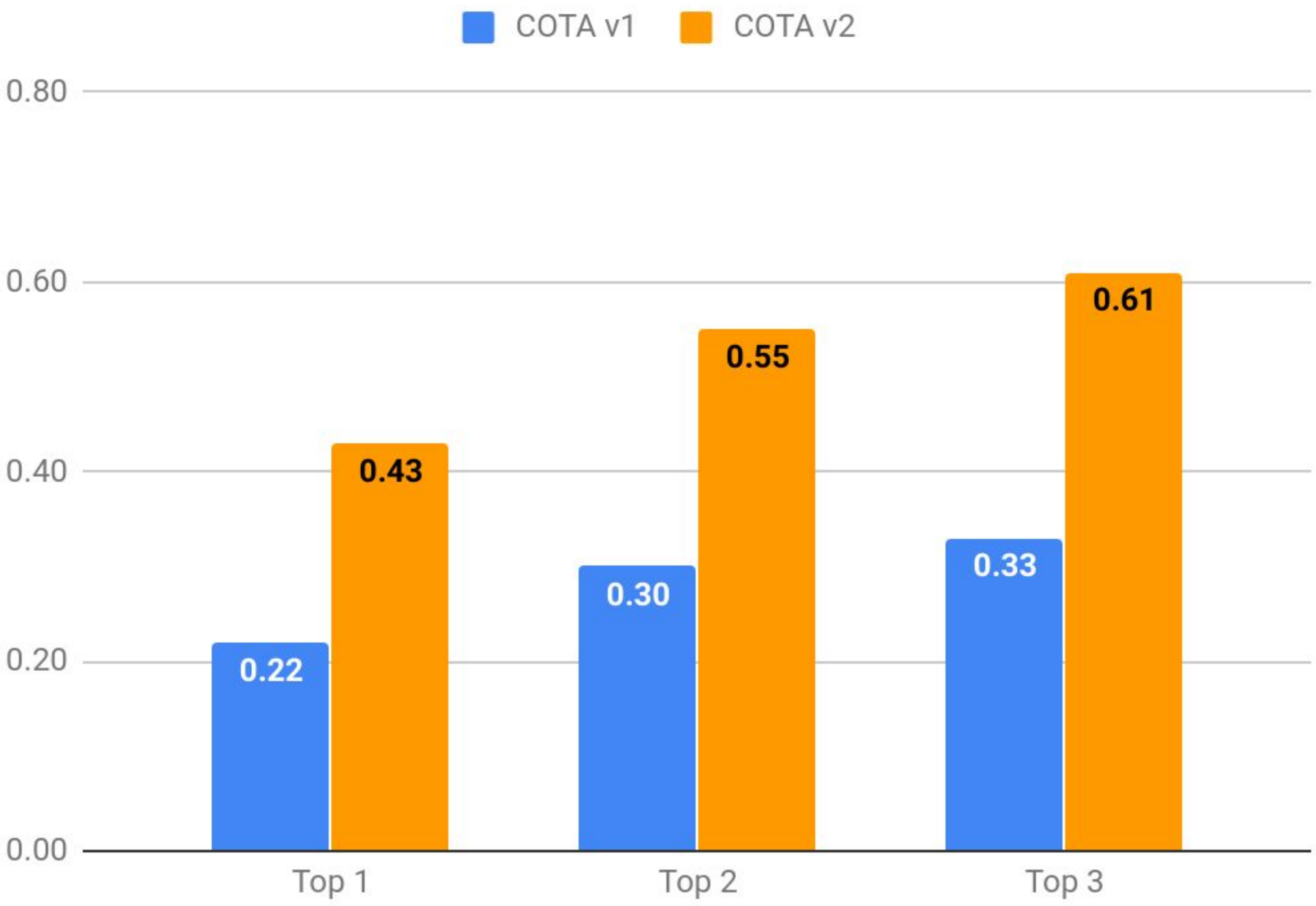
COTA v2 is **consistently more effective** than COTA v1 on **all metrics** for **both models**

The combined accuracy in particular shows an absolute **~+9%** (relative **+~20%**)

COTA v1 vs. COTA v2 A/B Test



COTA v1 vs. COTA v2 A/B Test

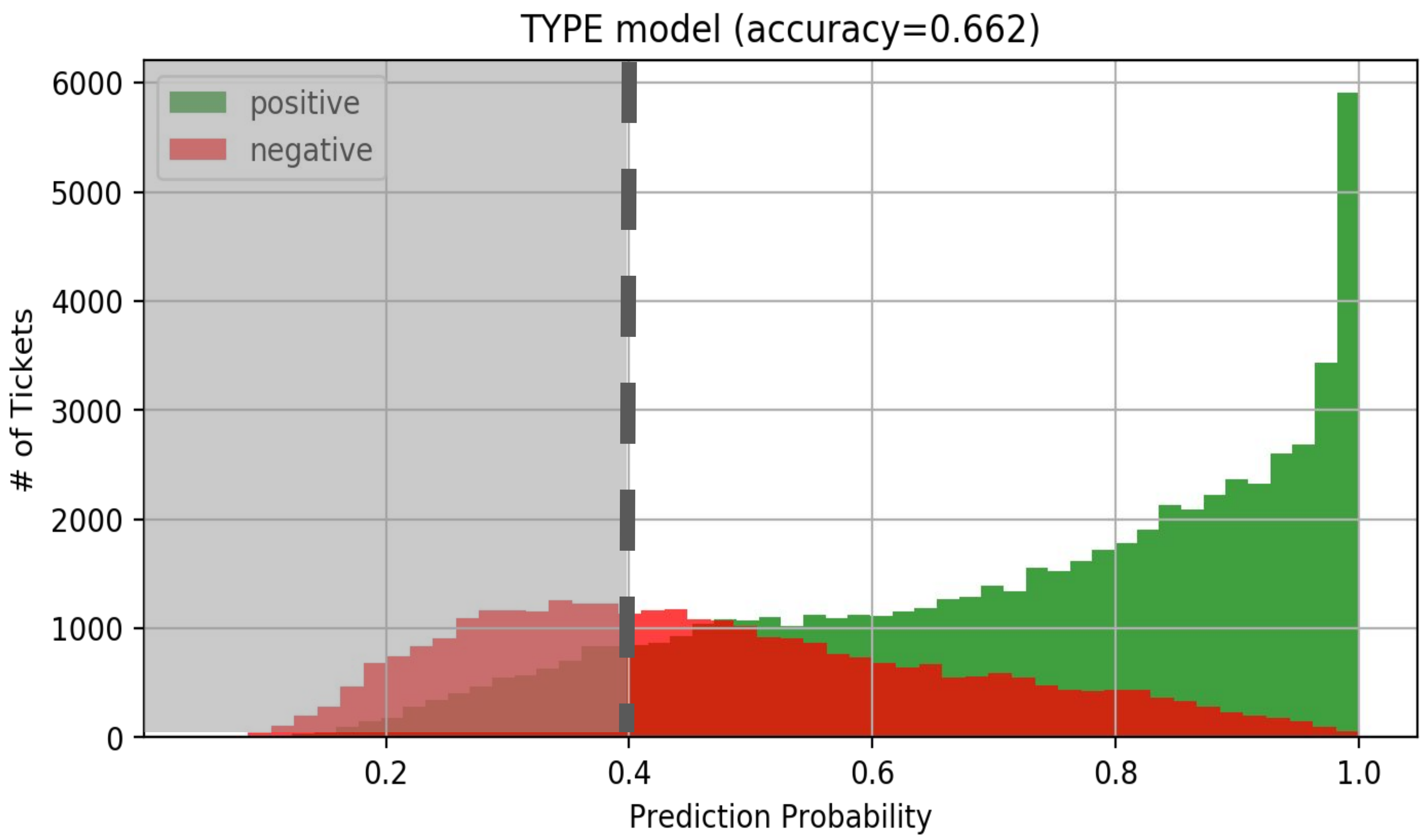


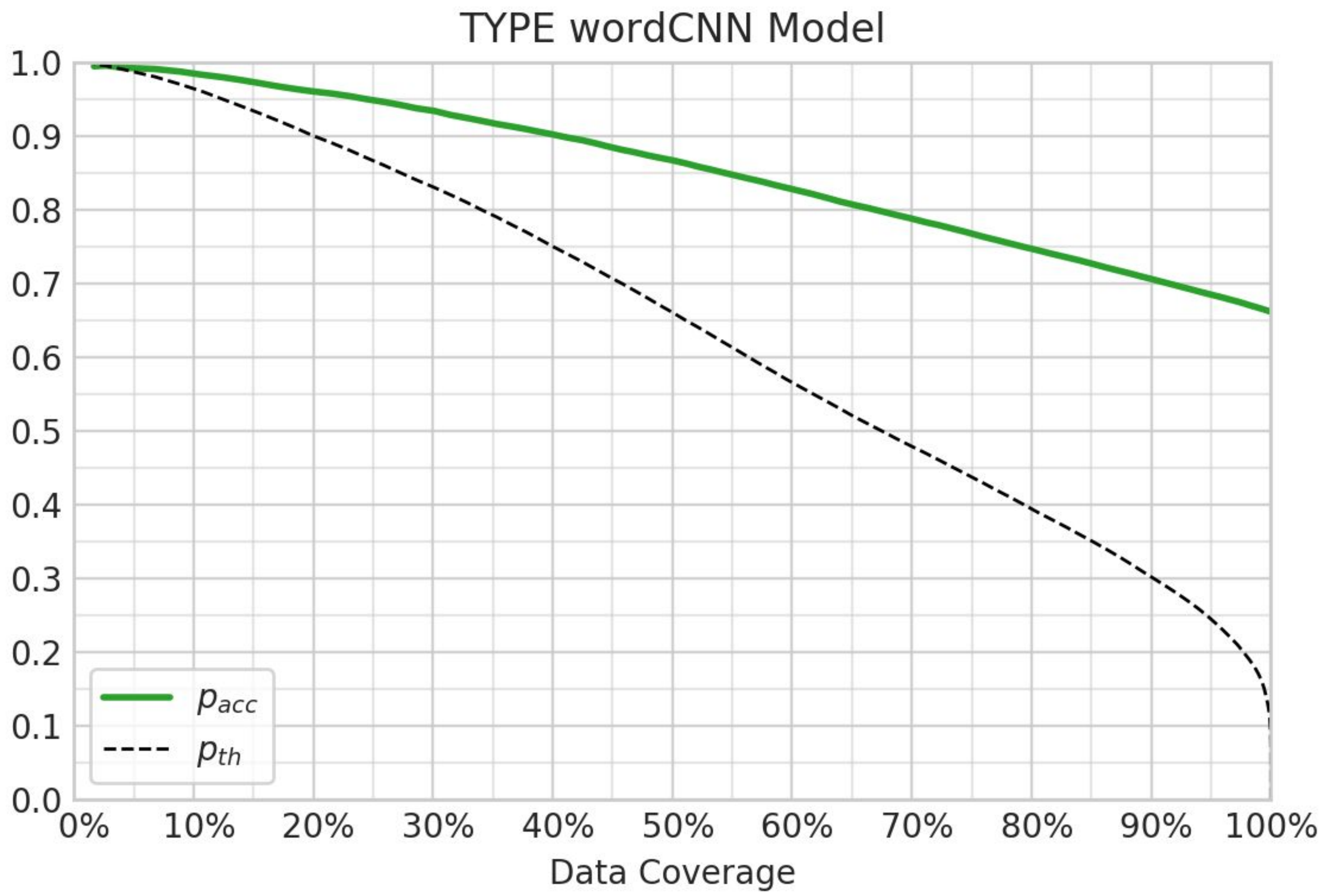
COTA v2 is **20-30% more accurate** than COTA v1 in online A/B tests

COTA v1 **reduces handling time** of **~8%,** while COTA v2 provides an additional **~7% reduction**, more than ~**15% overall reduction**

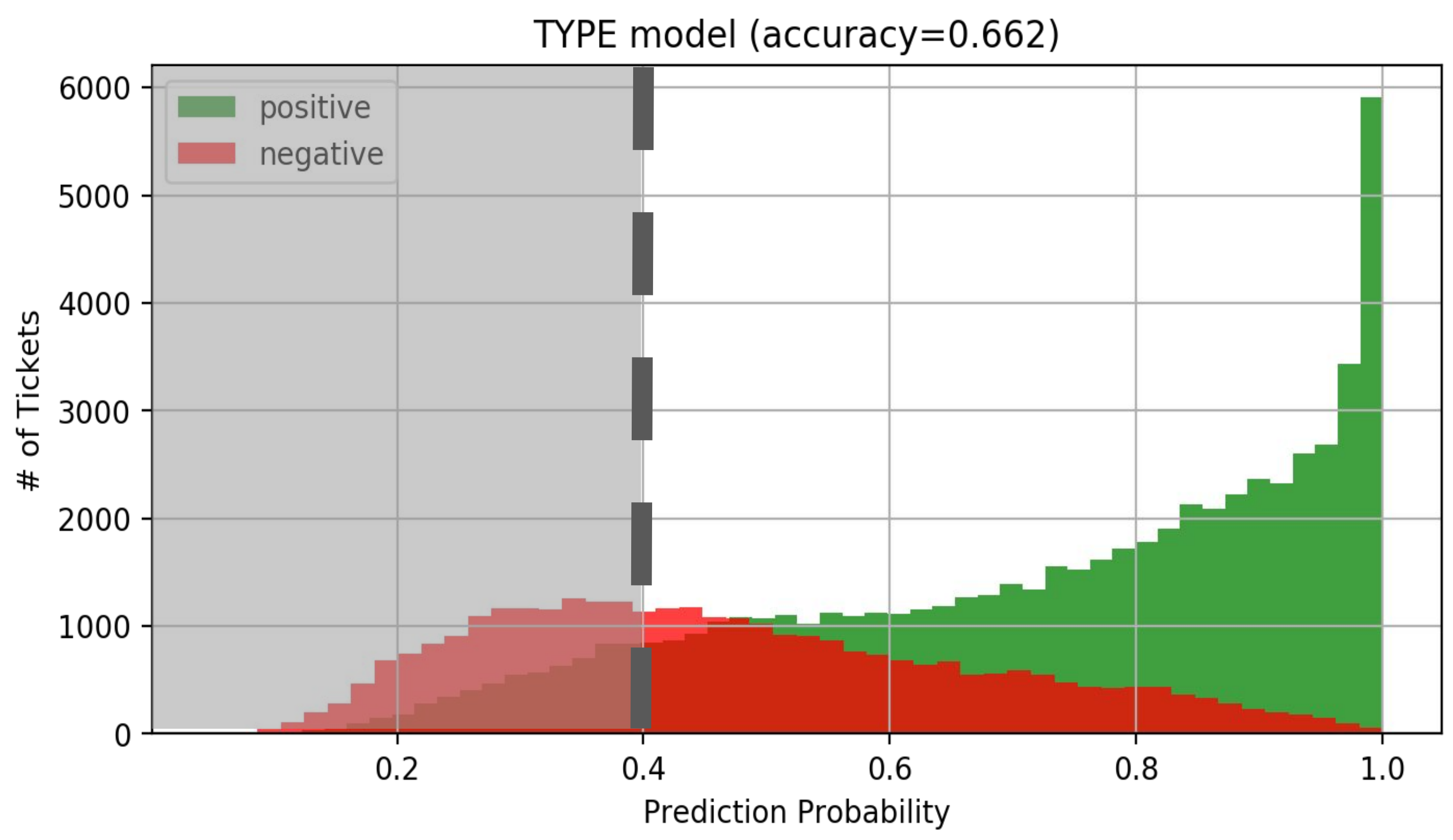
Statistically significant **customer satisfaction improvement**

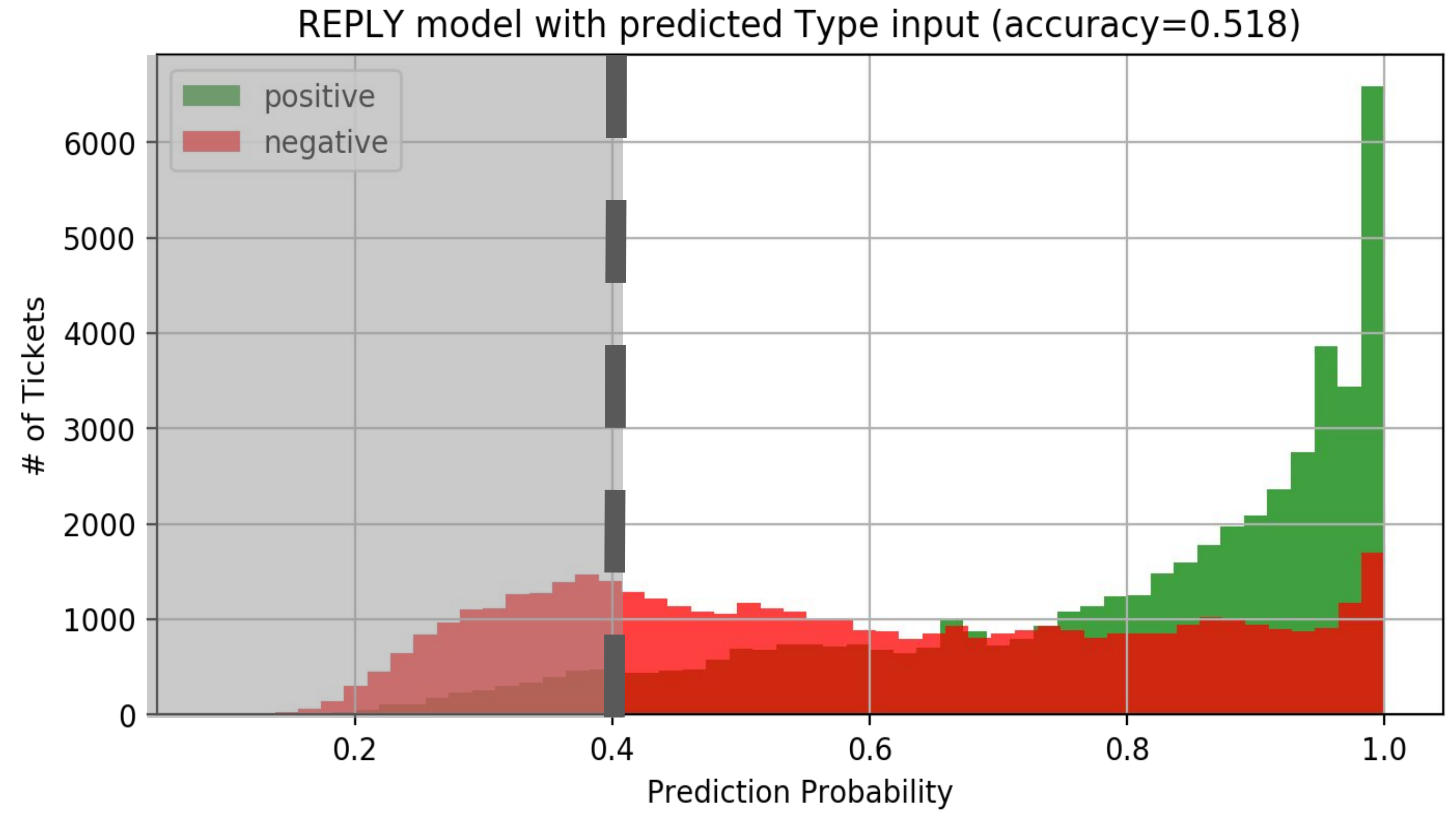
Threshold on Type Model Confidence

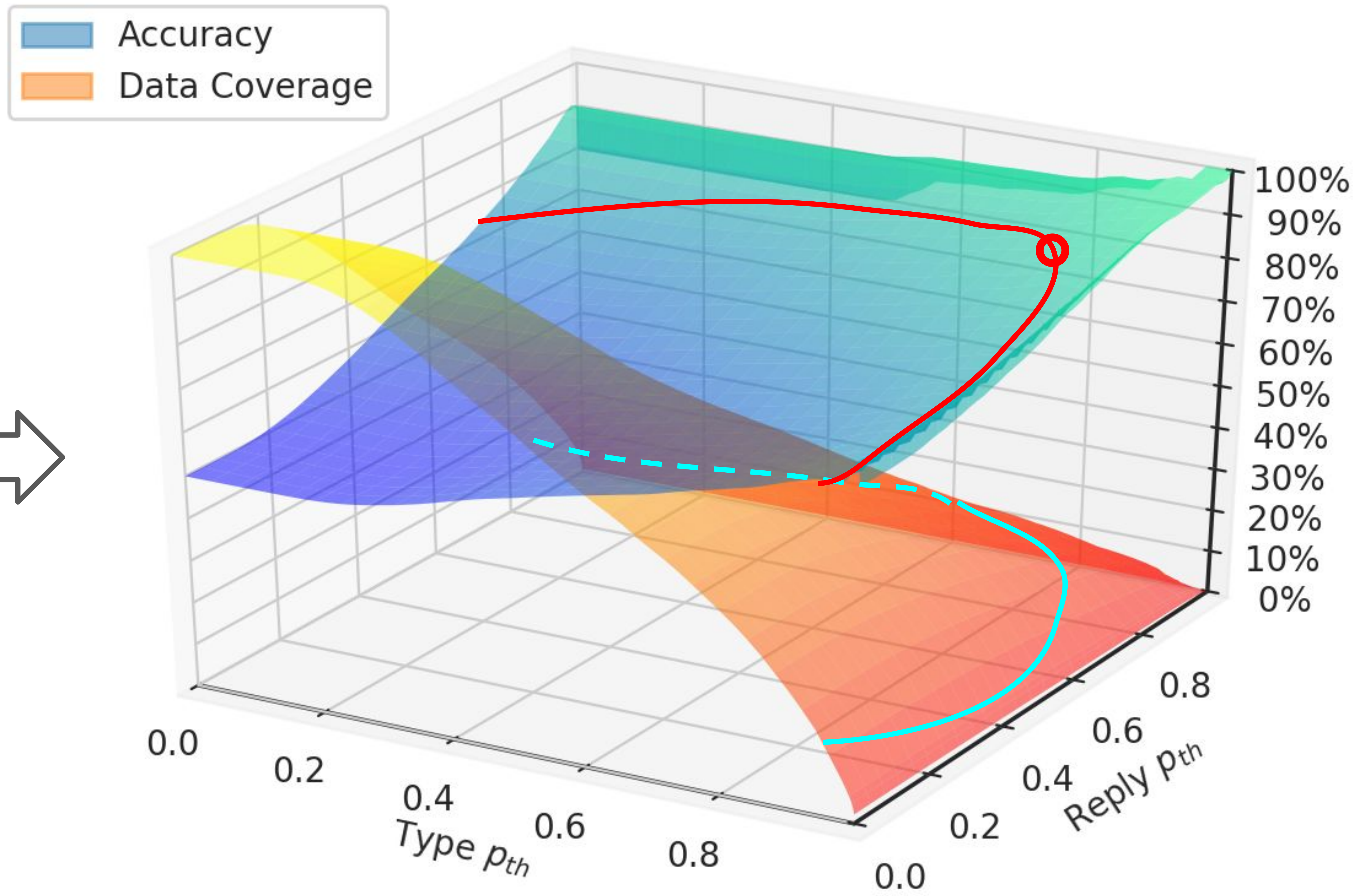




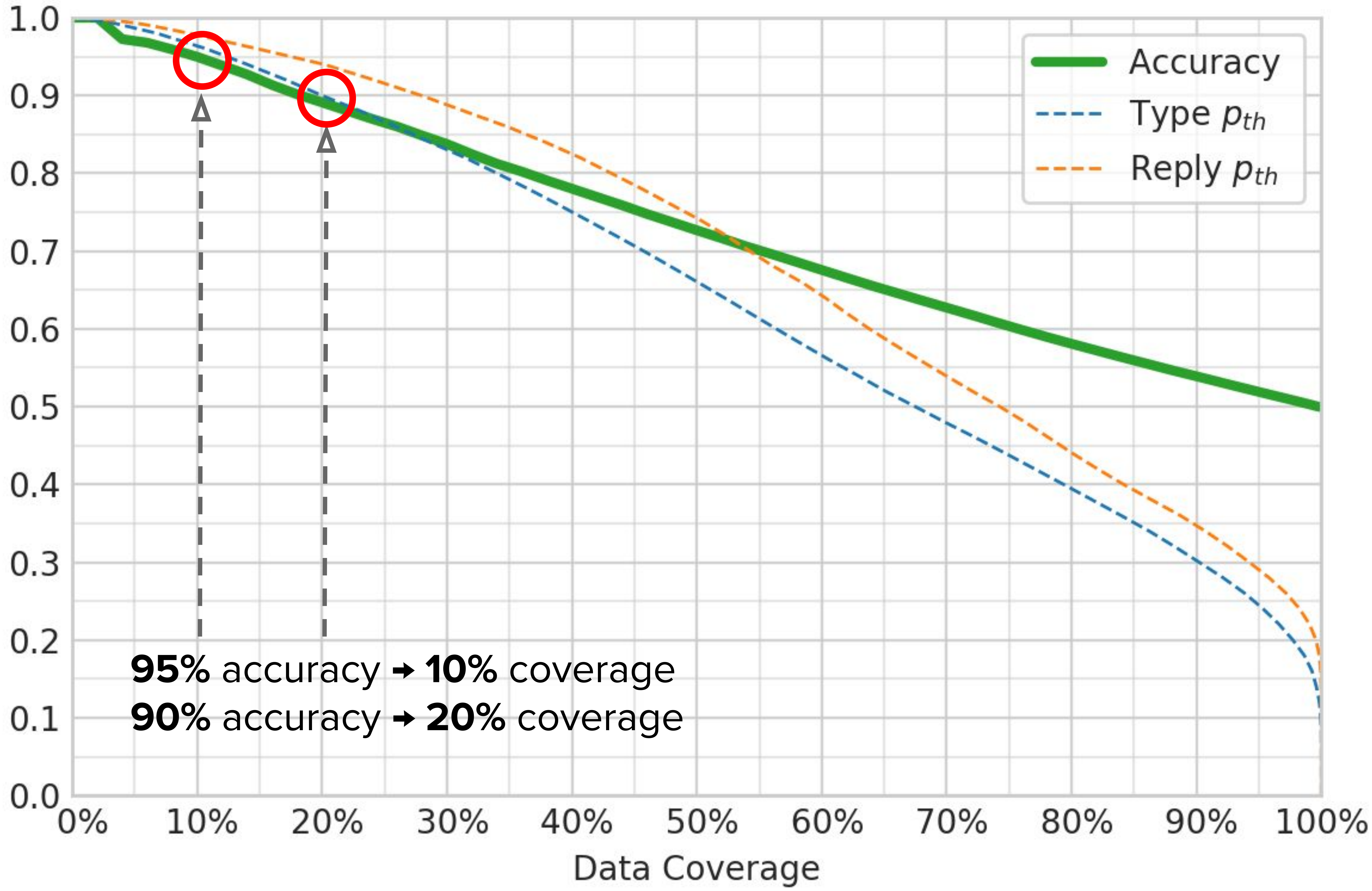
Threshold on Both Models' Confidence







Coverage vs. Maximum Accuracy



Conclusions

Using NLP & ML COTA makes customer care experience **faster** and **more accurate** while **saving Uber millions of $**

Moving from traditional to deep learning models, we observe a substantial **performance boost** (up to **30%**)

Using intelligent suggestions we were able to **reduce ticket handling time without impacting customer satisfaction**

Model degradation

Distribution shift in the real world

- Bugs get solved, probability of a issue type can decrease

- New products can be added (UberPool) so new issue types appear

Older data becomes noise

- We often talk about distribution shift in the test set, but the test set of a month ago is the training set now

Retraining Strategy

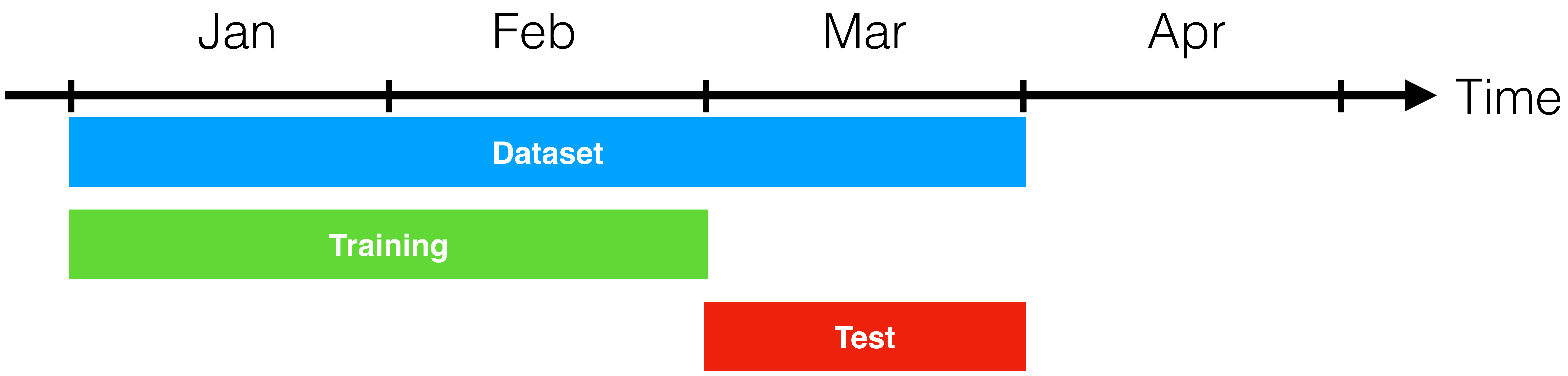
Dealing with distribution shift is an **open research topic**

In practice in most cases the safest route is just
retraining the model

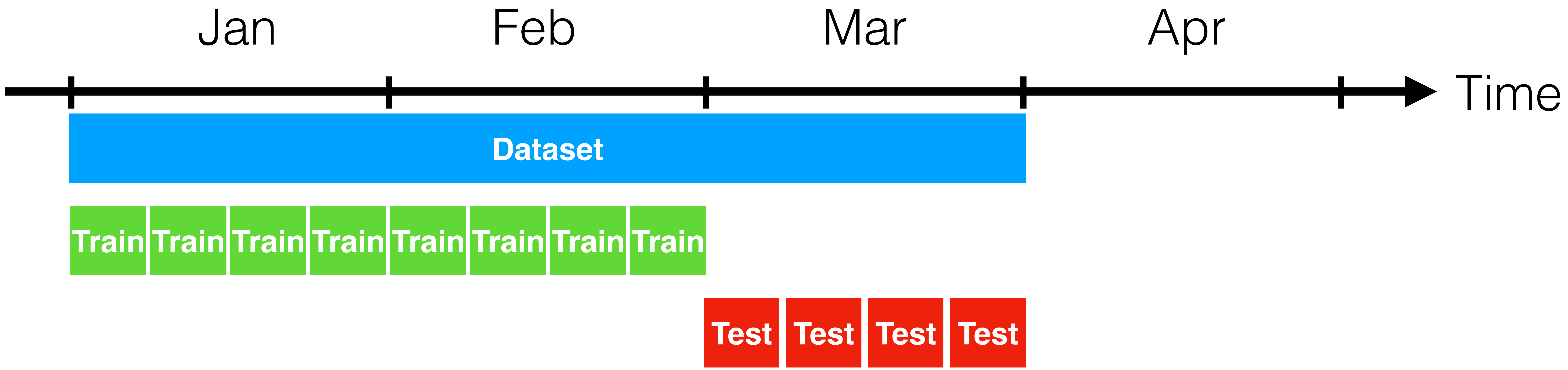
But...

- How often to retrain?
- What triggers retraining?
- With how much data?

Offline simulation: time-based split



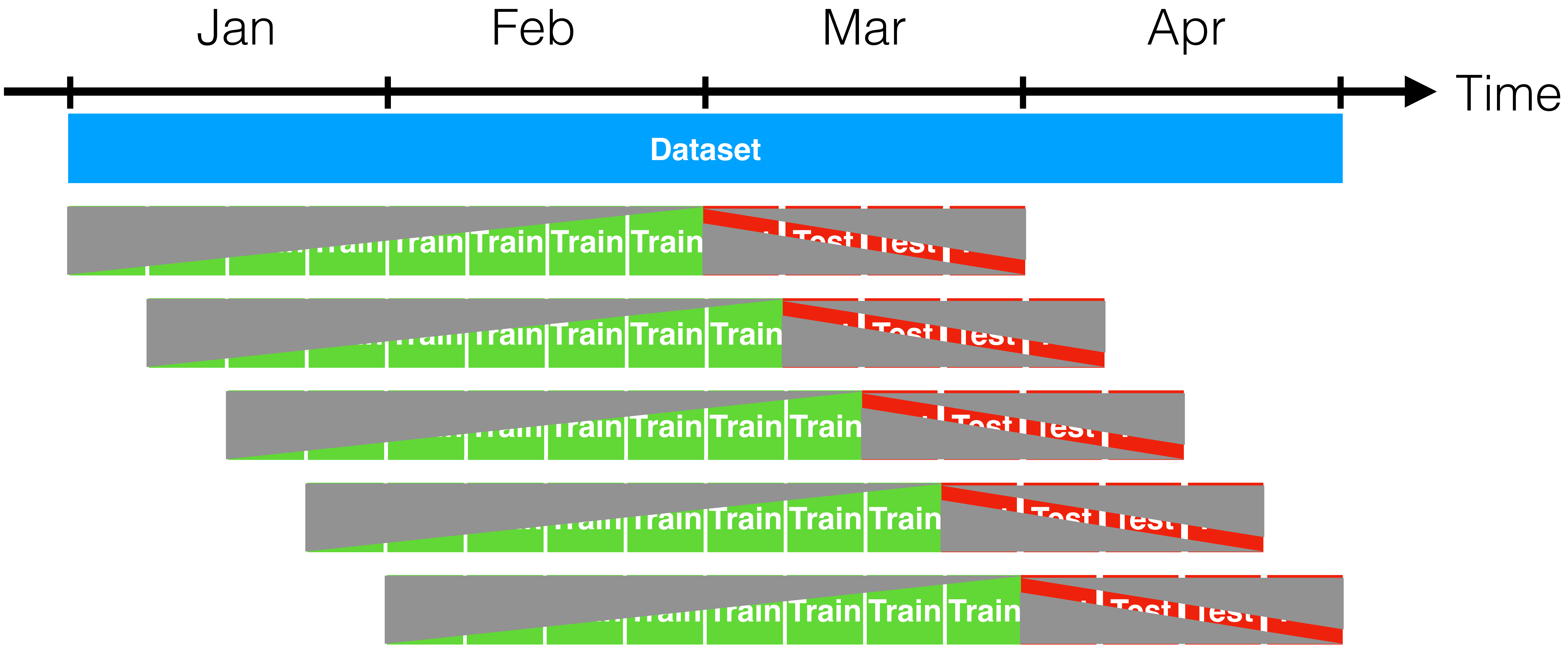
Offline simulation: split in weeks



Offline simulation

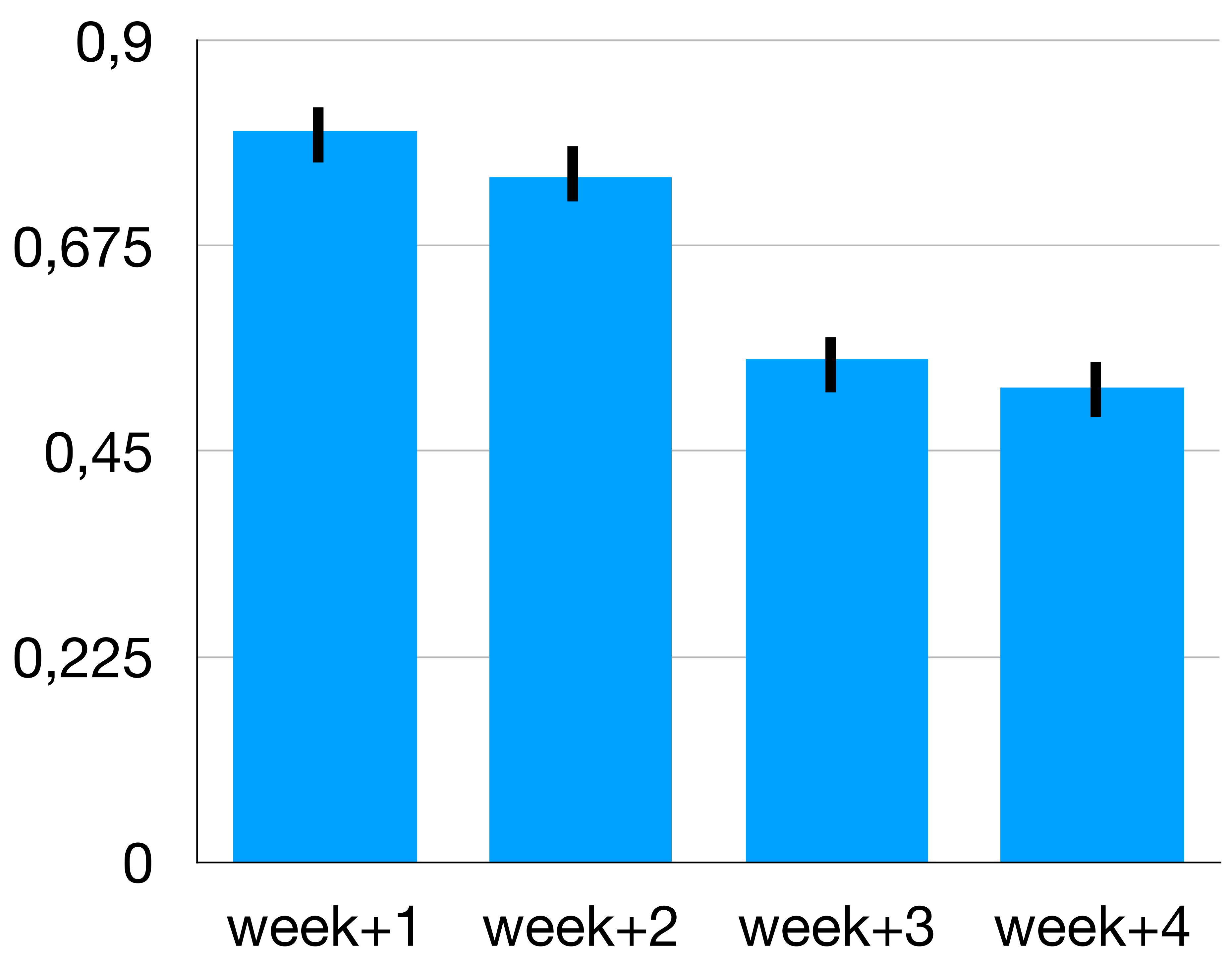
Offline simulation

Offline simulation

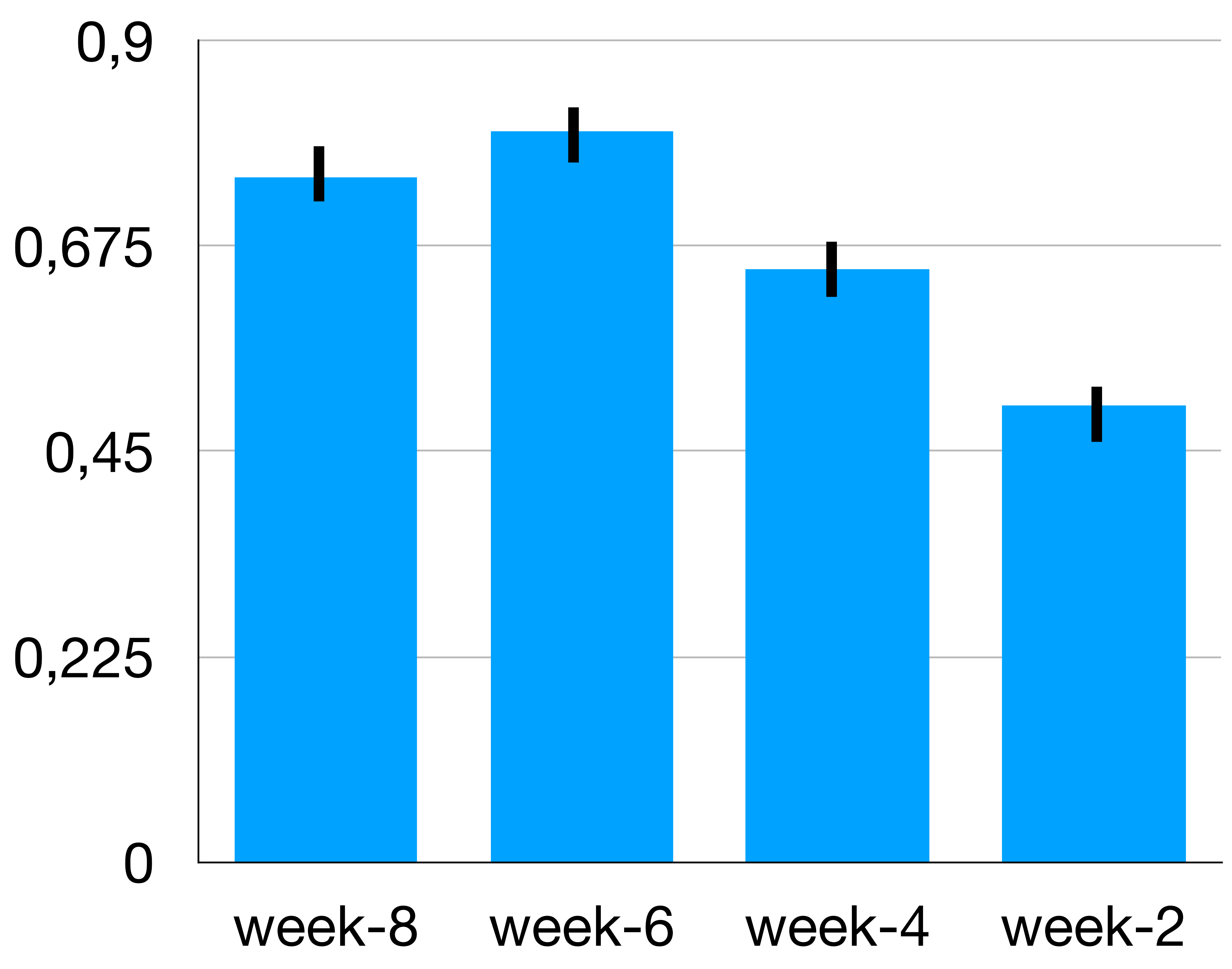


Retraining Strategy

How often to retrain?



With how much data?



Online Retraining

What triggers retraining?

Used learnings from offline simulation

Retrained **when performance dropped** below performance on the test set at training original training time - 8% (relative)

Retrained with **1.5 months** of training data, as we learned from the offline simulation that more was detrimental to performance

COTA Team

Cross-functional collaboration

AI Labs
Applied Machine Learning
Customer Obsession
Michelangelo
Sensing and Perception

Enhancing Recommendations on Uber Eats with Graph Convolutional Networks

Ankit Jain/Piero Molino

ankit.jain/piero@uber.com

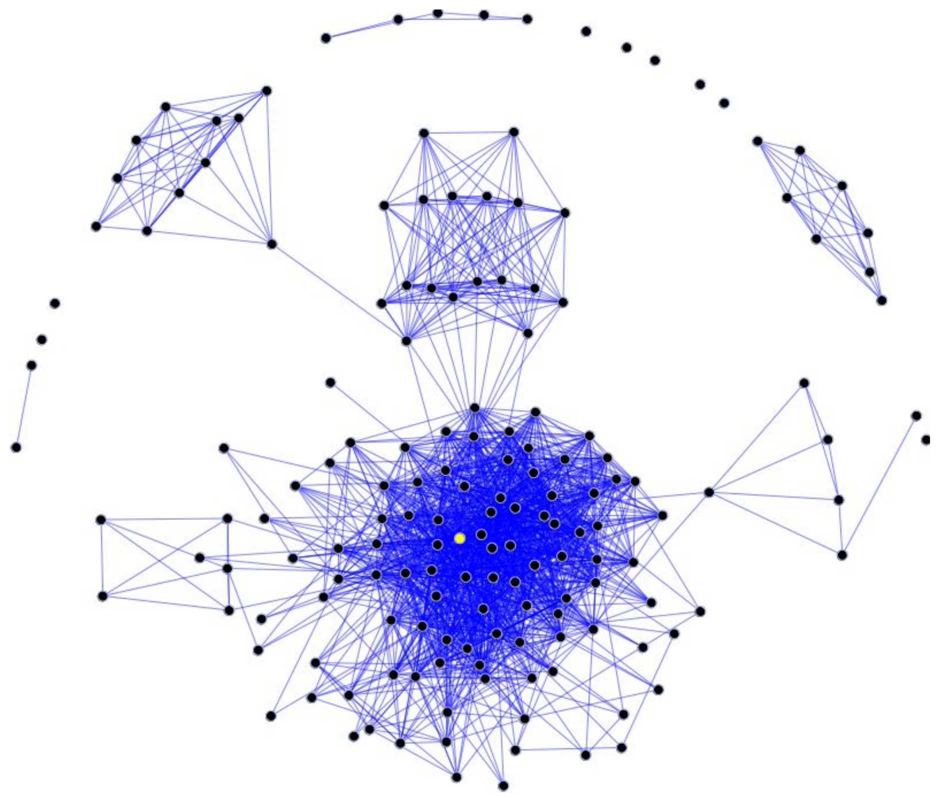
Uber AI

Agenda

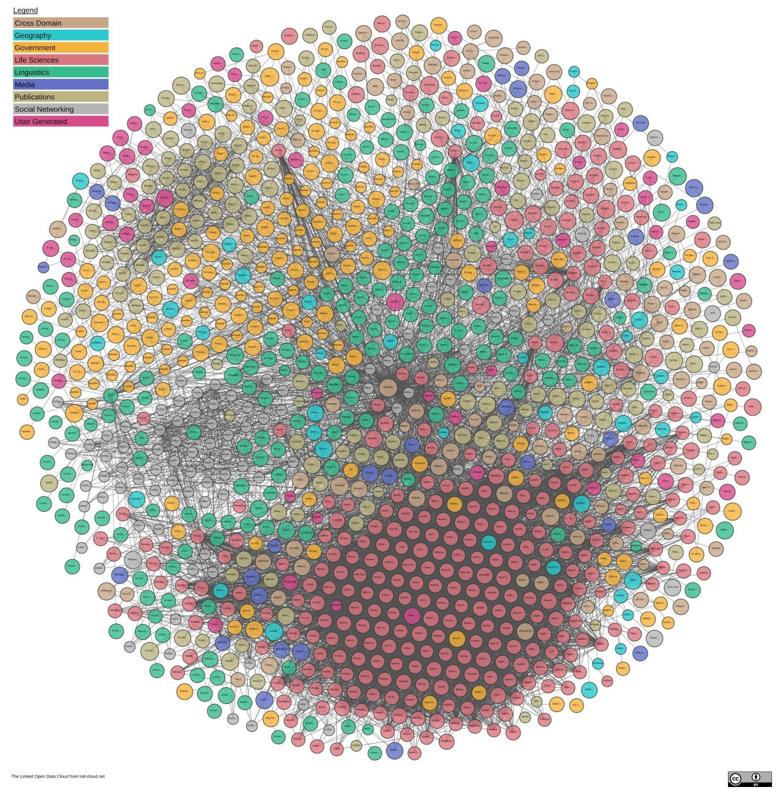
1. **Graph Representation Learning**
2. **Dish Recommendation on Uber Eats**
3. **Graph Learning on Uber Eats**

Graph Representation Learning

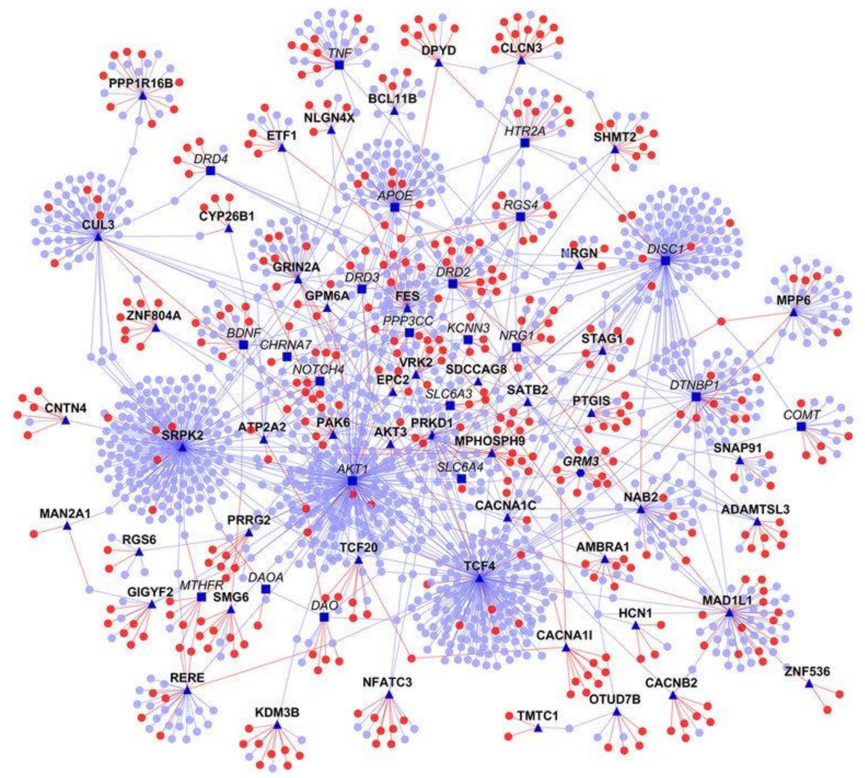
Graph data



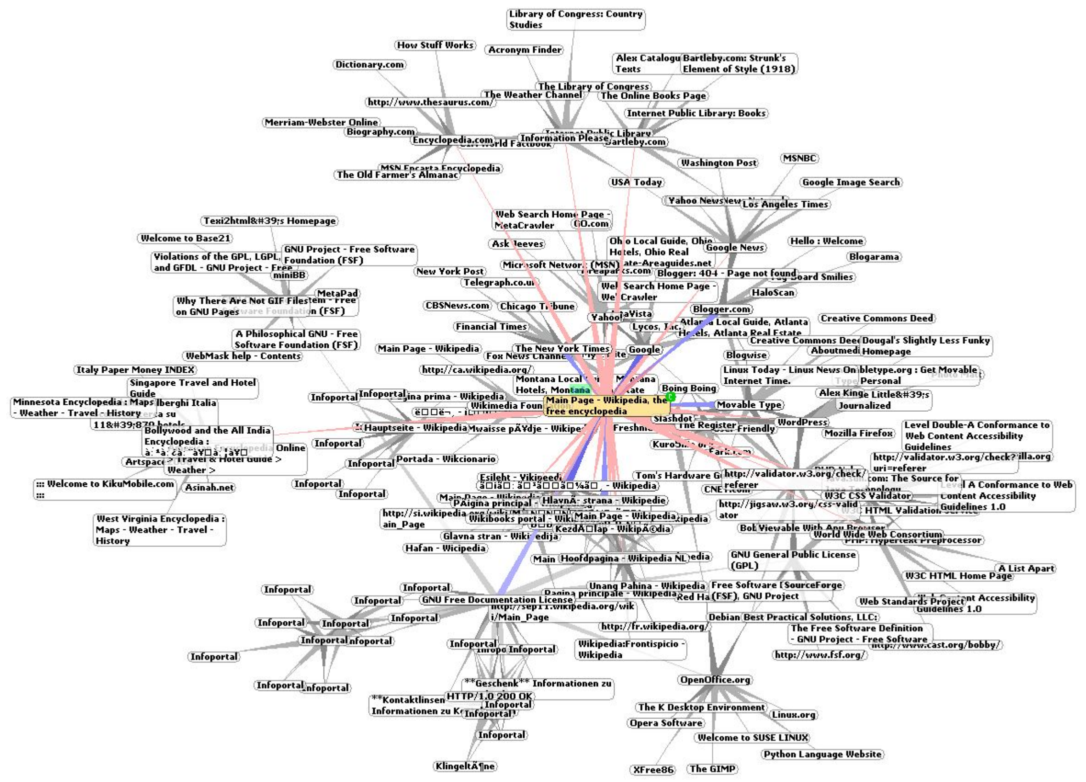
Social networks



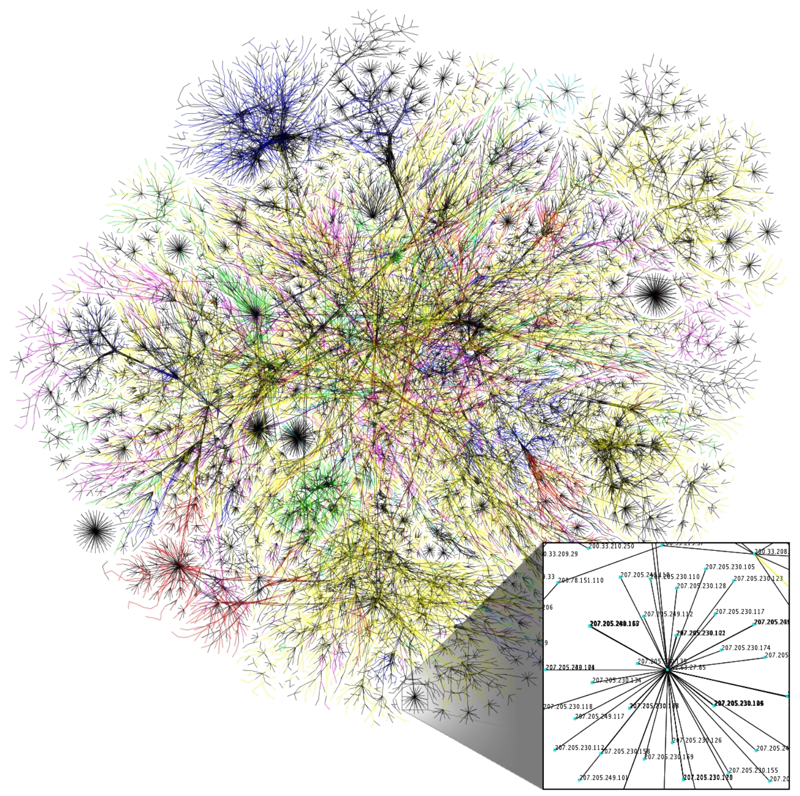
Linked Open Data



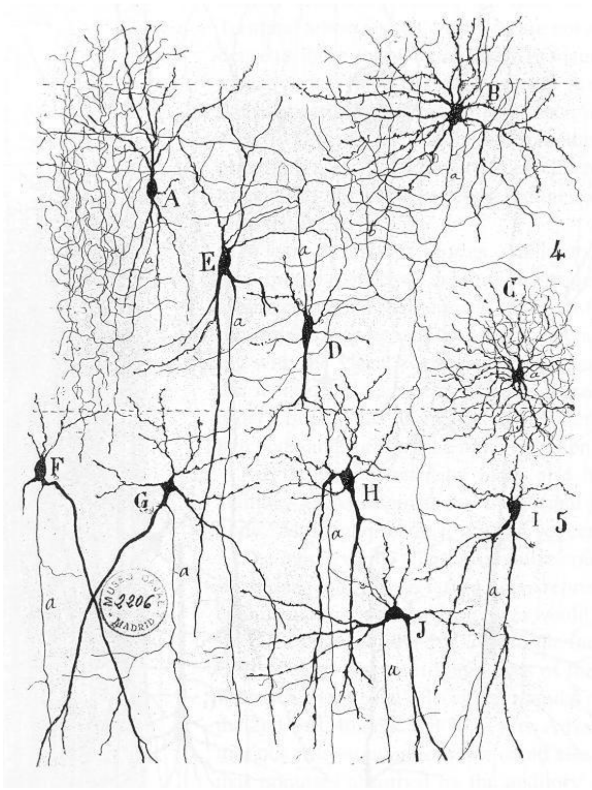
Biomedical networks



Information networks



Internet



Networks of neurons

Tasks on graphs

Node classification

Predict a type of a given node

Link prediction

Predict whether two nodes are linked

Community detection

Identify densely linked clusters of nodes

Network similarity

How similar are two (sub)networks

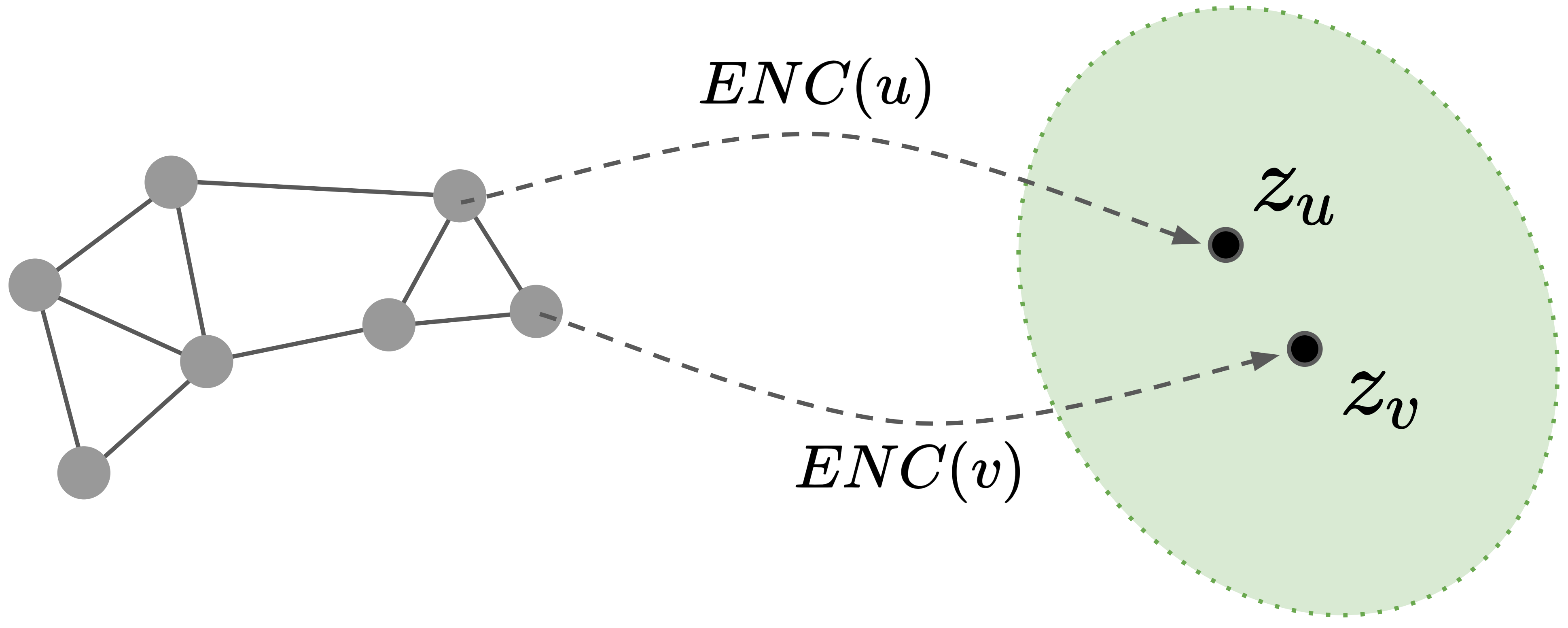
Learning framework

Define an encoder mapping from nodes to embeddings

Define a node similarity function based on the network structure

Optimize the parameters of the encoder so that:

$$similarity(u, v) \approx z_v^\top z_u$$

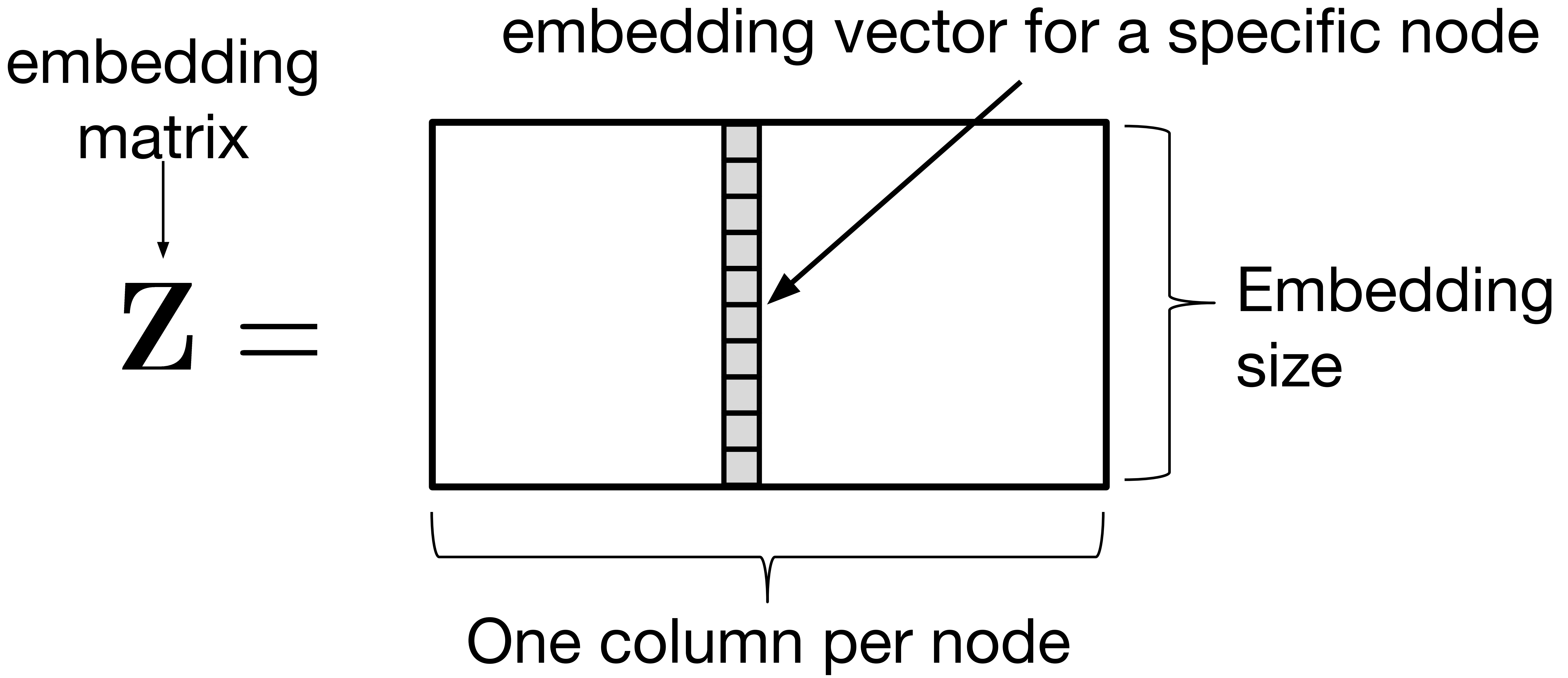


original graph **embedding space**

Shallow encoding

Simplest encoding approach: encoder is just an embedding-lookup

Algorithms like Matrix Factorization, Node2Vec, Deepwalk fall in this category



Shallow encoding limitations

O(|*V*|) parameters are needed, every node has its own embedding vector

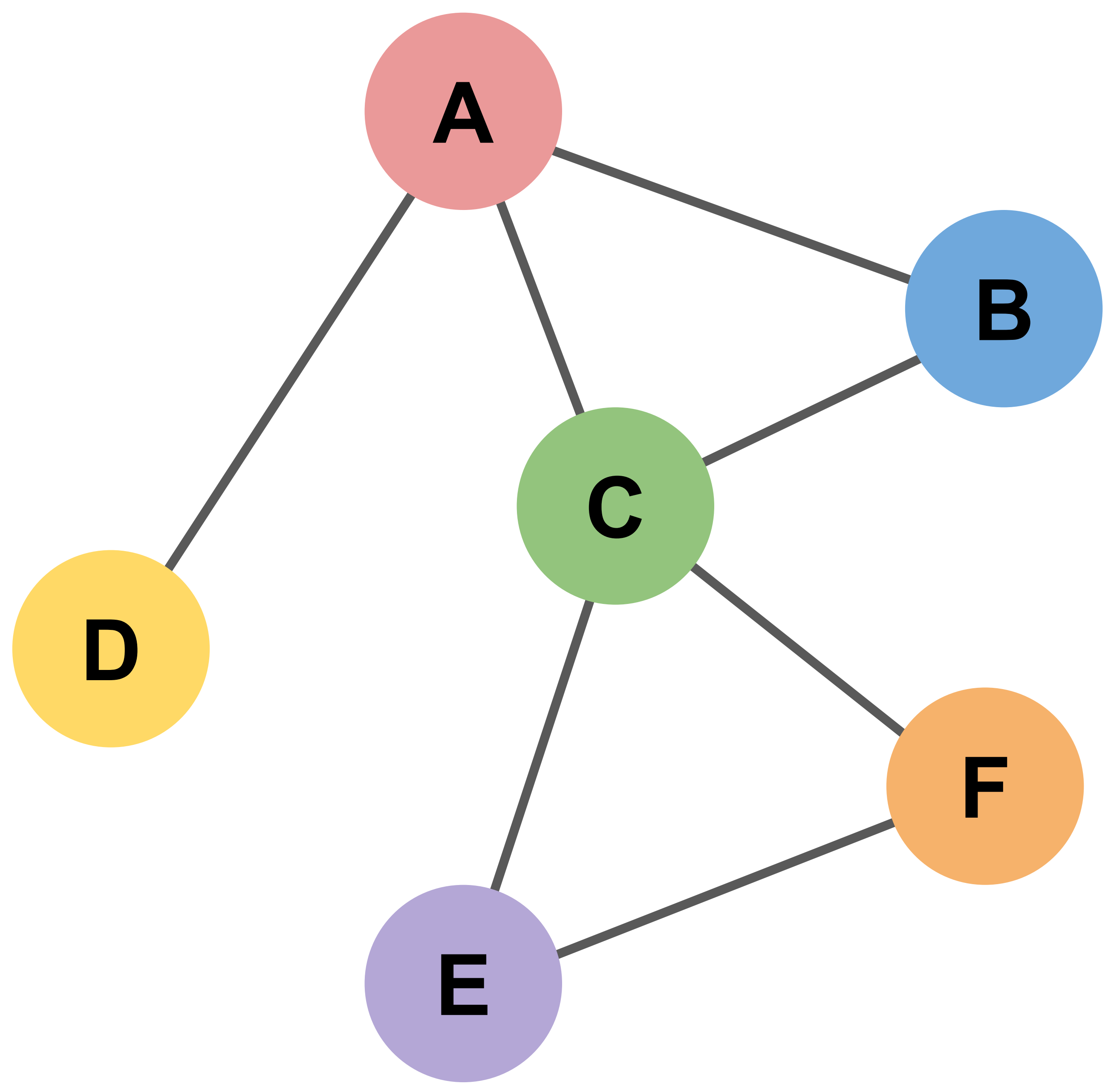
Either not possible or very time consuming to generate embeddings for nodes **not seen during training**

Does not incorporate **node features**

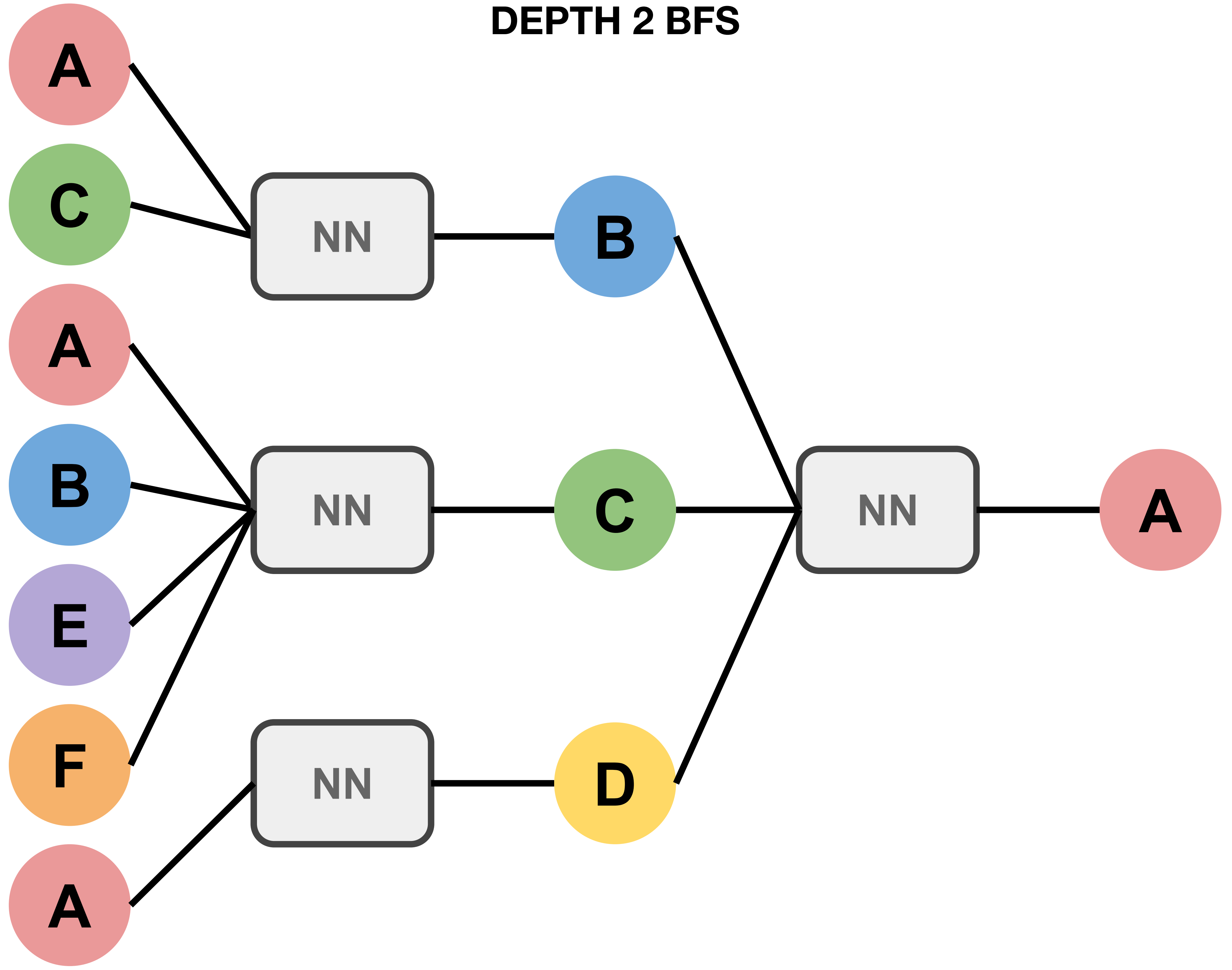
Graph Neural Network

Key Idea: To obtain node representations, use a neural network to aggregate information from neighbors recursively by limited Breadth-FIrst Search (BFS)

INPUT GRAPH



DEPTH 2 BFS

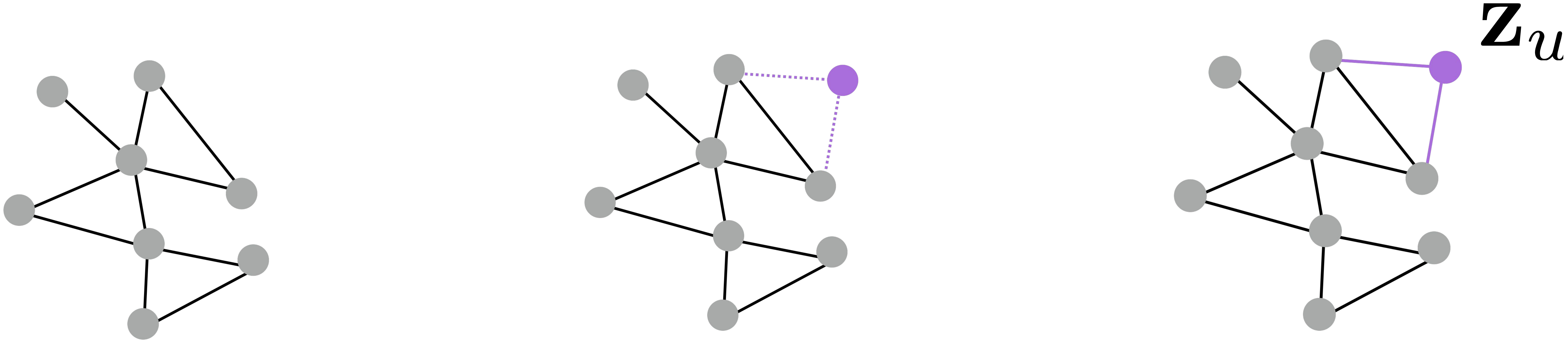


Inductive capability

In many real applications new nodes are often added to the graph

Need to generate embeddings for new nodes without retraining

Hard to do with shallow methods

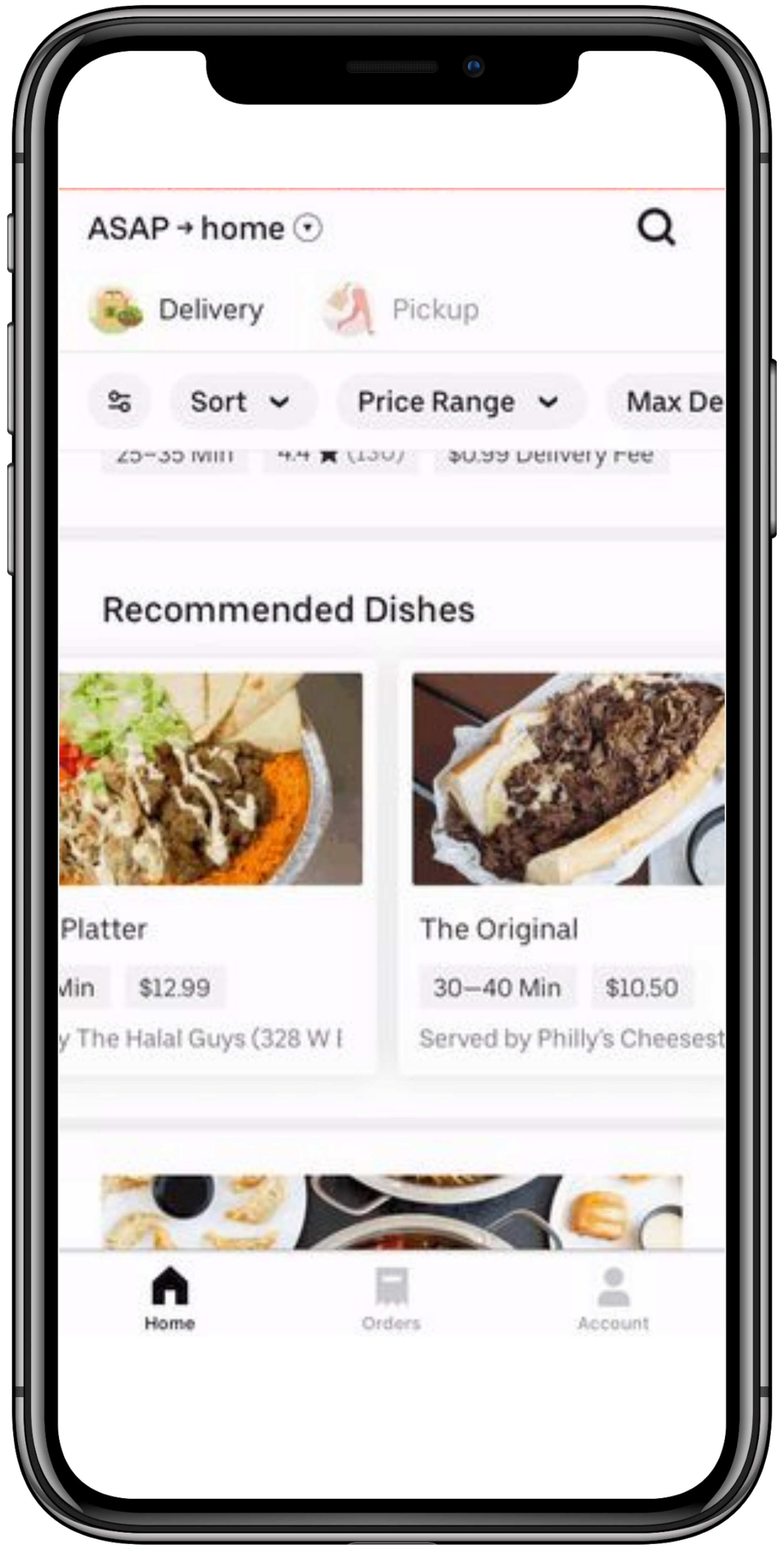


train with snapshot **new node arrives** **generate embedding for new node**

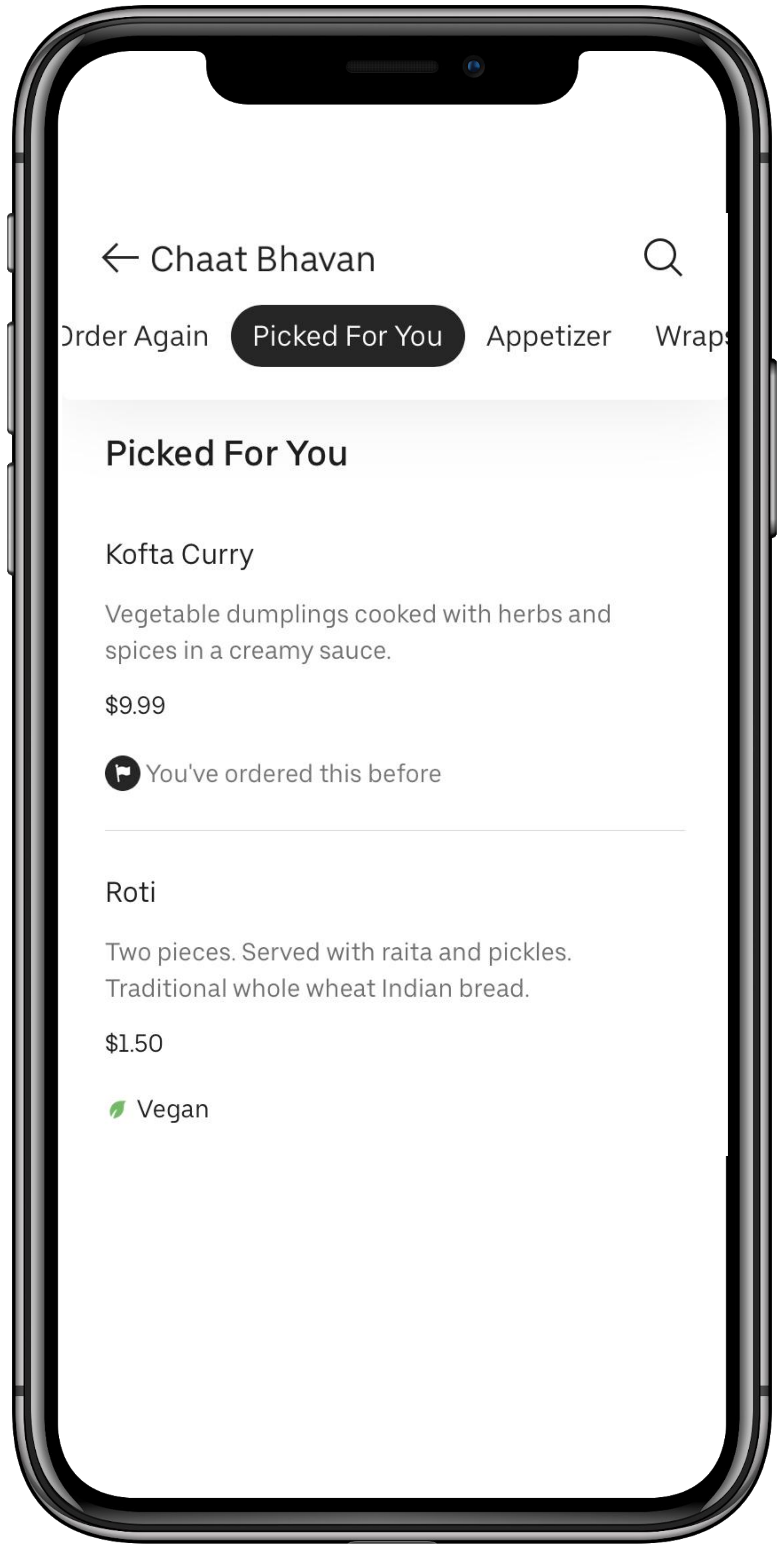
Dish Recommendation on Uber Eats

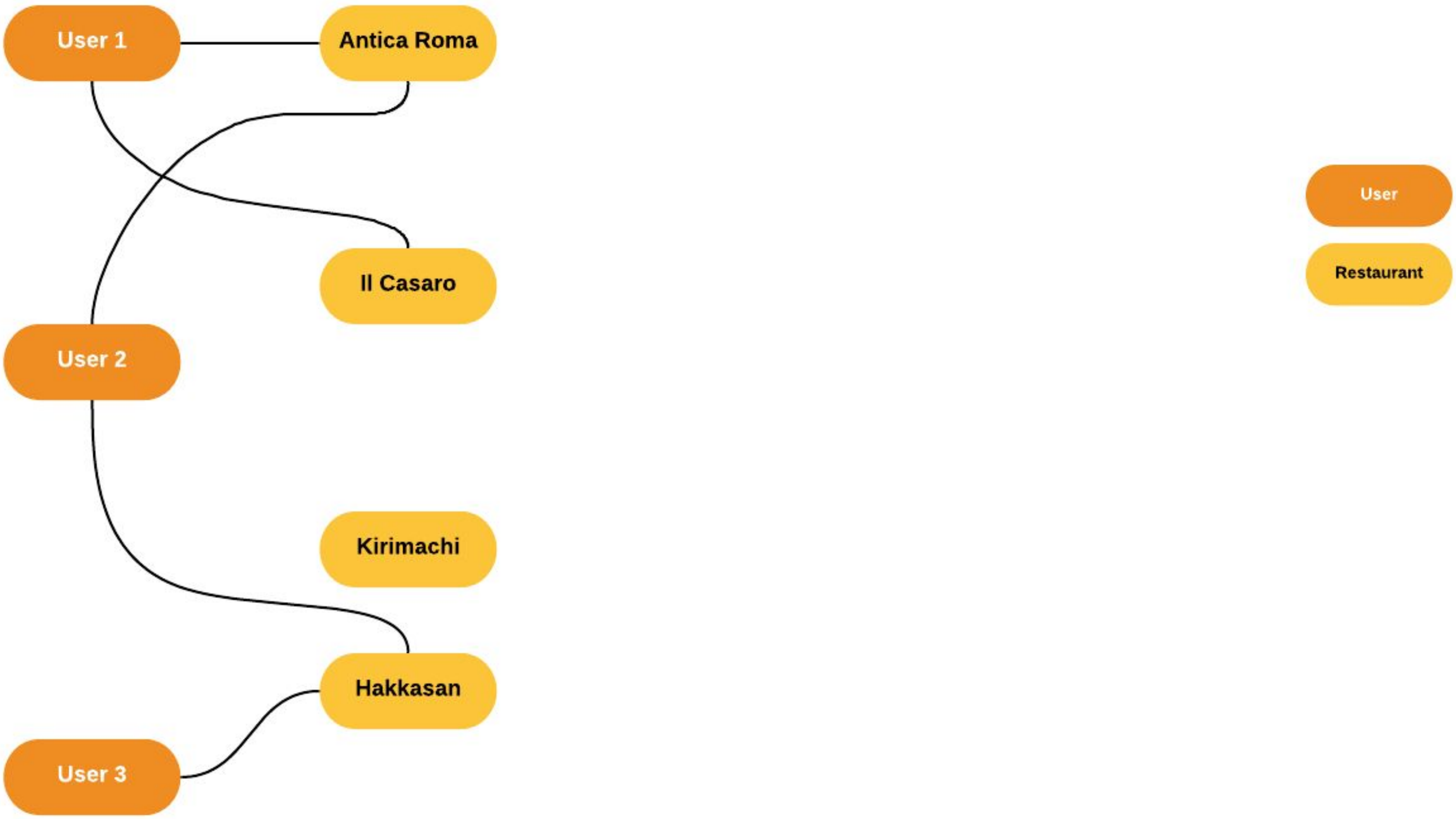
Suggested Dishes

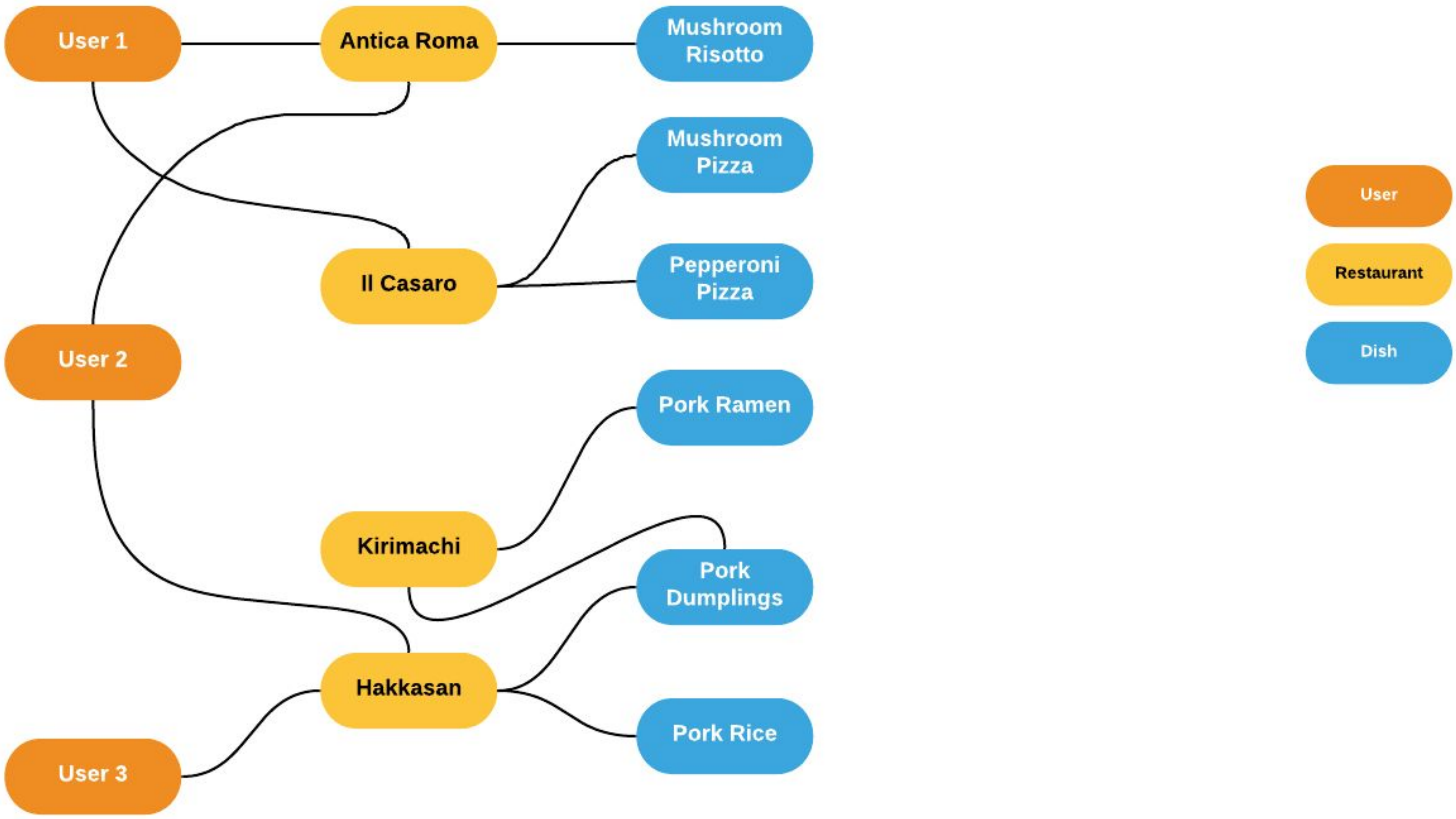
Recommended Dishes Carousel

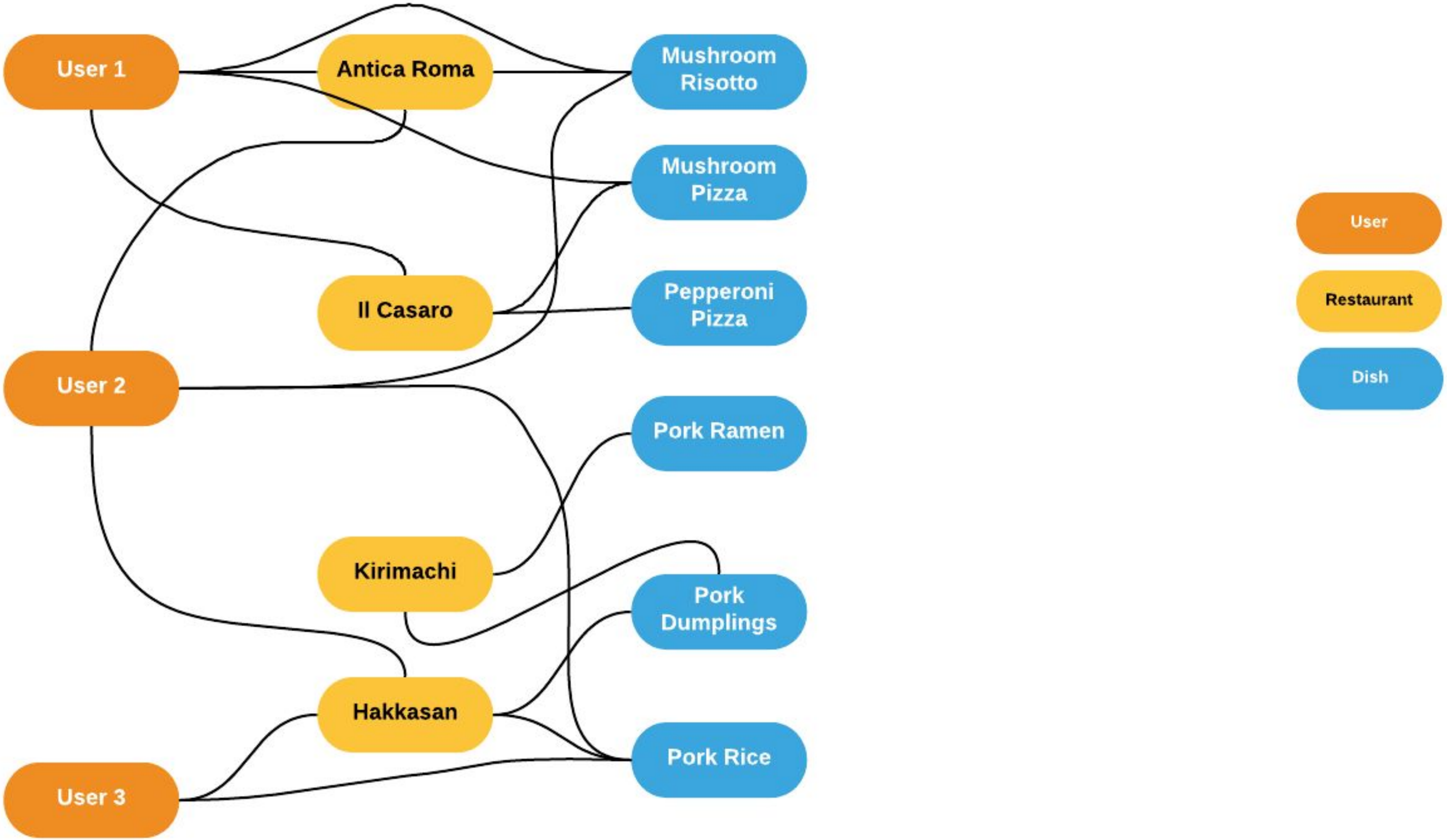


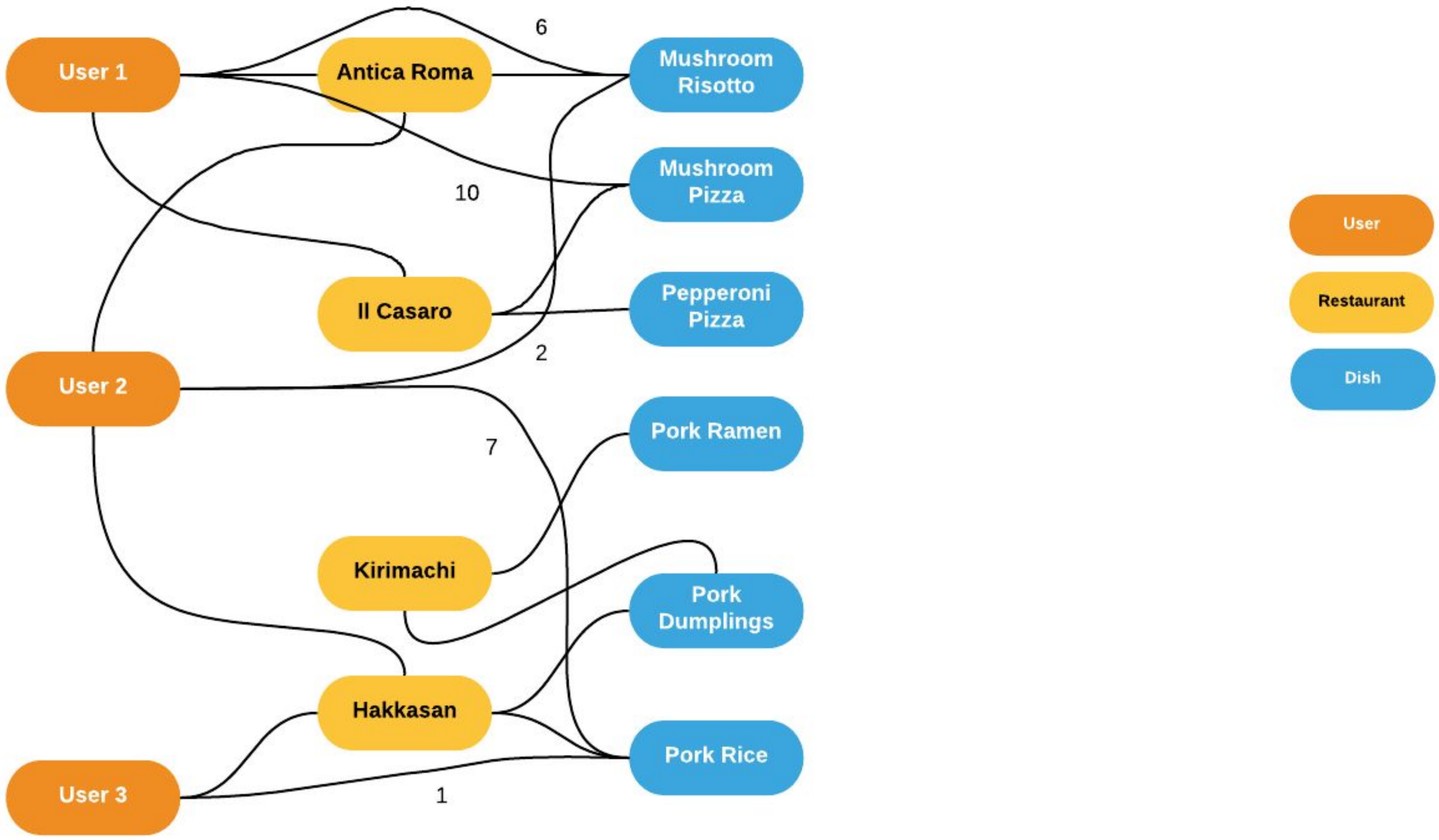
Picked for You




User 1
Antica Roma
Il Casaro
User 2
Kirimachi
Hakkasan
User 3
User
Restaurant


User 1
Antica Roma
Mushroom Risotto
Mushroom Pizza
Il Casaro
Pepperoni Pizza
User 2
Pork Ramen
Kirimachi
Pork Dumplings
Hakkasan
Pork Rice
User 3
User
Restaurant
Dish


User 1
Antica Roma
Mushroom Risotto
Mushroom Pizza
Il Casaro
Pepperoni Pizza
User 2
Pork Ramen
Kirimachi
Pork Dumplings
Hakkasan
Pork Rice
User 3
User
Restaurant
Dish



User 1
Antica Roma
Mushroom Risotto
Mushroom Pizza
Il Casaro
Pepperoni Pizza
User 2
Pork Ramen
Kirimachi
Pork Dumplings
Hakkasan
Pork Rice
User 3
6
10
2
7
1
User
Restaurant
Dish

Graph Learning in Uber Eats

Bipartite graph for dish recommendation

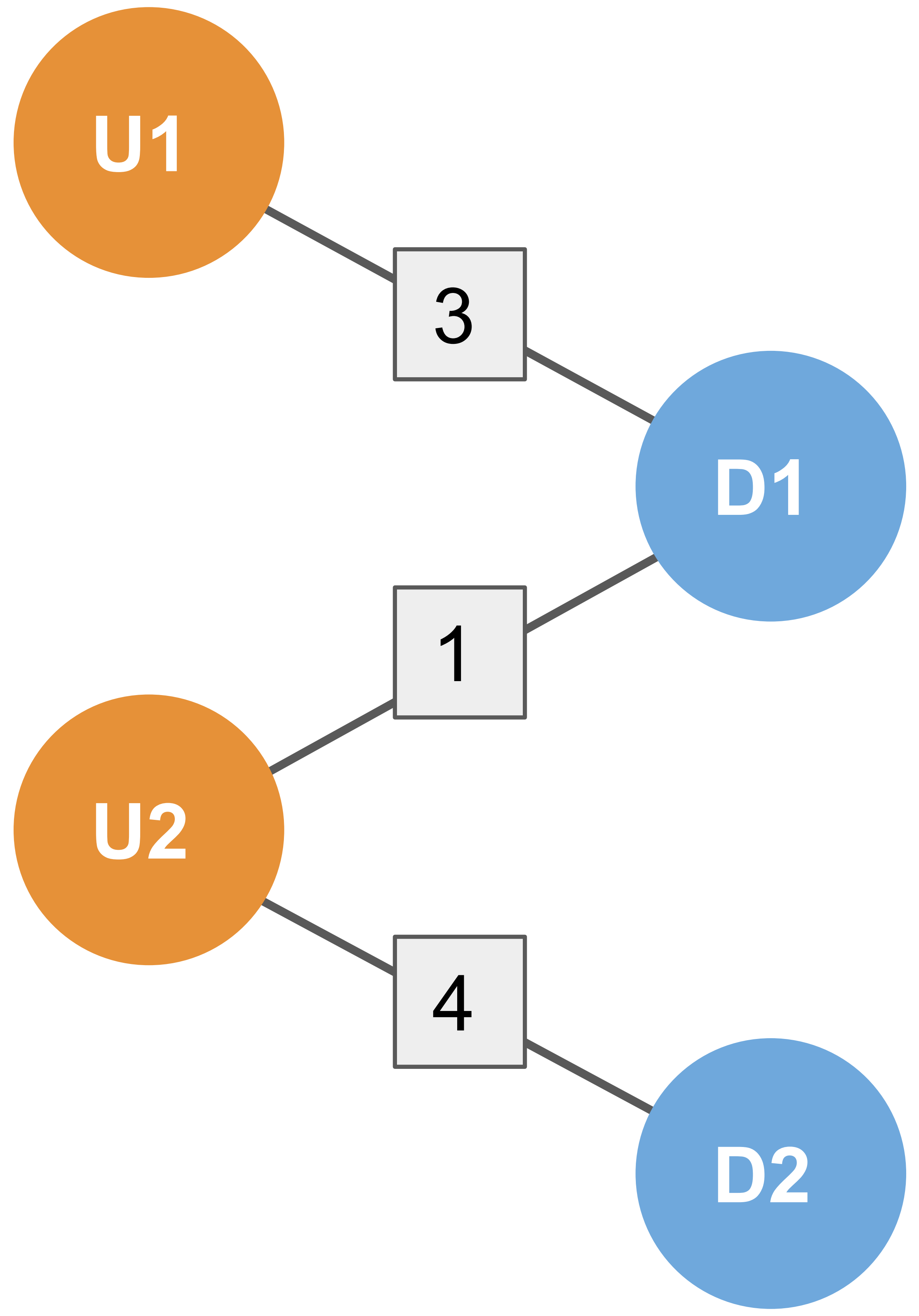
Users connected to dishes they have ordered in the last M days

Weights are frequency of orders

Graph properties

Graph is dynamic: new users and dishes are added every day

Each node has features, e.g. word2vec of dish names



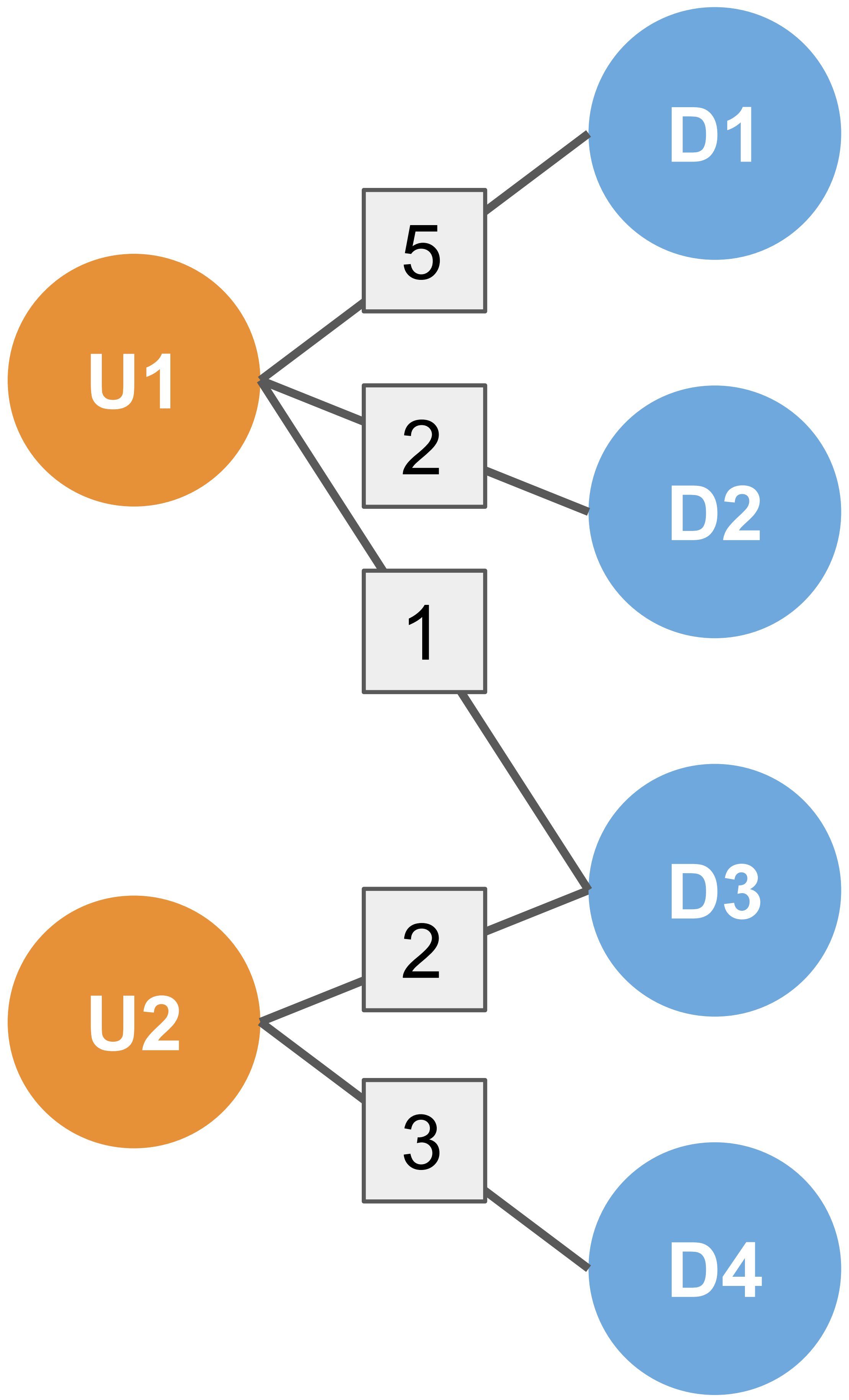
Max Margin Loss

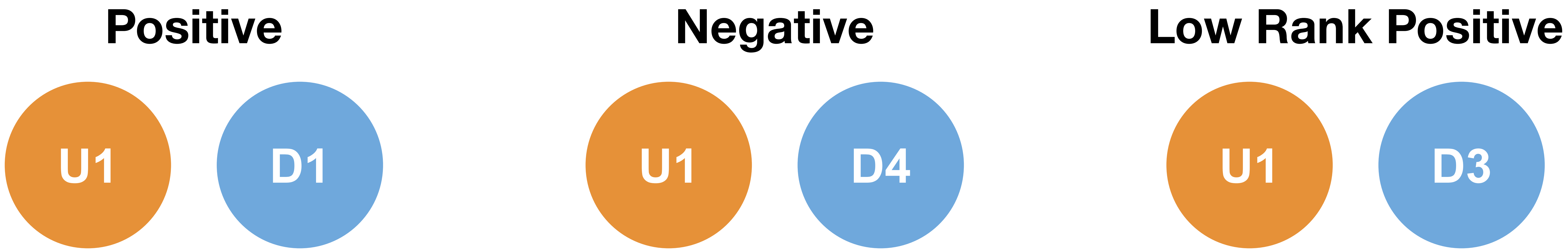
For dish recommendation we care about **ranking**, not actual similarity score

Max Margin Loss:

$$L = \sum_{(u,v)\in E} \max(0, -\underbrace{z_u z_v}_{\text{positive pair}} + \underbrace{z_u z_n}_{\text{negative sample}} + \underbrace{\Delta}_{\text{margin}})$$

New loss with Low Rank Positives





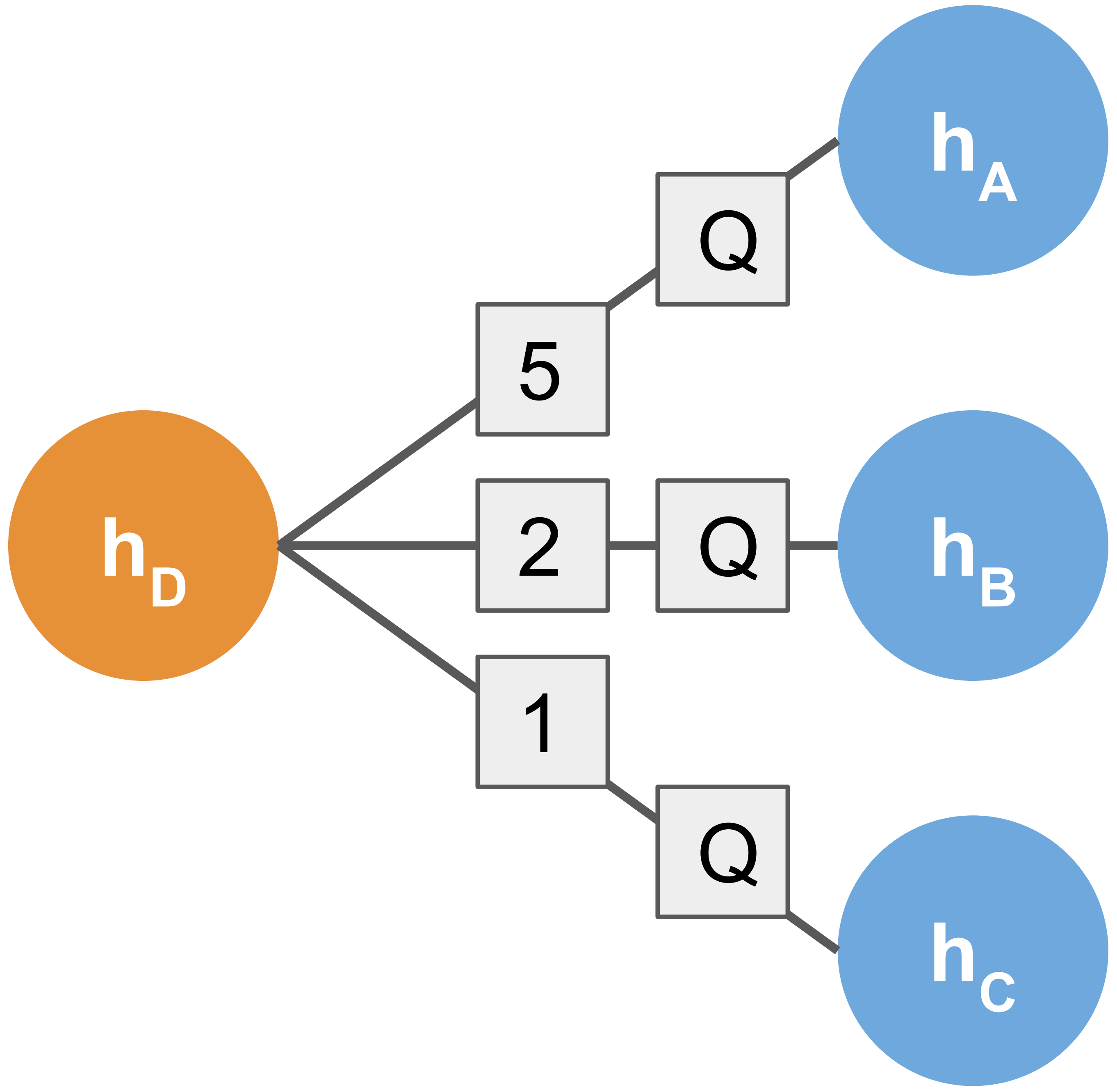
$$L = \sum_{(u,v)\in E} \alpha_n \max(0, -z_u z_v + z_u z_n + \Delta_n) +$$

$$\alpha_l \max(0, -z_u z_v + z_u z_l + \Delta_l)$$

$$\Delta_l < \Delta_n$$

Weighted pool aggregation

Aggregate neighborhood embeddings based on edge weight



$$\mathbf{AGG} = \sum_{u \in N(v)} w(u,v) Q h_u^{k-1}$$

Q denotes a fully connected layer

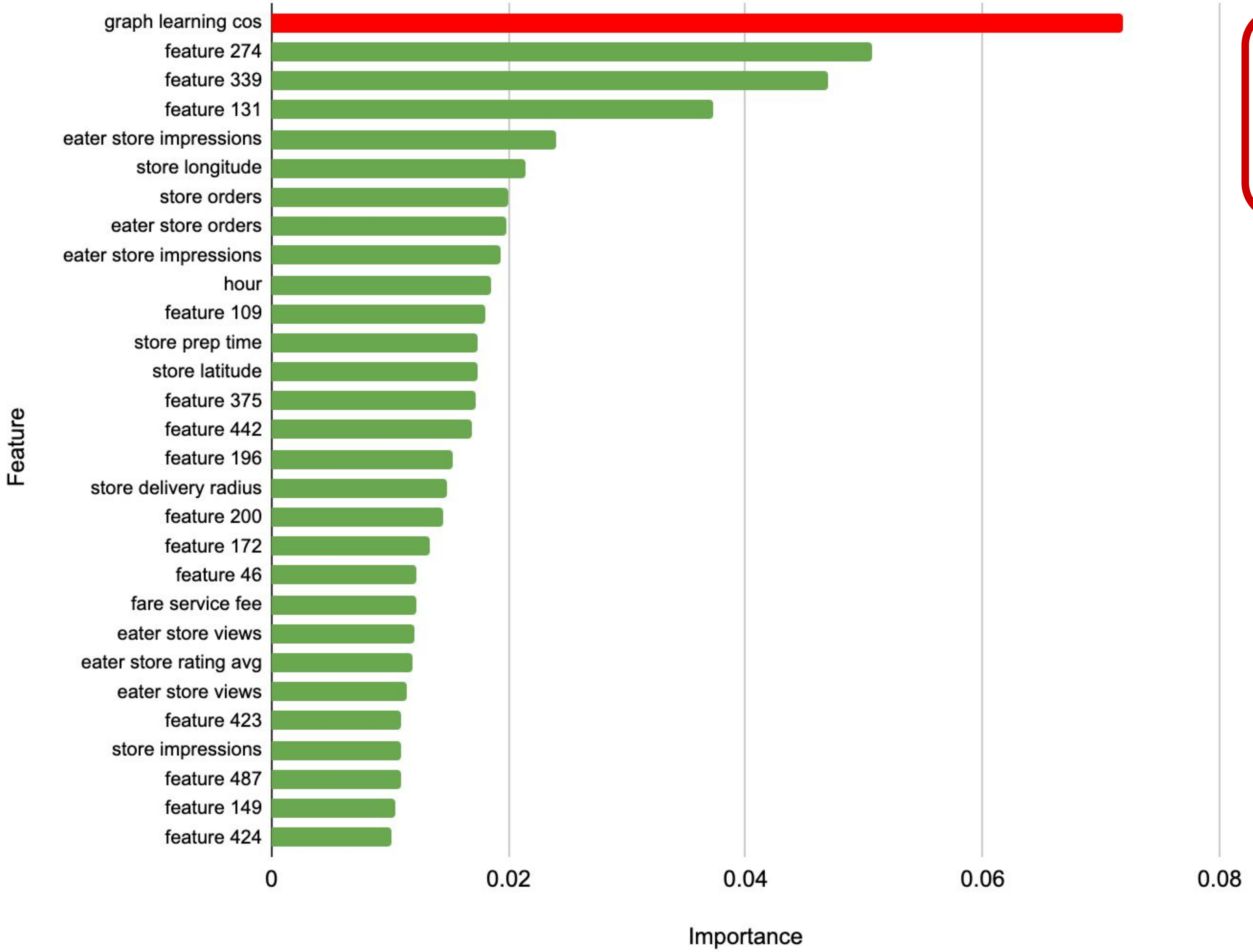
Offline evaluation

Trained the downstream Personalized Ranking Model using graph node embeddings

~12% improvement in test AUC over previous production model

Model	Test AUC
Previous production model	0.784
With graph embeddings	**0.877**

Feature Importance



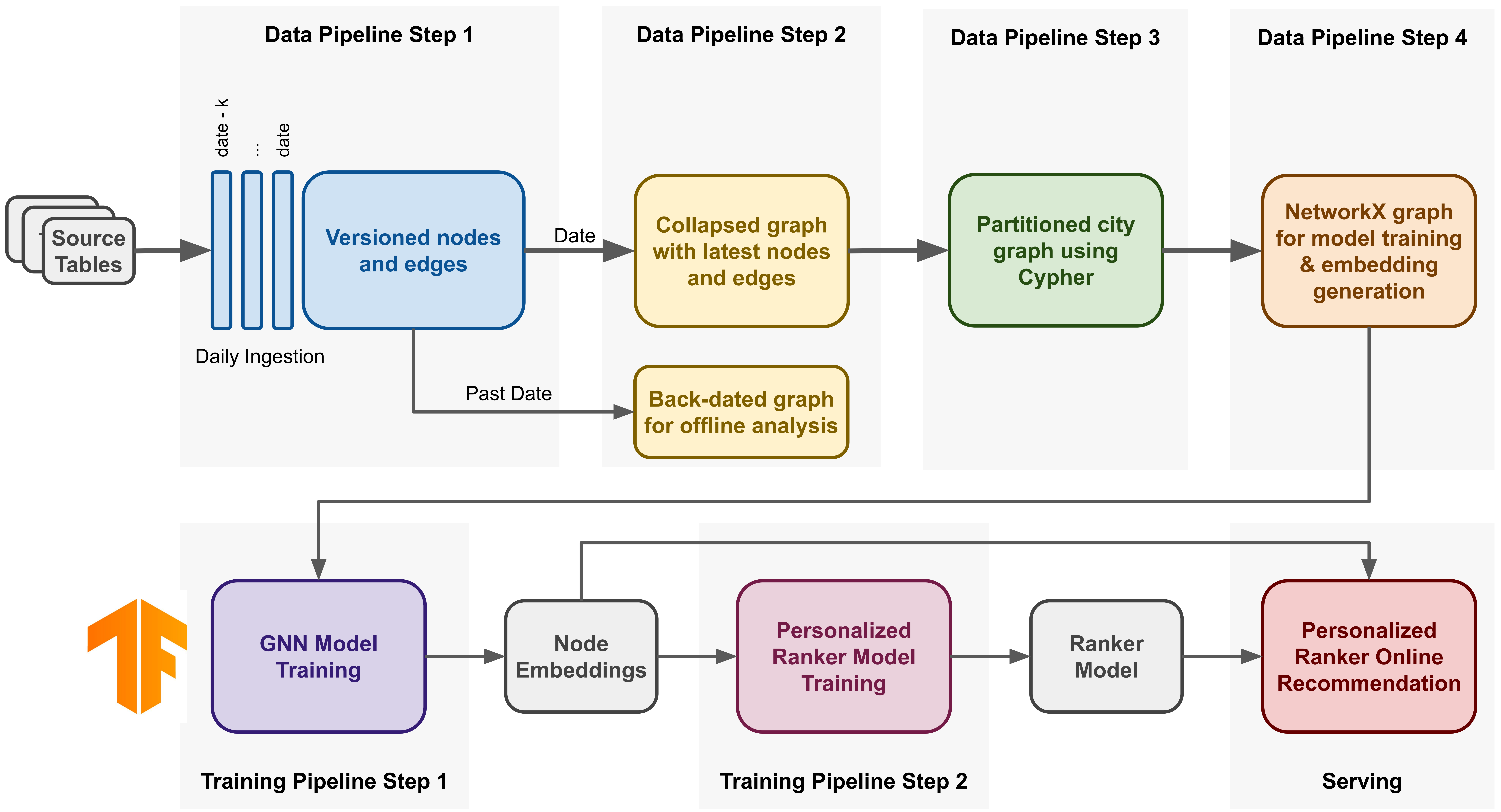
Graph learning cosine similarity is the top feature in the model

Online evaluation

Ran a A/B test of the Recommended Dishes Carousel in San Francisco

Significant uplift in Click-Through Rate with respect to the previous production model

Conclusion: Dish Recommendations with graph learning features are live in San Francisco, soon everywhere else



Data Pipeline Step 1
Data Pipeline Step 2
Data Pipeline Step 3
Data Pipeline Step 4
date - k
...
date
Source Tables
Versioned nodes and edges
Daily Ingestion
Date
Past Date
Collapsed graph with latest nodes and edges
Back-dated graph for offline analysis
Partitioned city graph using Cypher
NetworkX graph for model training & embedding generation
GNN Model Training
Node Embeddings
Personalized Ranker Model Training
Ranker Model
Personalized Ranker Online Recommendation
Training Pipeline Step 1
Training Pipeline Step 2
Serving

More Resources

Uber Eng Blog Post

Learn better representation in data scarcity regimes like small/new cities through meta-learning [NeurIPS Graph Representation Learning Workshop 2019]

Learnings

In complex data pipelines, **the model isn't always the bottleneck**

- Graph processing was more expensive than model inference because of sheer size

Even when the model (or the data proc + model) is the bottleneck **you can often precompute and cache**

- Precomputed a big LRU cache of user-to-dish/restaurant similarities. It was recomputed entirely only when the model was updated and refreshed after user ordered

Learnings: online evaluation issues

Q: Despite big offline gains, only got small improvement in Click through rate and orders (still statistically significant and worth millions of dollars), why?

A:

Learnings: online evaluation issues

Q: Despite big offline gains, only got small improvement in Click through rate and orders (still statistically significant and worth millions of dollars), why?

A: Our recommendations where a small part of the UI, "favourite restaurants" and "Daily Deals" came always first in the UI and gathered most of clicks and orders. Bewre how you choose the denominator of your metrics!

Learnings: online evaluation issues

Q: Why is it hard to show big online gains in recommender systems in general?

A:

Learnings: online evaluation issues

Q: Why is it hard to show big online gains in recommender systems in general?

A: If there's a model in production your are comparing against, you are likely using biased data for both training and prediction!

Learnings: data bias

The world changes (new restaurants and dishes) -> ML lifecycle is a loop

The user behavior changes (now that my favorite pizza place is on the app, I start always ordering from there)

Model deployment changes user behavior (the items the model suggest influence your behavior)

Biased training data and biased evaluation data

Learnings: data bias

Q: How to collect unbiased data?

A:

Learnings: data bias

Q: How to collect unbiased data?

A: Complicated, one option is to show random recommendations to x% of users

Learnings: data bias

Q: What is the cost of collecting unbiased data?

A:

Learnings: data bias

Q: What is the cost of collecting unbiased data?

A: The likelyhod of those users actually selecting those items is very low -> small positive data is collected, those users may not buy anything -> the company looses money!

Learnings: data bias

Q: What could be compromise solutions?

A:

Learnings: data bias

Q: What could be compromise solutions?

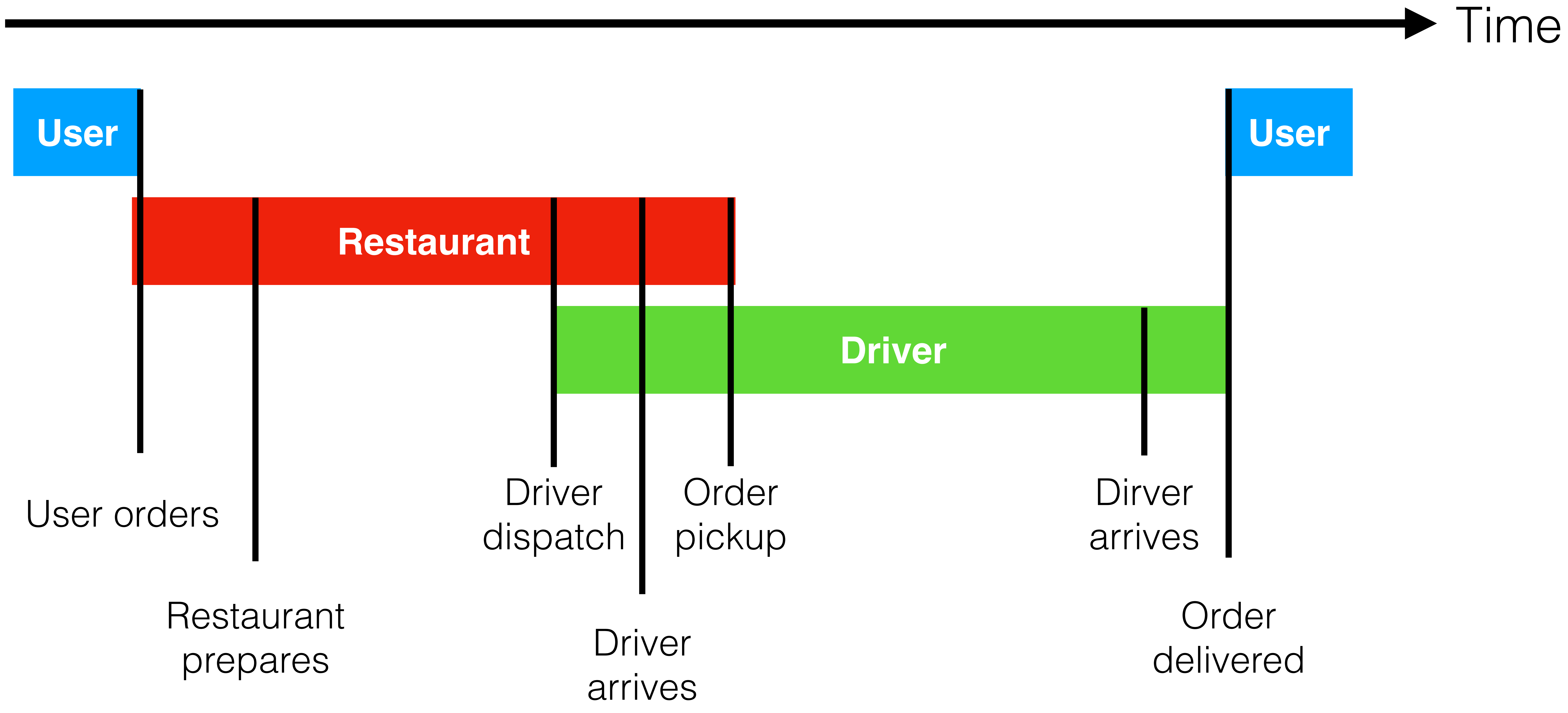
A: Show to users random predictions from within the top 100 predicted by the model. Data is still biased, but more likely to collect unexpected positive datapoints.

Team

Ankit Jain	Isaac Liu	Ankur Sarda
Piero Molino	Long Tao	Jimin Jia
Jan Pedersen	Nathan Berrebbi	Santosh Golecha
Ramit Hora	Alex Danilychev	

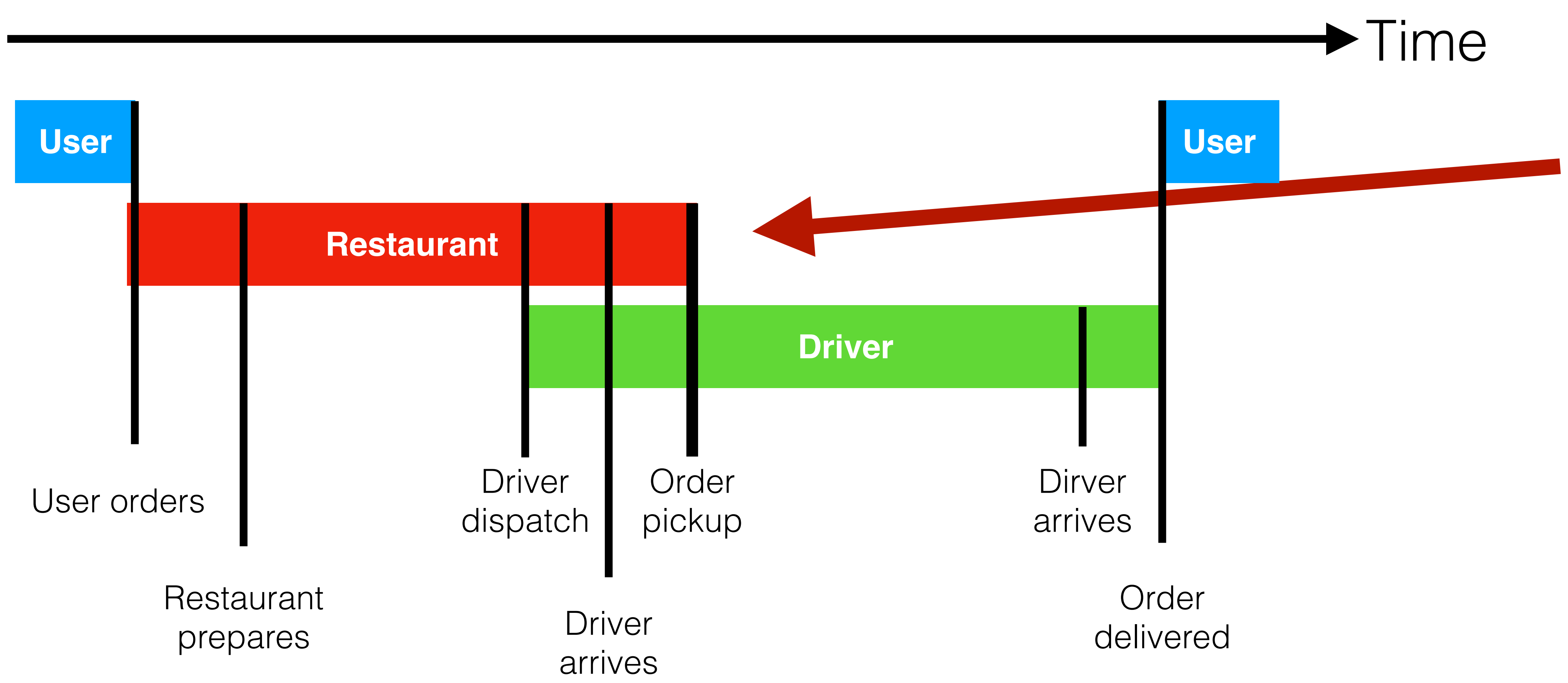
Restaurant preparation time

The data generation process



Restaurant preparation time

The data generation process



Predict **restaurant preparation time** is useful, I can decide when to dispatch the driver to reduce wait! (If I can also predict when the driver will arrive)

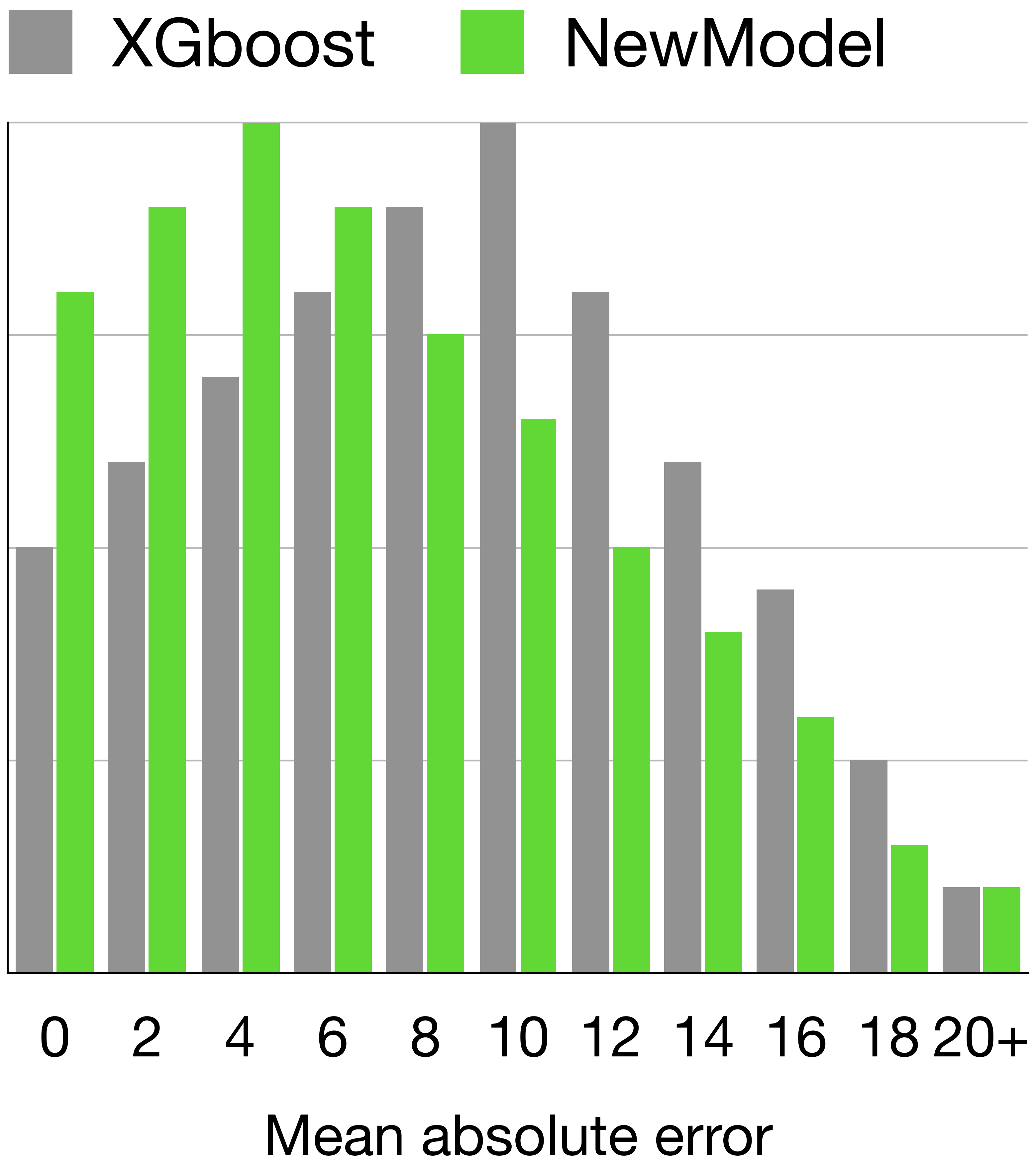
- How do you know when a restaurant is done preparing?
- The driver can arrive early, in which case the preparation time is from initial order to order pickup
- If the driver arrives late, and the dish is already prepared, the order pickup time is a upper bound

Restaurant preparation time modeling

We tried trainign a model anyway using order pickup

Huge variance in the training data -> H**uge variance** in predictions!

Our model was **5min more acurate** than previous one, but with stddev +- 10min!



Restaurant preparation time variance

Drilled into the data to **understand** the source of variance

Same restaurant, same day, same order, few minutes after -> **20min** prep time vs **2min** prep time

Why?

Restaurant	Order	Day	Time	Prep Time
POD Thai	Pho Soup	Tuesday 2nd	19:10	20m
POD Thai	Pho Soup	Tuesday 2nd	19:15	2m

Restaurant preparation time new feature

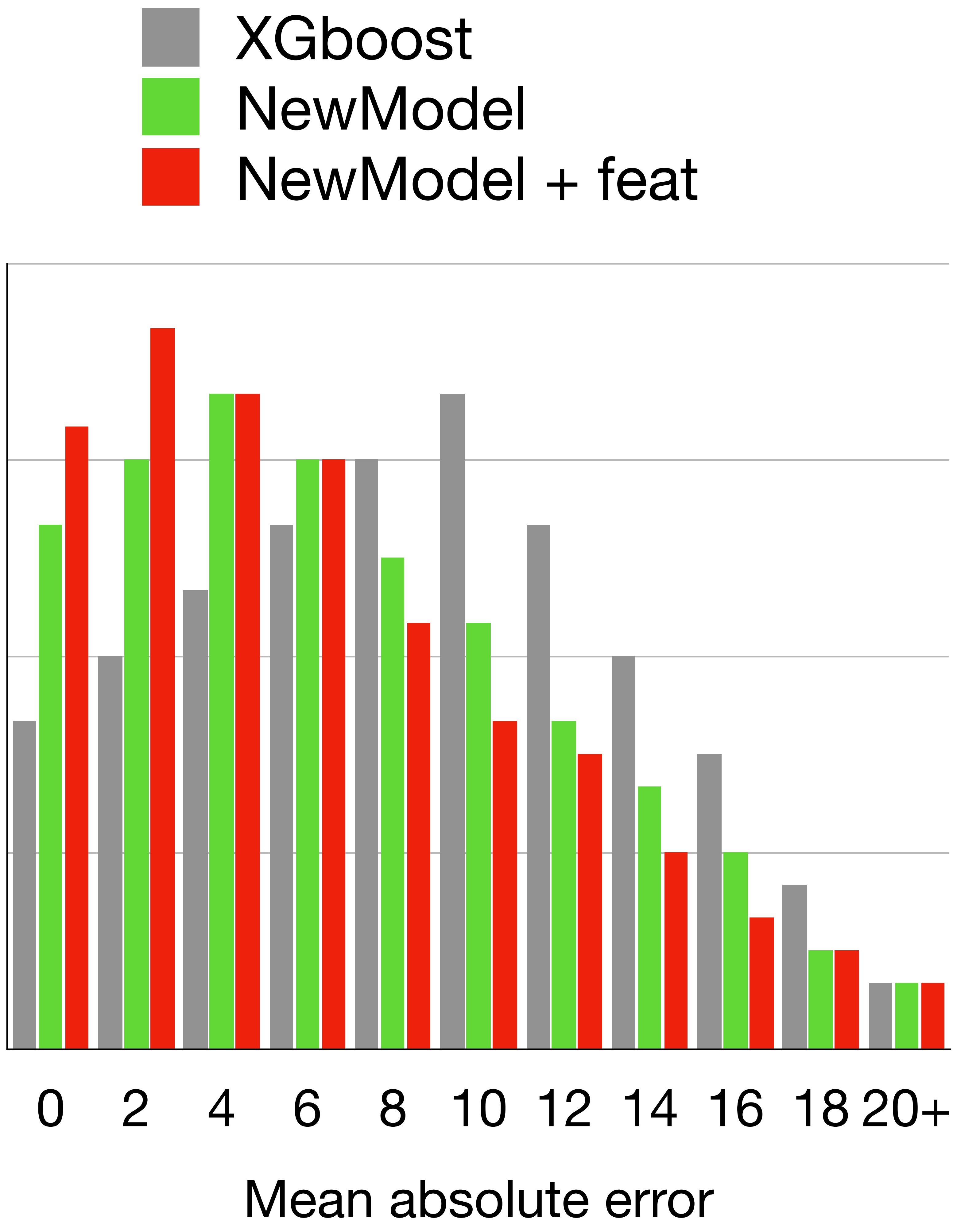
Restaurants batch orders!

Theory: They prepare a big amount of soup when first ordered, the next soup order will take much less because they are already prepared

Added a feature in the model:

> *were items in the order ordered in the last x minutes?*

Improved predictions by **2min**, reduced stddev by **1/3** (still a lot)



Restaurant preparation time moral

Went back to data collection, asked restaurants to notify us when the order was ready

Still noisy data (restaurants have no incentive to be precise, or they forget entirely), but better estimate

Moral: ML lifecycle is a loop and you can go back to the data collection process even after deployment, and iterate the process multiple times

What am I working on now

@Stanford with Chris Ré

Ludwig: declarative multimodal deep learning pipeline toolbox (no code needed, extensible, AutoML capabilities)

For a talk about Ludwig you can check my website http://w4nderlu.st or the last Stanford MLSys Seminar Series episode http://mlsys.stanford.edu

Founded a company to make ML accessible to less technical people: AutoML + end-to-end platform built on Ludwig + secret spicy sauce!