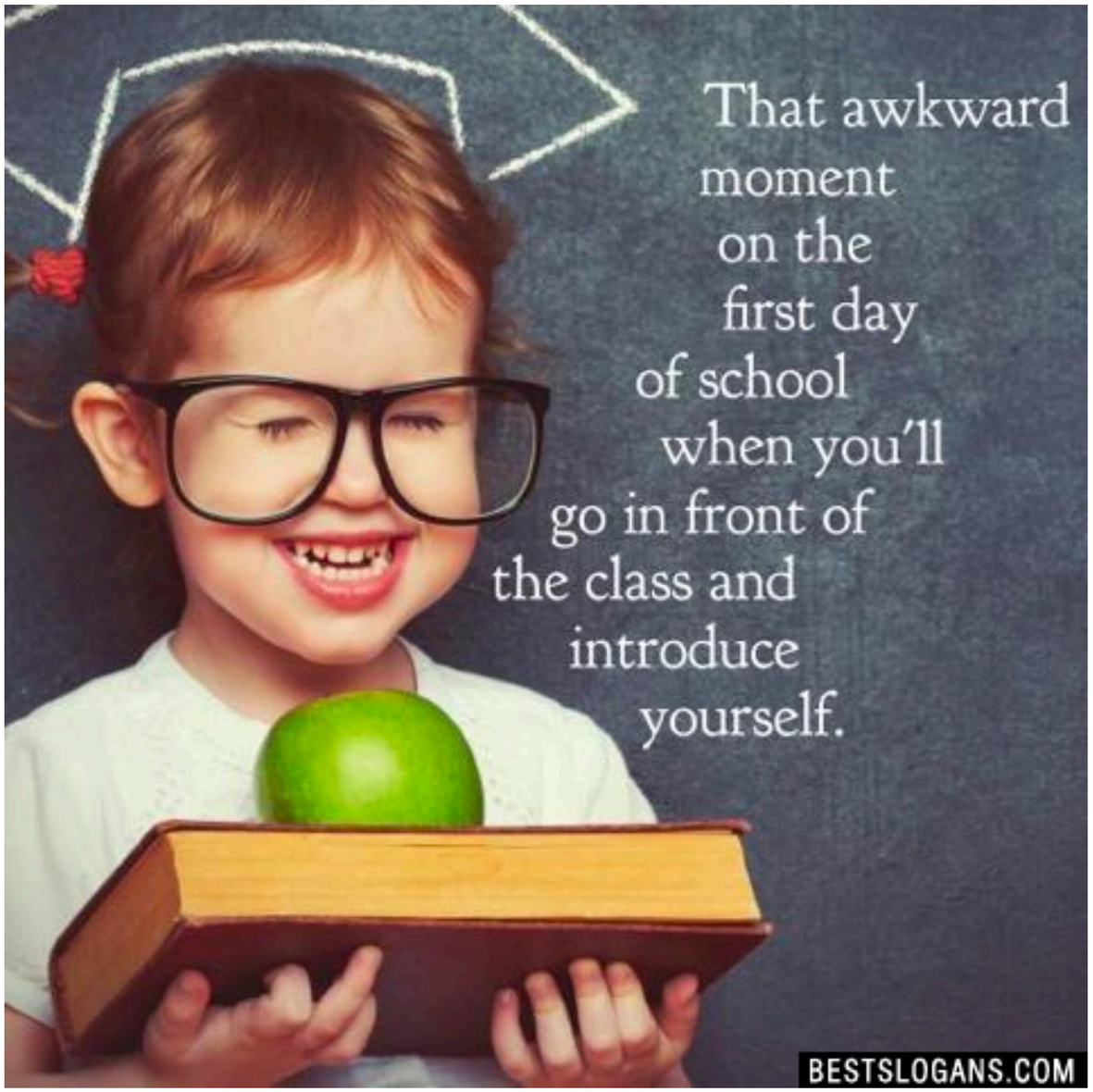
Introduction to Machine Learning Systems

Pooyan Jamshidi
USC


That awkward
moment
on the
first day
of school
when you'll
go in front of
the class and
introduce
yourself.
BESTSLOGANS.COM

First day, ha?

Target Audience

- First, it serves students who are interested in machine learning but **haven't built real-world machine learning systems**.

- The course is different from other ML courses you may have picked up. In this course, we will study techniques applicable to building whole **production-ready systems**, not just naive scripts.

- We'll explore the entire range of possible components you might need to implement in a ML system, with lots of **common design pitfalls**.

- Along the way, you'll learn about the various functions of a machine learning system, in the context of implementing systems that fulfill those needs.

Target Audience

- Second, this course serves **students who are interested in the bigger picture** of machine learning systems.

- I presume they know the concepts of ML but may only have implemented simple machine learning functionality (for example, scripts over files on a laptop).

- For such students, the course may introduce you to a range of concerns that you've never before considered part of the workflow of machine learning.

- I'll introduce components of ML systems that are often **neglected in academic machine learning discussions**.

Learning goals

- Understand how to **build** a system that can put the power of machine learning to use.

- Understand how to incorporate ML-based components into a **larger system**.

- Understand the **principles** that govern these systems, both as software and as predictive systems.

By the end of this class, I hope:

- You will be able to apply state of the art ML algorithms, in whatever problem you are interested in, at **scale** and learn how to deal with **unique challenges** that only may happen when building real-world production-ready AI/ML systems.

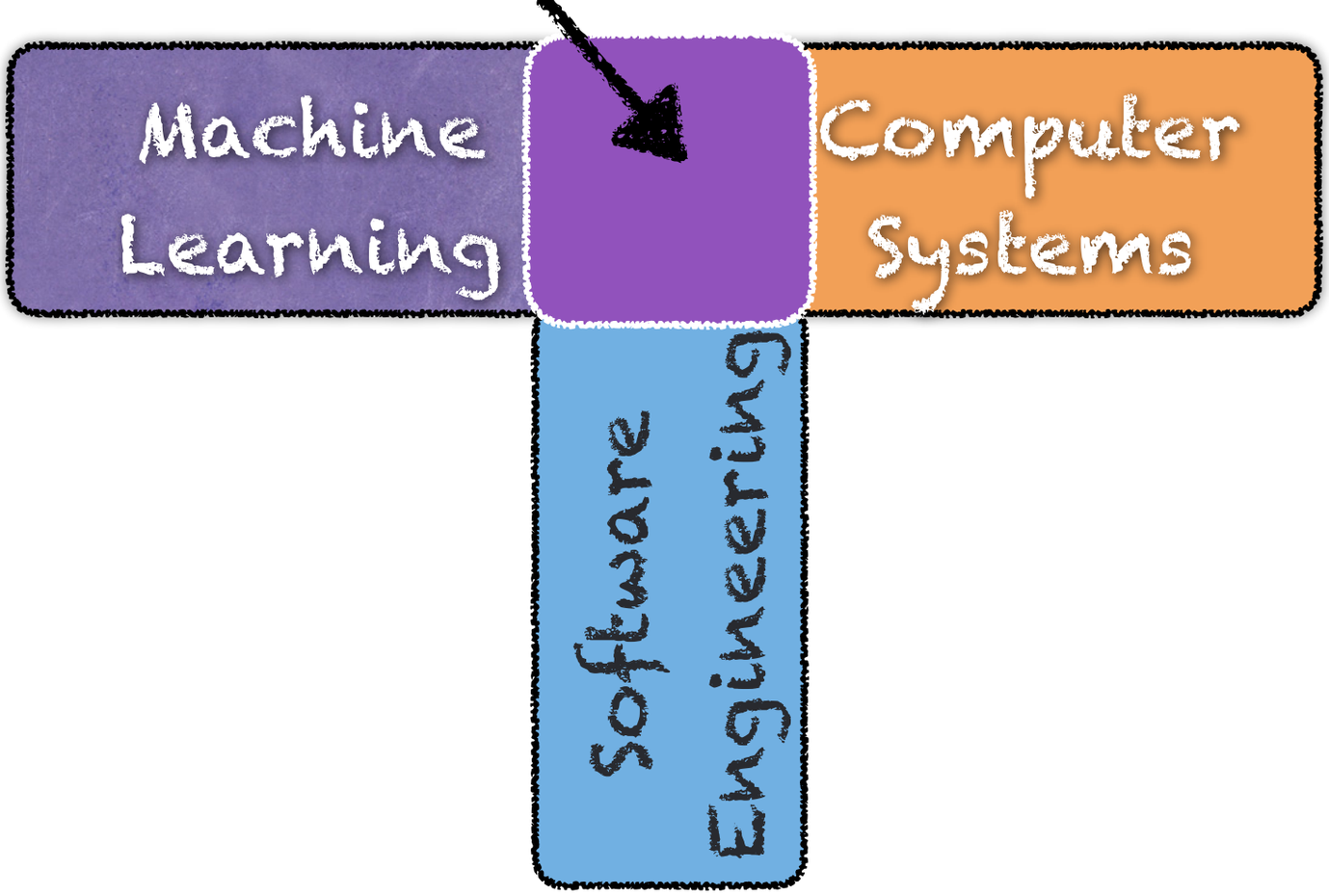
- You will be able to do **AI at the Edge**; there will be projects for an end-to-end, cloud-to-edge, hardware + software infrastructure for facilitating the deployment of AI-based solutions using Edge TPU/ NVIDIA Jetson Nano, TensorFlow Lite and similar technologies.

- I also hope I can convey my own excitement about AI/ML systems to you.

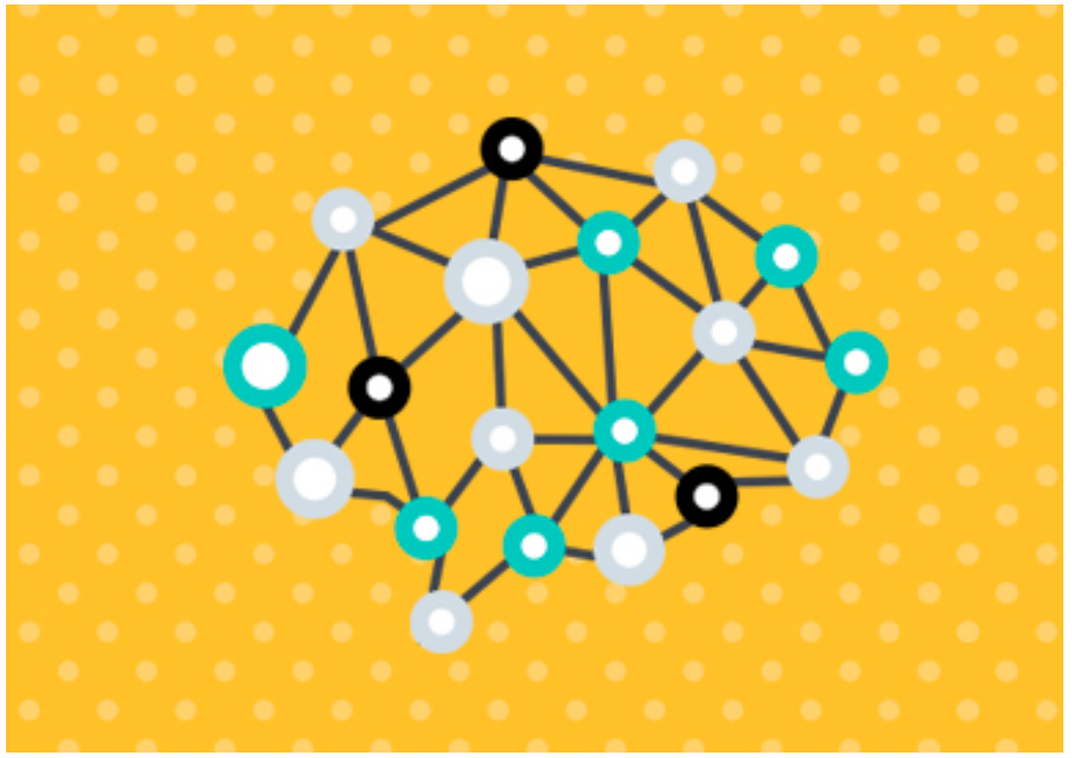
- You are well qualified for doing **research** in AI/ML systems research.

Learning Please interrupt and ask your questions at anytime!

ML Systems

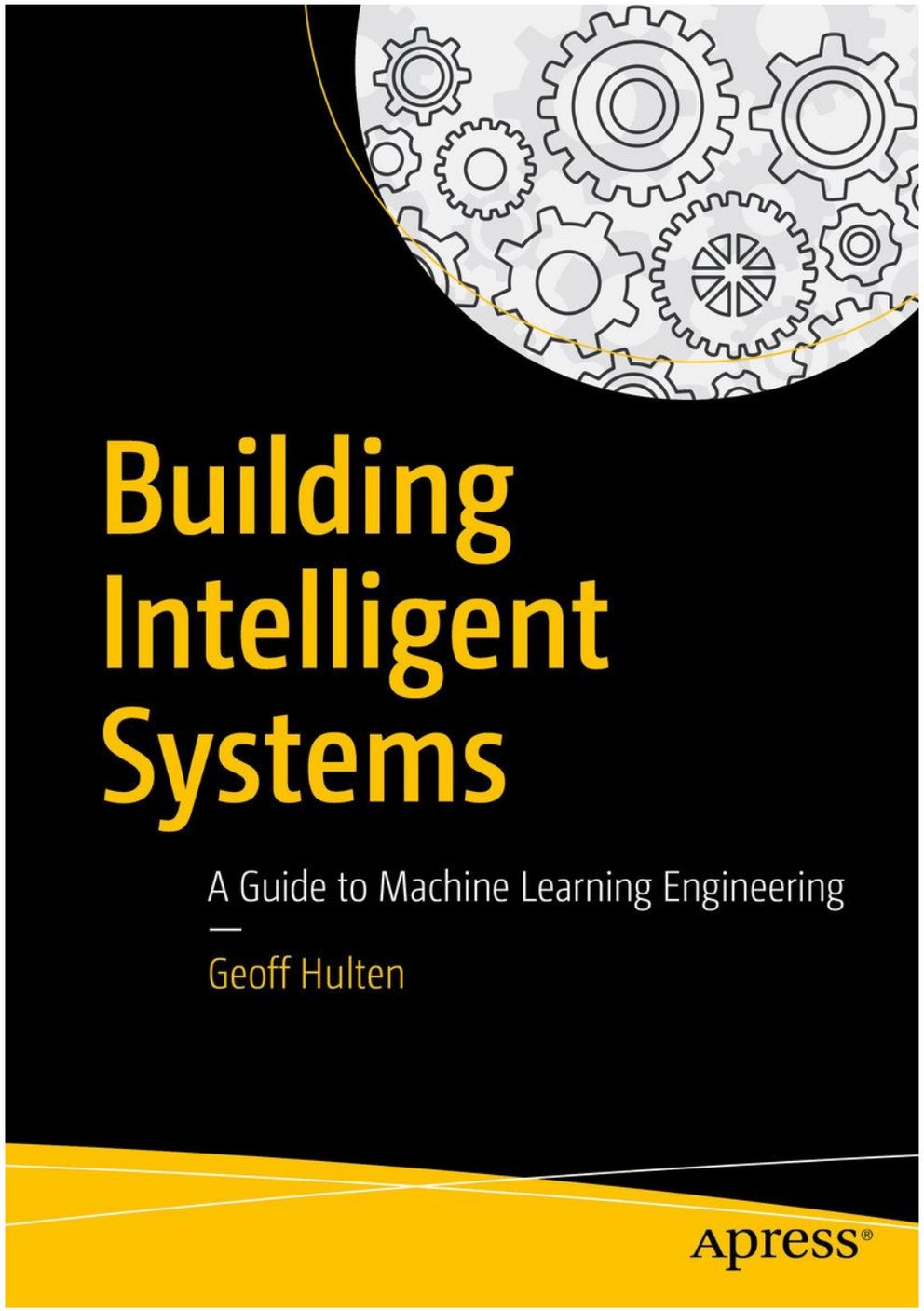


In this course, we study real-world challenges of adopting ML and solutions that scale



Main Sources

Not required!



UBER Engineering

THE NETFLIX TECH BLOG

Spotify Labs

NVIDIA AI

facebook artificial intelligence research

intel AI

Grading (undergraduate)

- 10% Participation
- 60% Course Project (Code+Short Report)
- 30% Homework/Quizzes/Assignments/Exams

Grading (graduate)

- 10% Participation
- 40% Course Project (Code+Short Report)
- 20% Short Paper (SysML workshops at ICLR/ICML)
- 30% Homework/Quizzes/Assignments/Exams

Grading

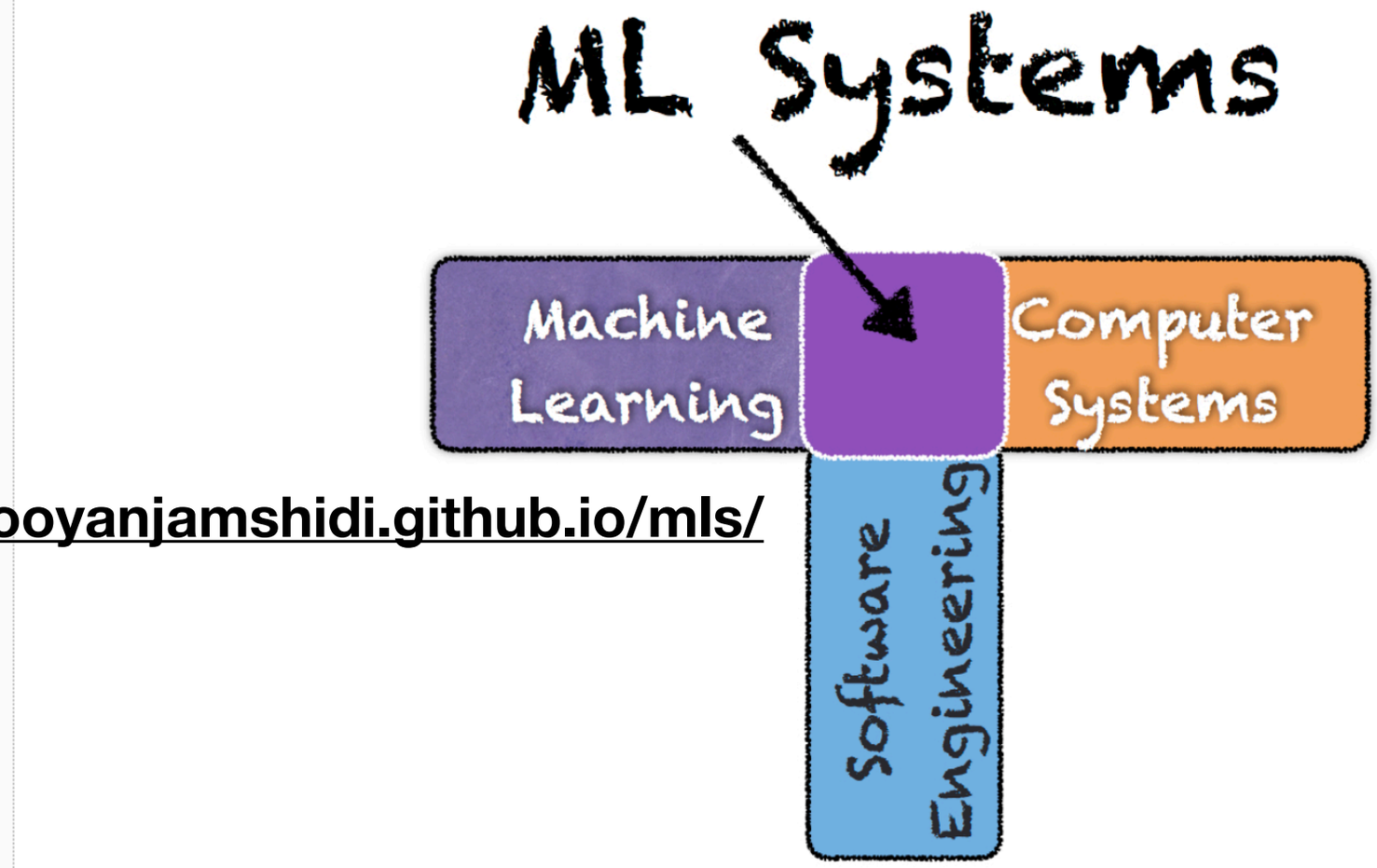
- A [90 – 100]
- B+ [86 – 90)
- B [75 – 86)
- C+ [70 – 75)
- C [60 – 70)
- D+ [55 – 60)
- D [40 – 55)
- F [0 – 40)

Office hours

- TR 13:00 pm – 14:30 pm

- That may change, please checkout the course website

Machine Learning Systems



https://pooyanjamshidi.github.io/mls/

New to machine learning? Not sure how ML works in production? You're welcome to follow and learn from this graduate-level course.

When we talk about Artificial Intelligence (AI) in general and Machine Learning (ML) in particular, we typically refer to a technique or an algorithm that gives the computer systems the ability to learn and to reason with data. However, there is a lot more to AI/ML than just implementing an algorithm or a technique. In this course, we will learn the fundamental differences between AI/ML as a technique versus AI/ML as a system in production. **By the end of this class, I hope**:

ML/AI Systems
pjamshid@cse.sc.edu

Office hours: Friday 1-3pm
Computer Science and Engineering, RM 2207
550 Assembly Street

pooyanjamshidi
pooyanjamshidi
pooyanjamshidi.github.io

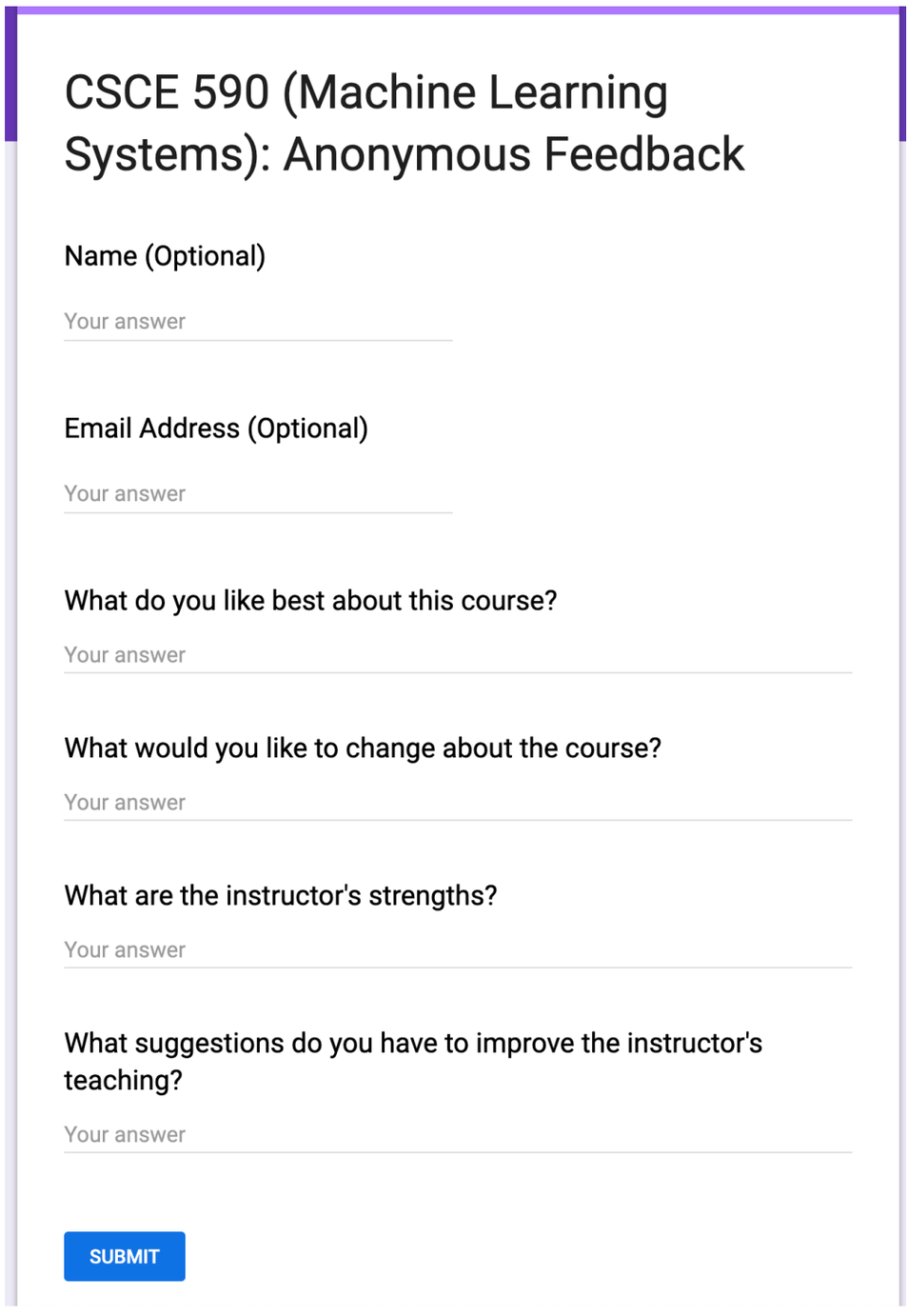
Discussions

- Piazza: https://piazza.com/sc/fall2019/csce590
- Ask questions
- Answer others’ questions
- Learn from others’ questions and answers
- Find teammates

Course Information: Feedback

- Please give feedback (positive or negative) as often as and as early as you can.

Link: tiny.cc/s2tkbz



CSCE 590 (Machine Learning Systems): Anonymous Feedback

Name (Optional)

Your answer

Email Address (Optional)

Your answer

What do you like best about this course?

Your answer

What would you like to change about the course?

Your answer

What are the instructor's strengths?

Your answer

What suggestions do you have to improve the instructor's teaching?

Your answer

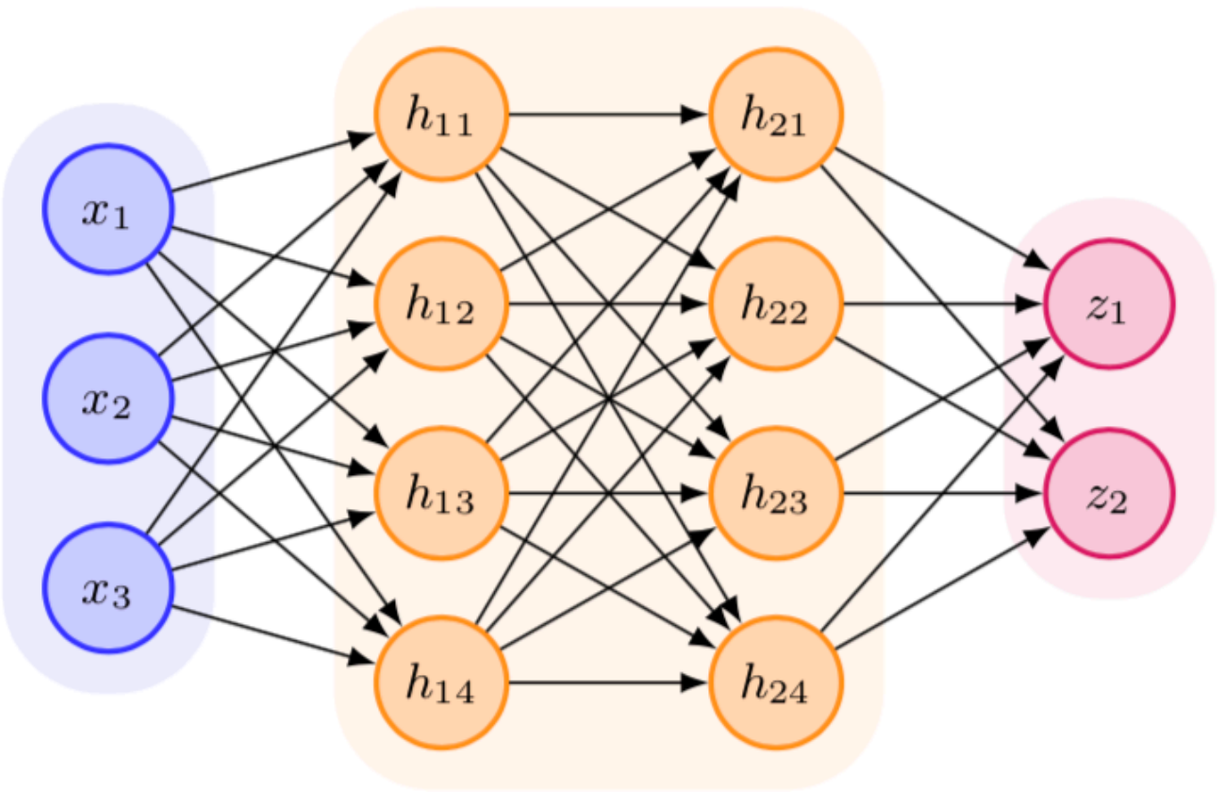
SUBMIT

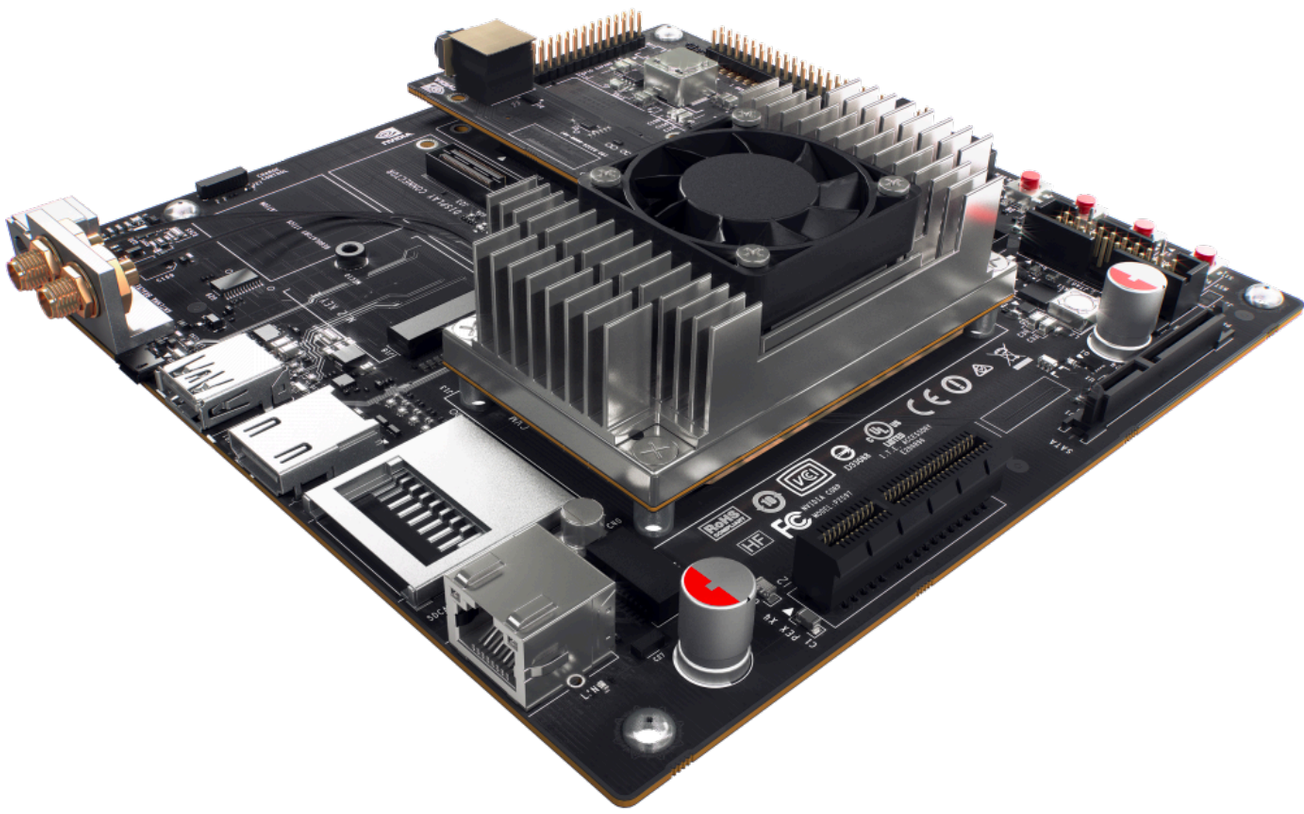
Project Proposal

- What is the **problem** that you will be investigating? Why is it interesting?
- What **reading** will you examine to provide context and background?
- What **data** will you use? If you are collecting new data, how will you do it?
- What **method** or algorithm are you proposing? If there are existing implementations, will you use them and how? How do you plan to improve or modify such implementations? You don't have to have an exact answer at this point, but you should have a general sense of how you will approach the problem you are working on.
- How will you **evaluate** your results? Qualitatively, what kind of results do you expect (e.g. plots or figures)? Quantitatively, what kind of analysis will you use to evaluate and/or compare your results (e.g. what performance metrics or statistical tests)?

Example

- **Project (DNN inference optimization)**: How the choice of DVFS (e.g., CPU frequency, enable/disable CPU cores) affect inference time and energy consummations of Deep Neural Networks?





How projects will be evaluated

- You can work in teams of up to 2 or 3 people.
- No communications between the two teams
- Every teammate should be able to demonstrate her/his contribution
- The outcome will be evaluated based on the quality of the results, report, and final presentation.
- The final report is an iPython notebook that has documentation, results, comparisons, discussions, and related work.

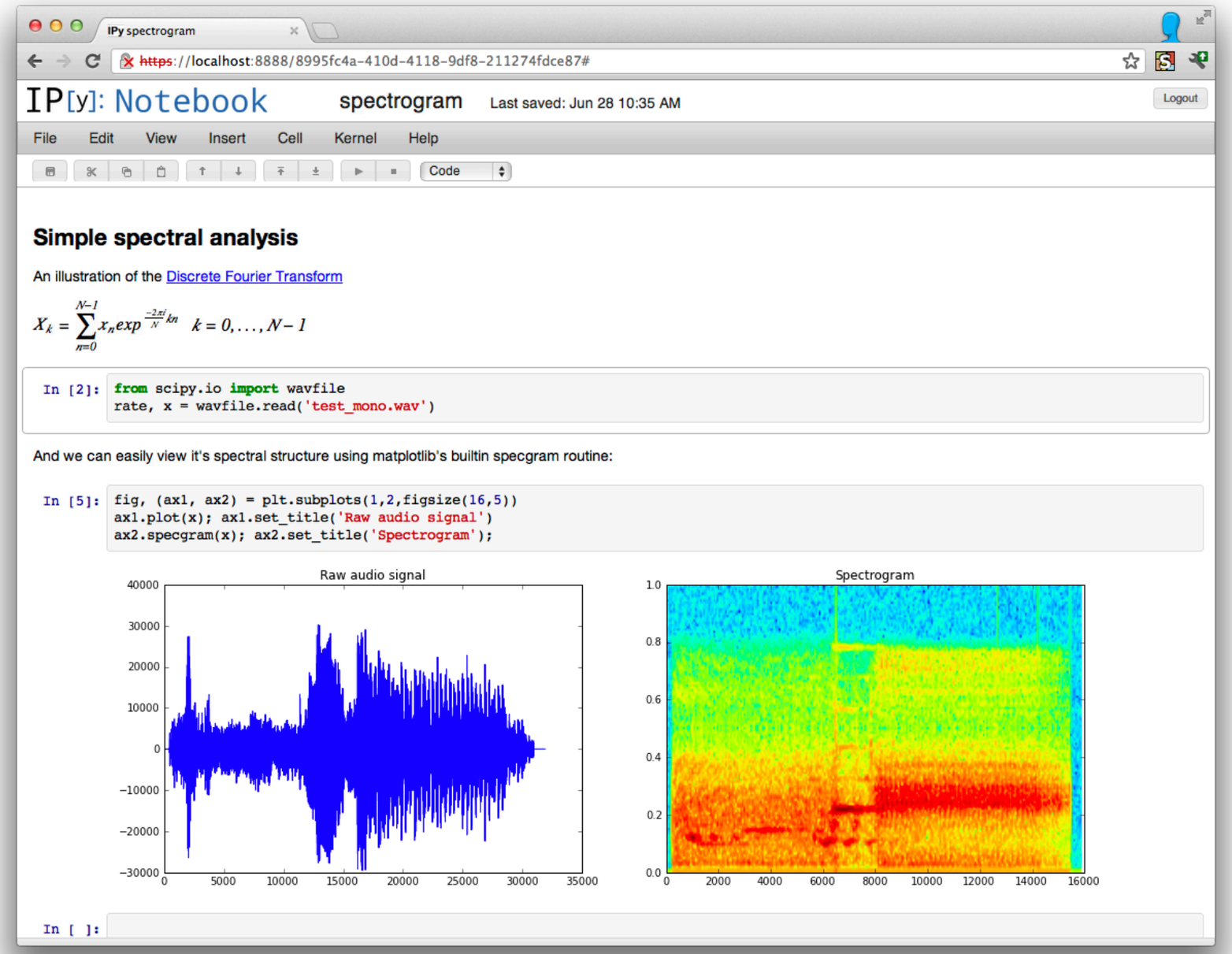
Honor Code

- You may consult any papers, books, online references, or publicly available implementations for ideas and code that you may want to incorporate into your strategy or algorithm, so long as you clearly cite your sources in your code and your writeup. However, under no circumstances may you look at another group's code or incorporate their code into your project.

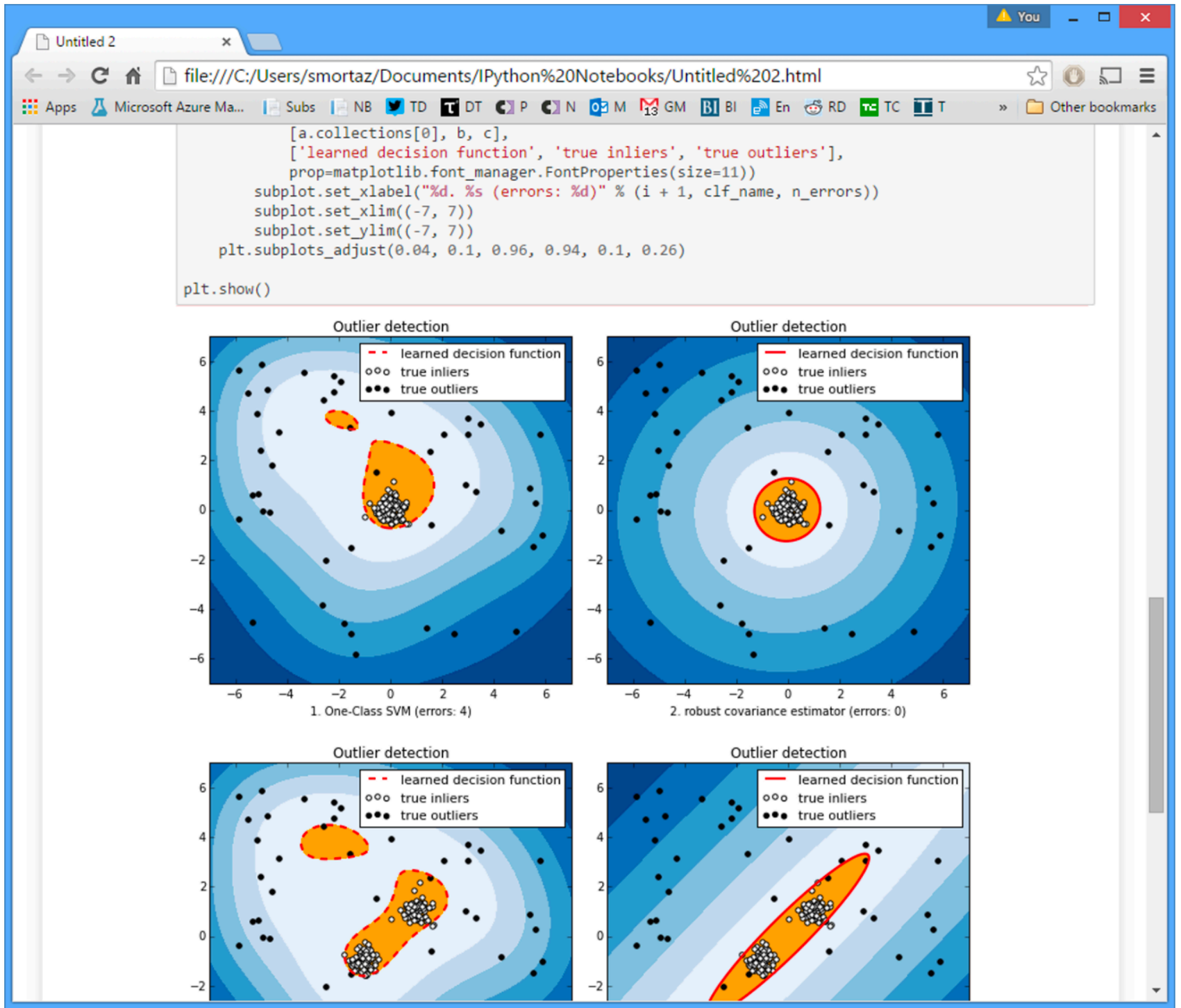
Important Dates

- Project proposal: due Thursday, September 12.
- Project milestone: due October 17.
- Final report: due December 3.
- Poster PDF: November 28

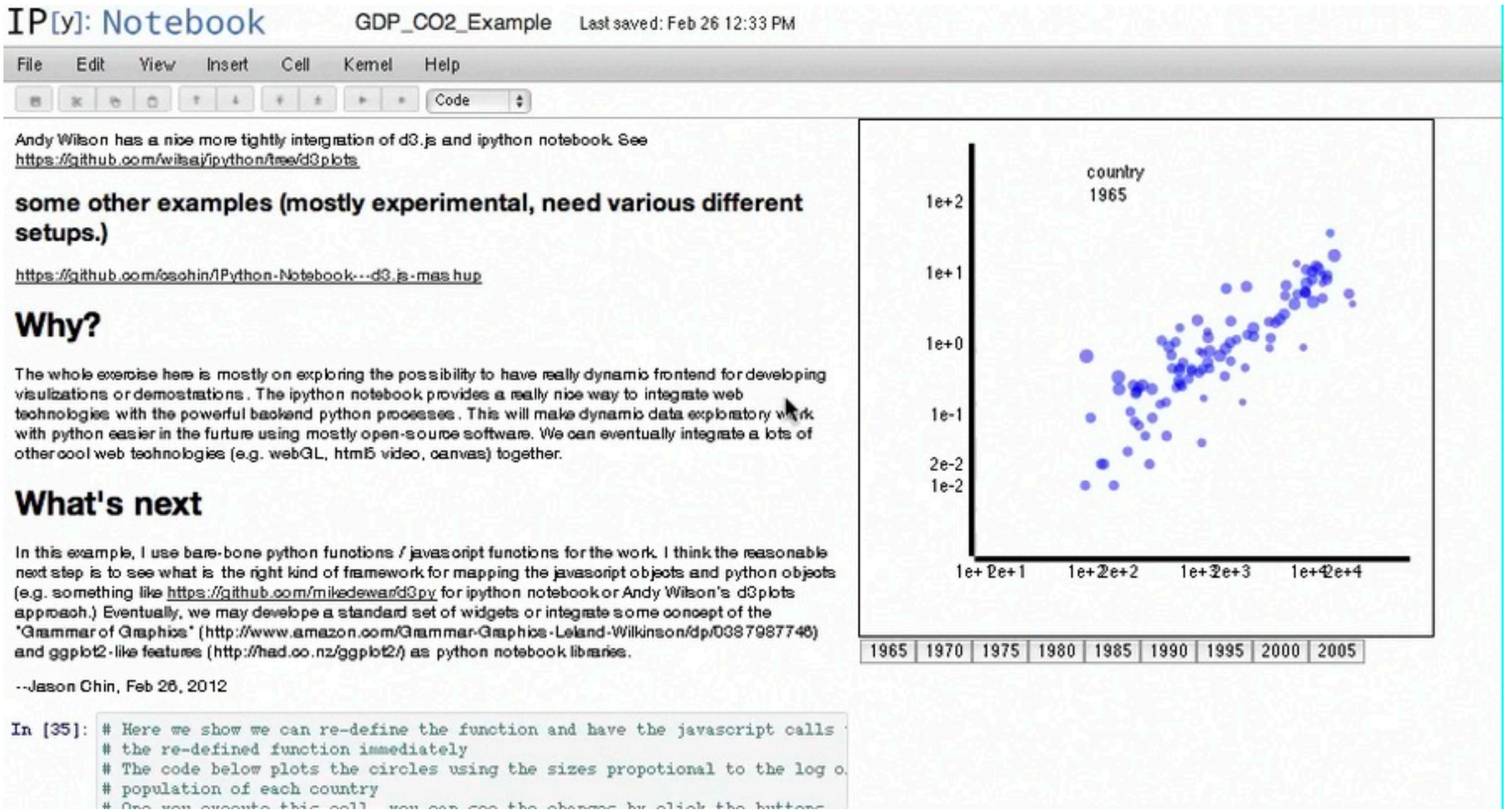
How the project report should looks like?



How the project report should looks like?



How the project report should looks like?



Design think it a lil

- Have each member of your team flesh out 20 quick ideas down on paper before meeting. Don’t be afraid to get creative

- Filter out list by doing quick Google searches on data a. Anything below GB scale of data...good luck. Vision = big datasets b. If you have an idea, Google it first! Don’t want to “just” reproduce the same result. There’s probably a Github with your project already

- Pay attention to how long and much data the models you see are trained on

- Find pattern in data+architecture combos

- Ask are there little tweaks or other experiments that haven’t been done yet?

- Can you extend the idea in one paper with another?

- Which idea gives you more things to experiment with? 8. How can you get pretty images / figures?

Try to avoid

- Nothing special in data pipeline. Uses prepackaged source Team starts late. Just instance and draft of code up by milestone

- Explore 3 architectures with code that already exists a. One RES-net, then a VGG, and then some slightly different thing

- Only ran models until they got ~65% accuracy 5. Didn’t hyperparameter search much

- A few standard graphs: loss curves, accuracy chart, simple architecture graphic

- Conclusion doesn’t have much to say about the task besides that it didn’t work

Aim for this

- Workflow set-up configured ASAP
- Have running code and have baseline model running and fully-trained
- Creative hypothesis is being tested
- Mixing knowledge from different aspects in DL
- Have a meaningful graphic (pretty or info rich)
- Conclusion and Results teach me something
- ++interactive demo
- ++novel / impressive engineering feat
- ++good results

Milestone Goals

- We want to see you have code up and running
- Data source explained correctly a. Give the true train/test/val split b. Number training examples c. Where you got the data
- What Github repo, or other code you're basing off of
- Ran baseline model have results a. Points off for no model running, no results
- Data pipeline should be in place
- Brief discussion of initial, preliminary results
- Reasonable literature review (3+ sources)
- 1-2 page progress report. Not super formal

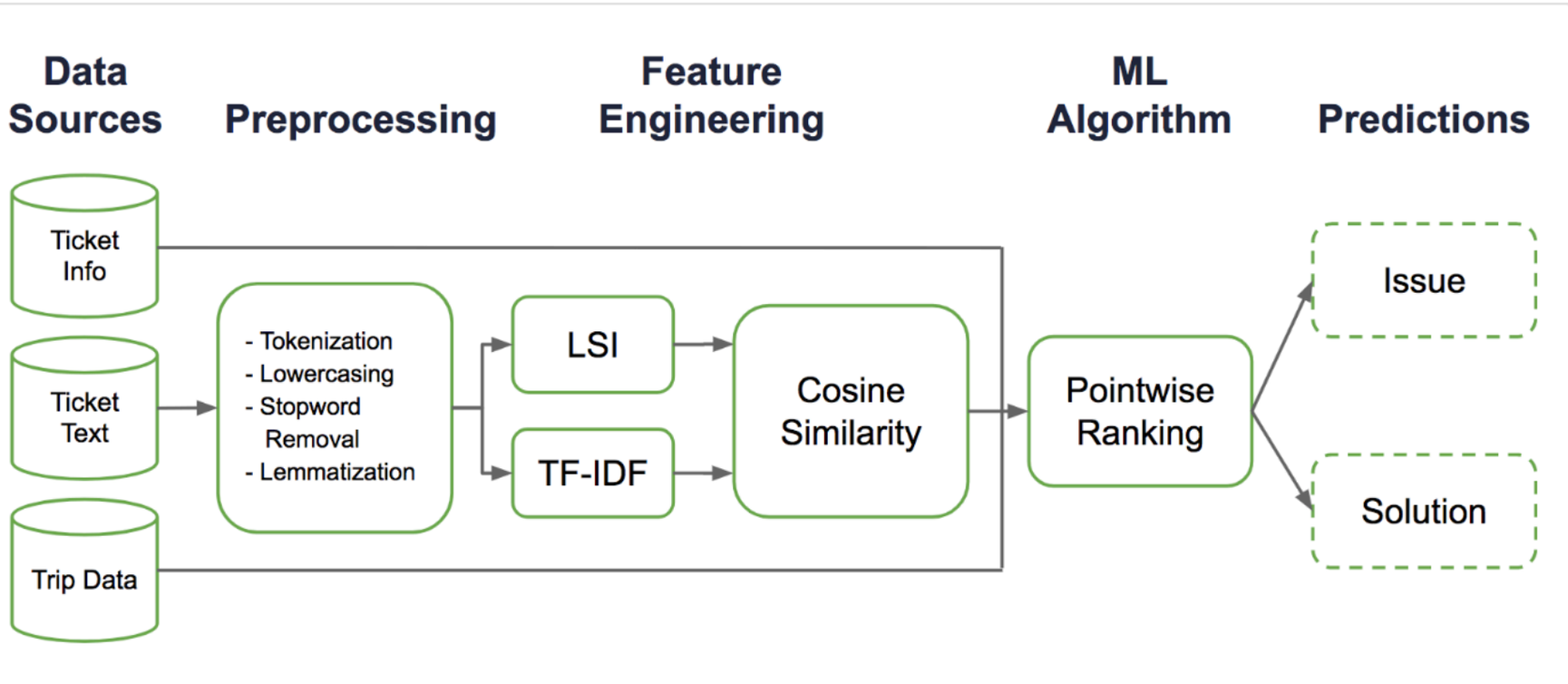
How each lecture looks like?

- We study ML systems in real world
- We discuss challenges for ML systems in real world
- We review solutions thats scale

COTA: Improving Uber Customer Care with NLP & Machine Learning

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

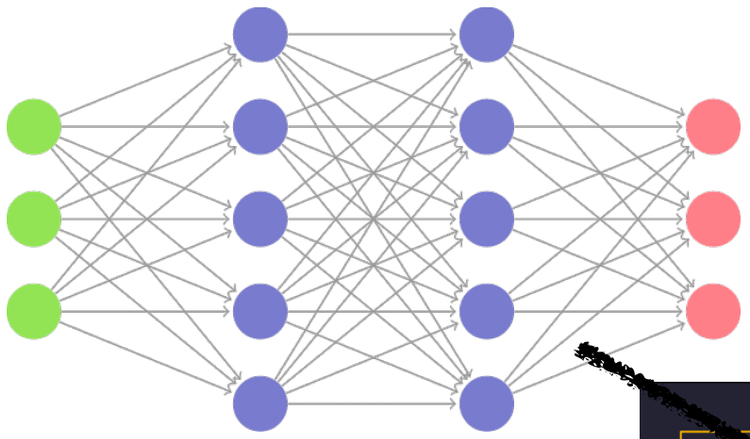
January 3, 2018

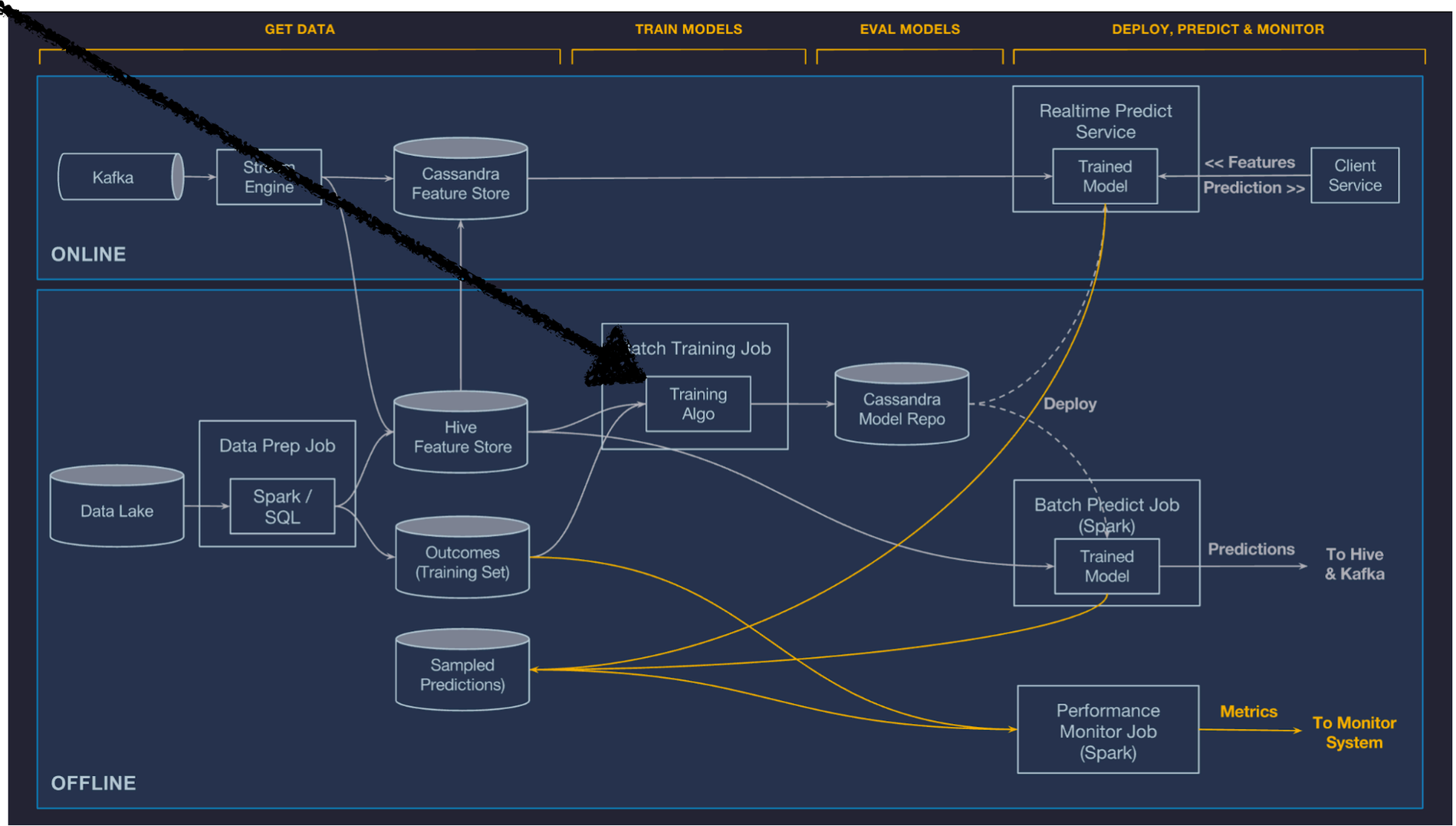


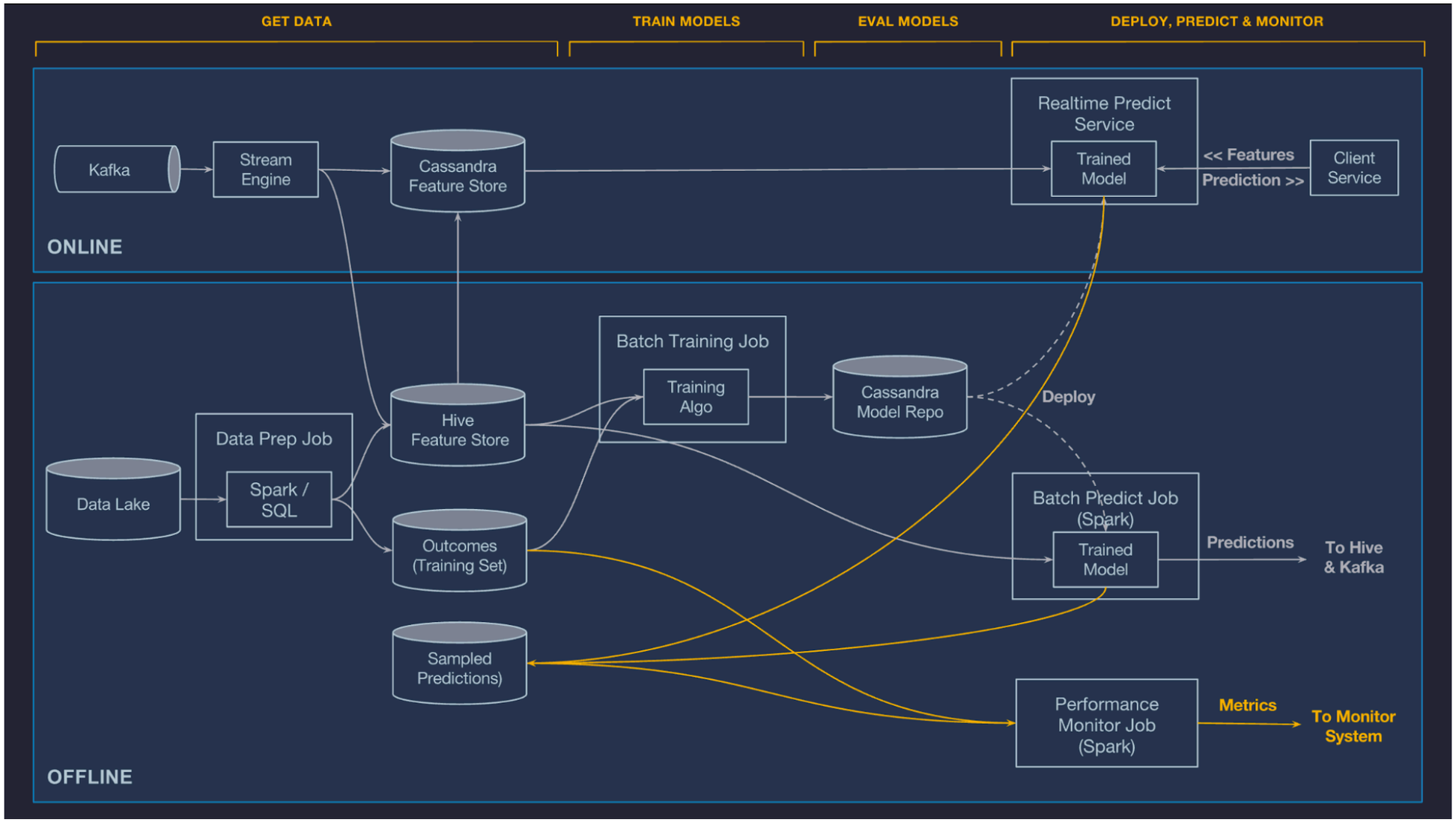
Outline

- Motivations behind creating an ML system
- Outline its backend architecture
- Showcase how such system has led to increased customer satisfaction in Uber

What is an ML system?





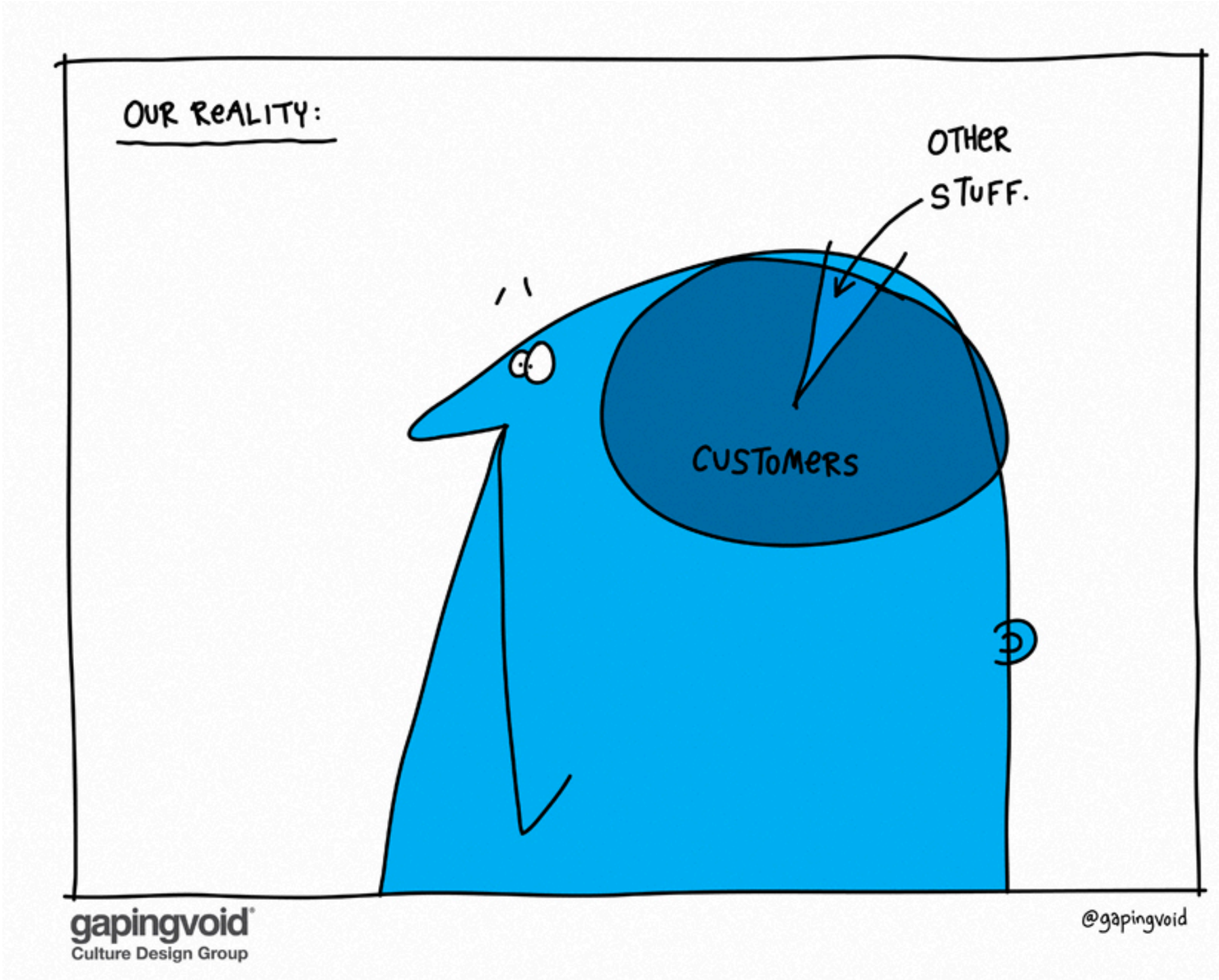


Machine learning systems comprise software/hardware components capable of learning from data and making predictions about the future.

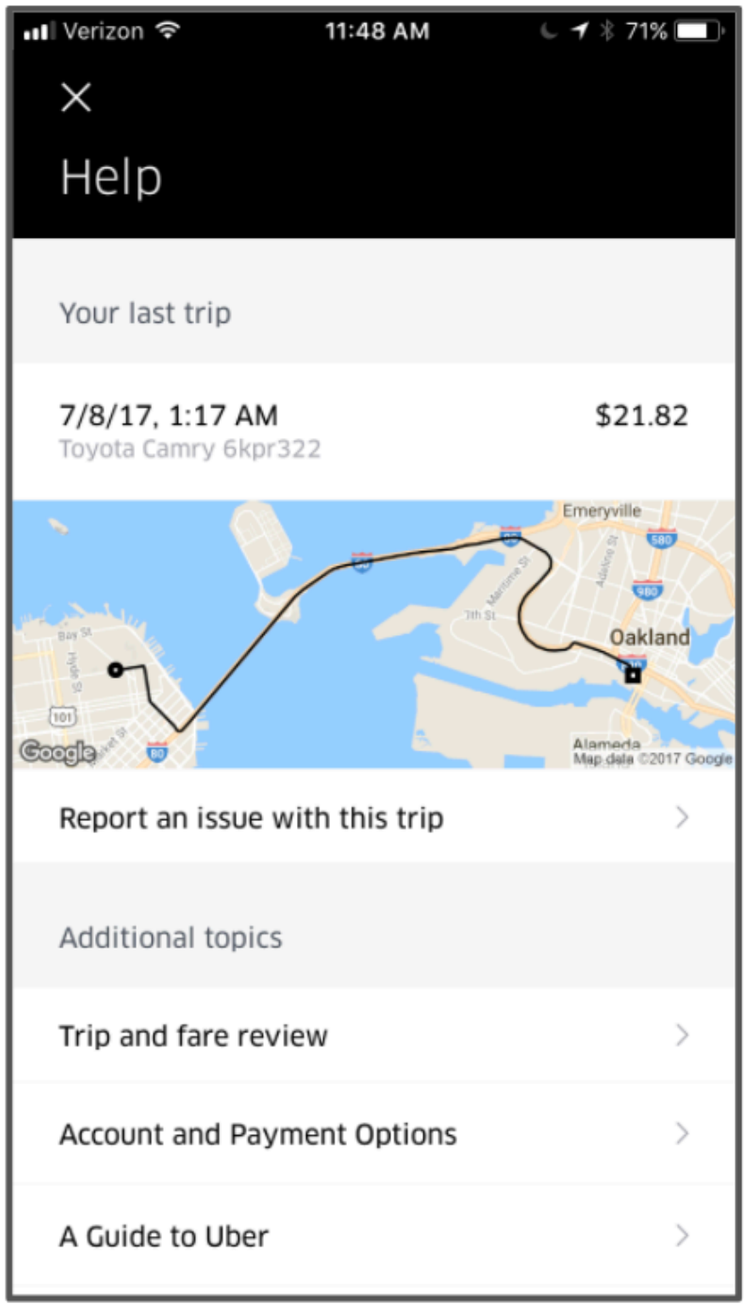
Customer Obsession Ticket Assistant (COTA) is an ML system

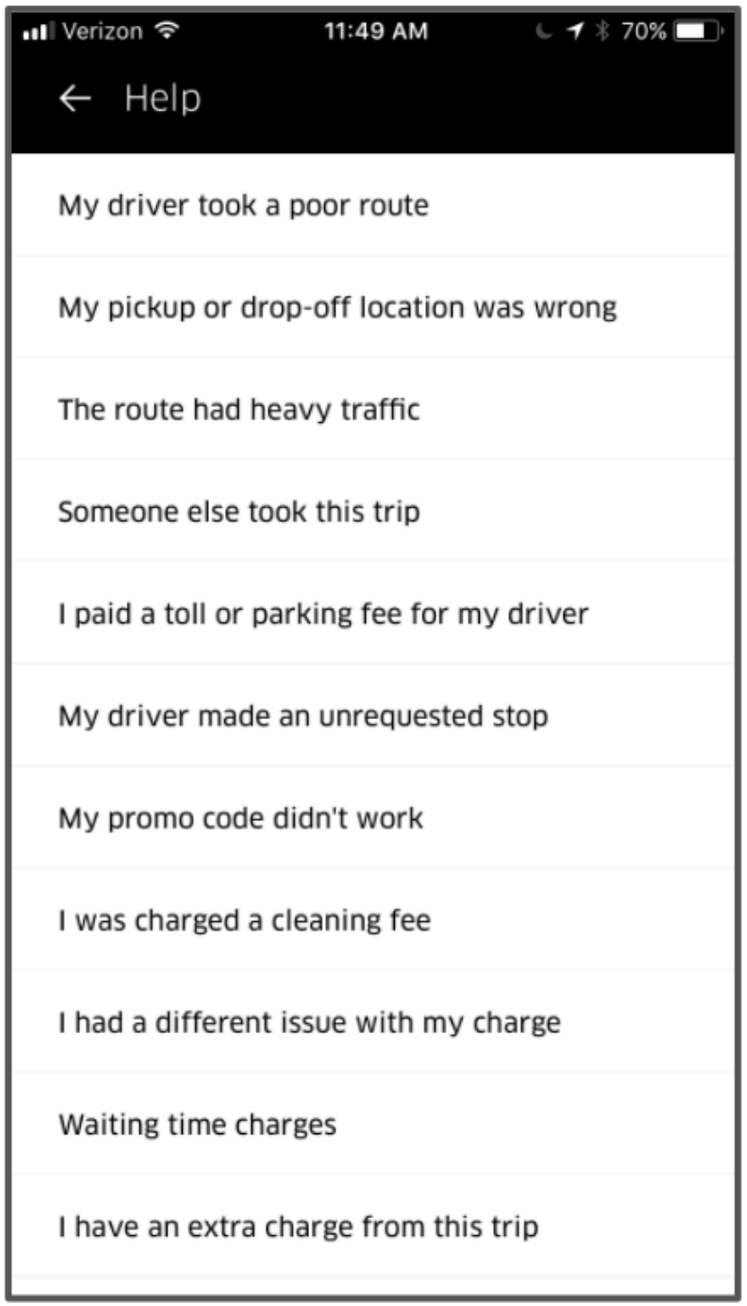
- Assists Uber agents to resolve issues
- Hundreds of thousands issues daily
- 400+ cities worldwide

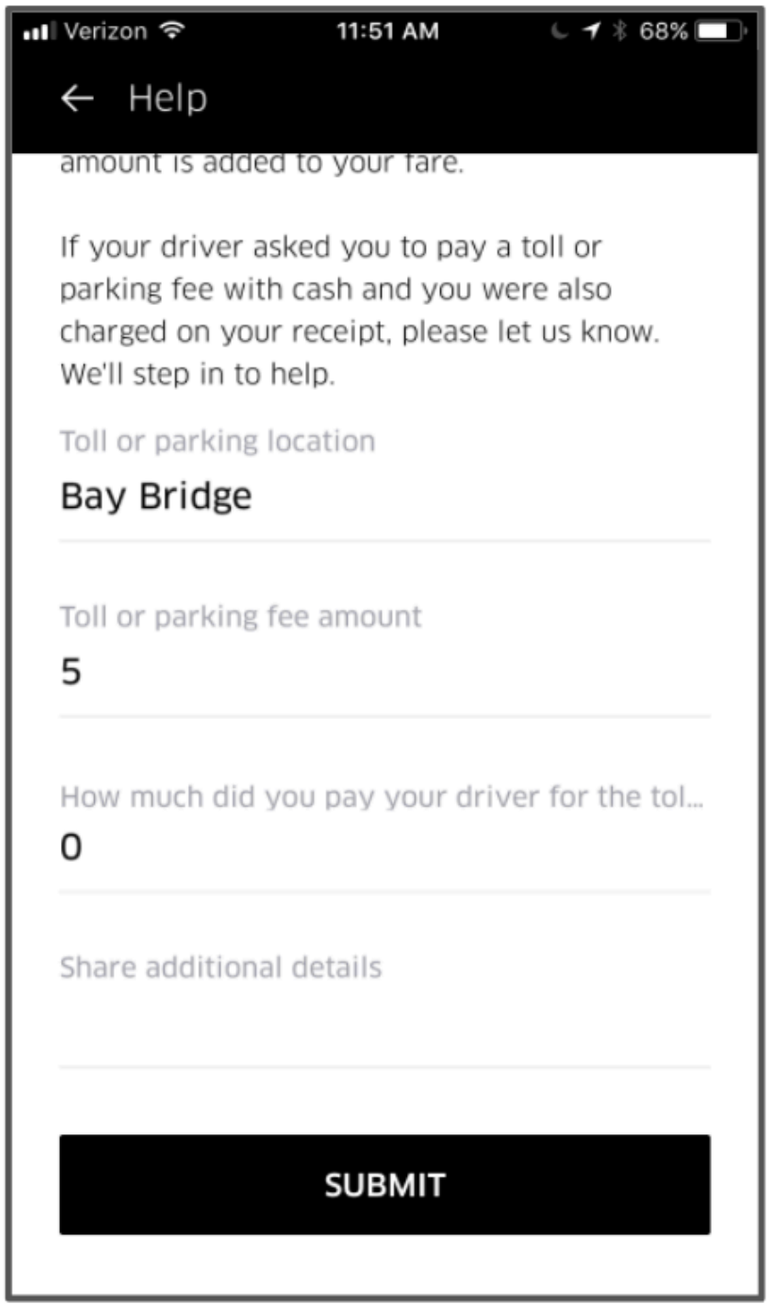
Have you ever wonder how Uber quickly resolve your issues?



Looks familiar?







So what is the challenge?

- Although this provides important context
- Not all of the information needed for solving an issue is obtainable through this process, particularly given the wide variety of possible solutions available.
- Moreover, the **diversity of ways a customer can describe an issue** associated with a ticket further complicates the ticket resolution process.

The main challenge is showing up at scale

- As Uber continues to grow at scale, support agents must be able to handle an ever-increasing **volume** and **diversity** of support tickets, from technical errors to fare adjustments.

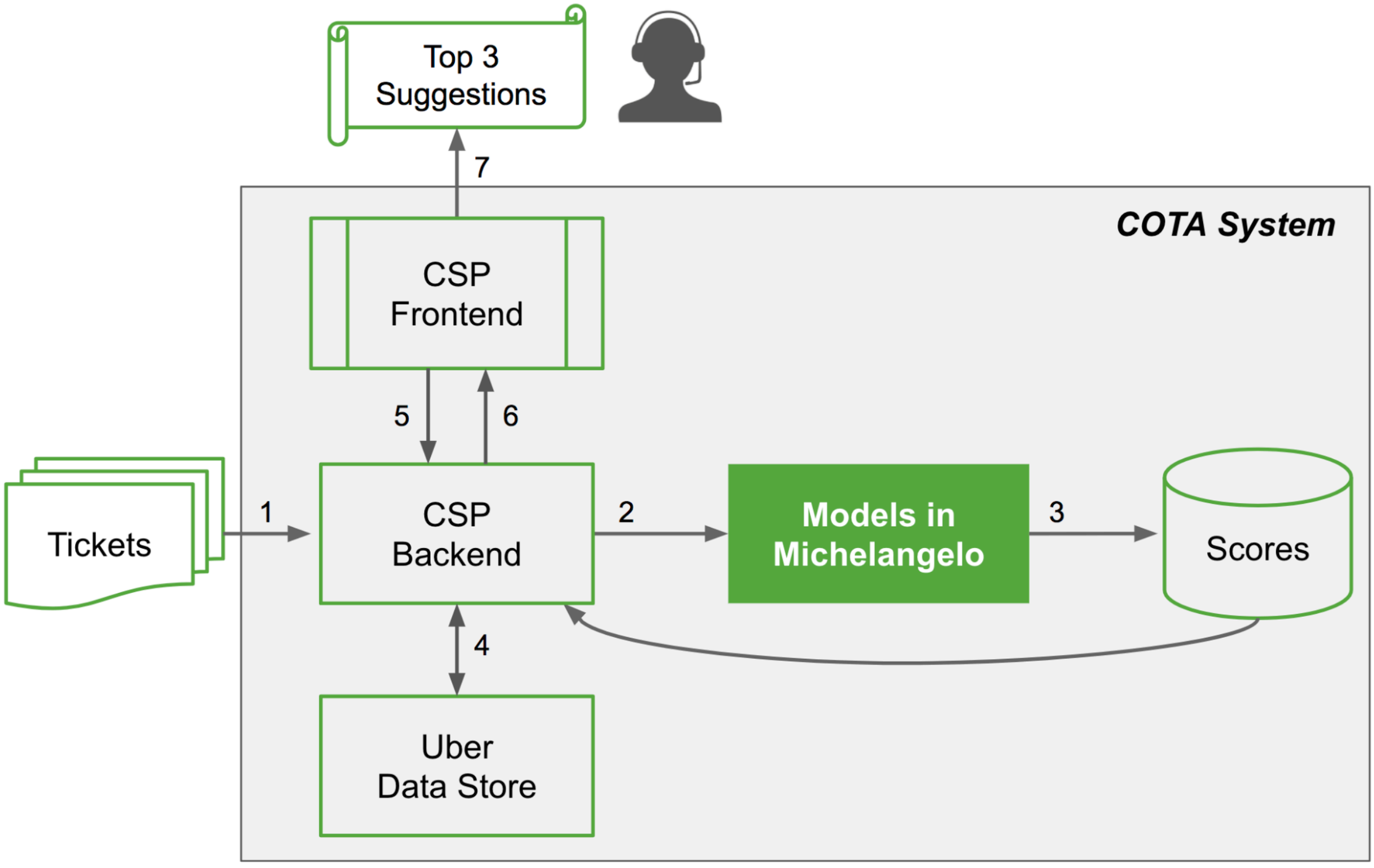
- In fact, when an agent opens a ticket, the first thing they need to do is determine the issue type out of thousands of possibilities—no easy task!

- Reducing the amount of time agents spend identifying tickets is important because it also decreases the time it takes to resolve issues for users.

Choosing resolution at scale

- Once an issue type is chosen,
- The next step is to identify the right resolution,
    - with each ticket type possessing a different set of protocols and solutions.
    - With thousands of possible resolutions to choose from, identifying the proper fix to each issue is also a time-intensive process.

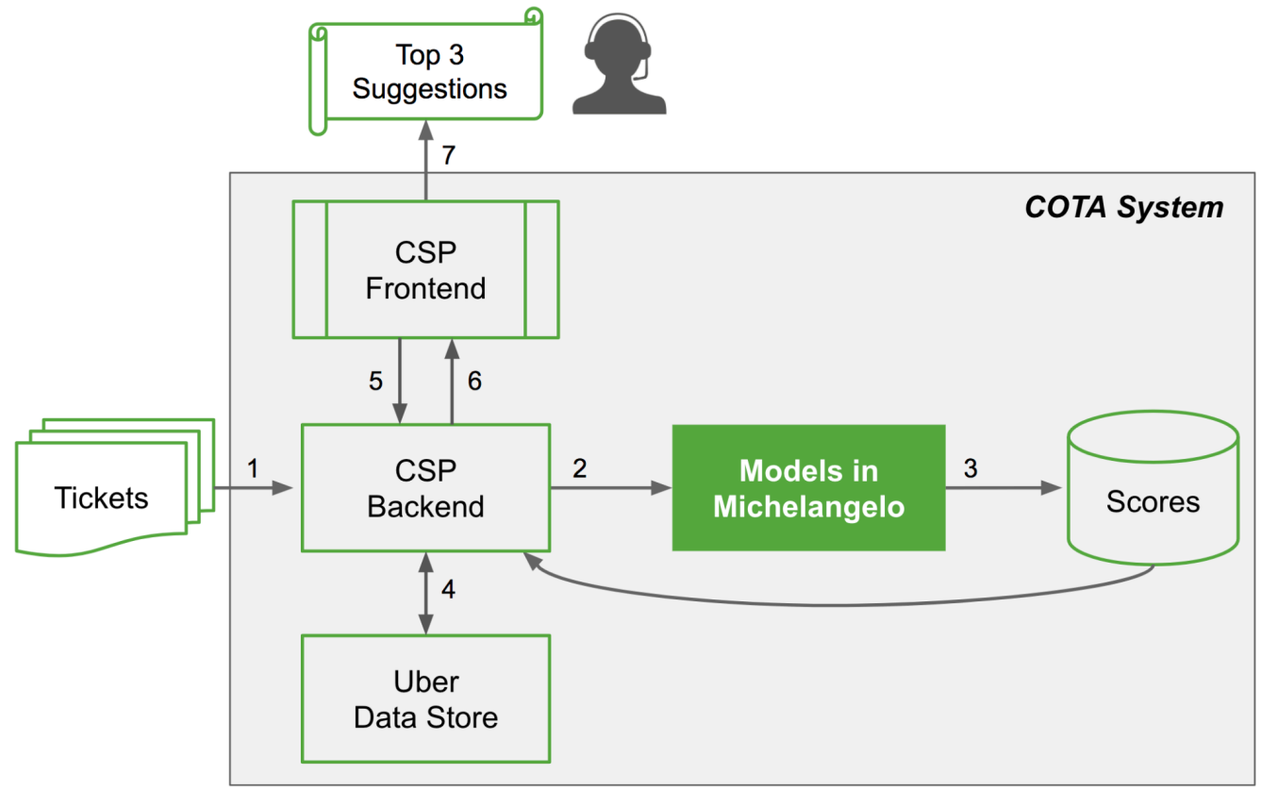
COTA: Customer Obsession Ticket Assistant



COTA composes/built on top of two subsystems

- Built on top of customer support platform,
- Michelangelo-powered models suggest the **three most likely issue types** and solutions based on ticket content and trip context.

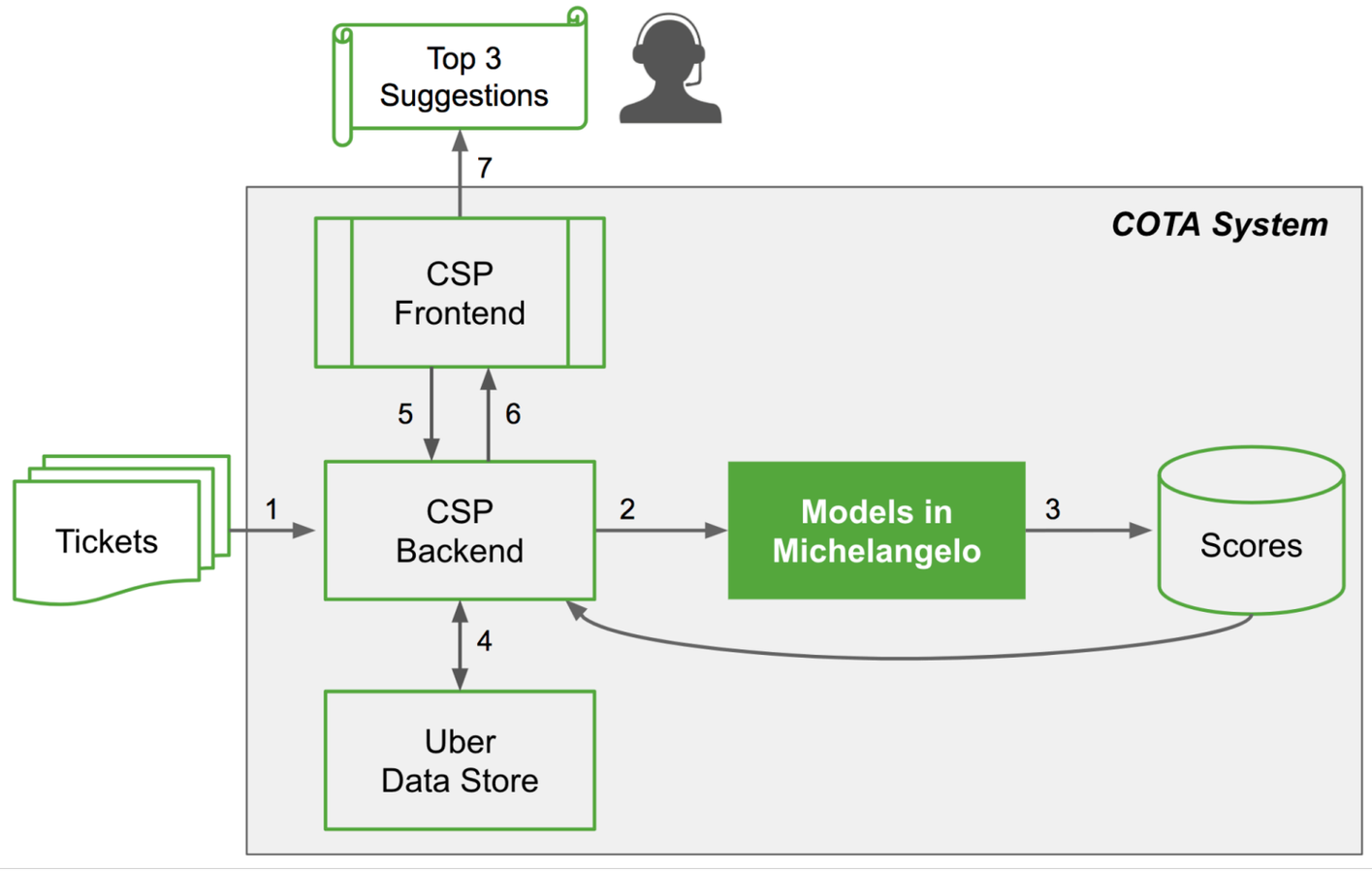
Is COTA an ML system?



COTA Architecture

1. Once a new ticket enters the customer support platform (CSP), the back-end service **collects all relevant features** of the ticket.

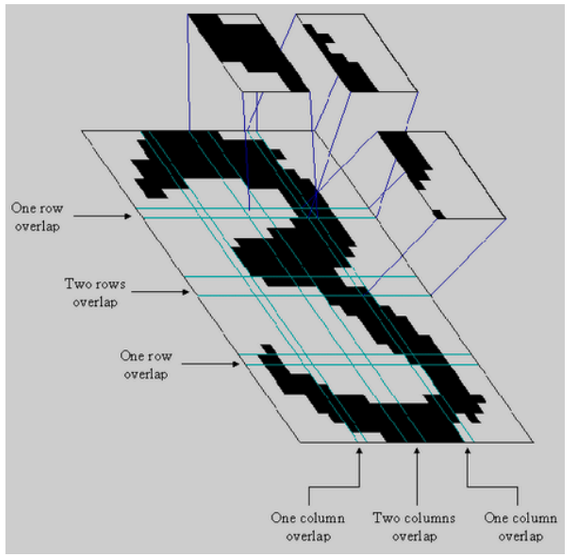
2. The back-end service then sends these features to the machine learning model in Michelangelo.

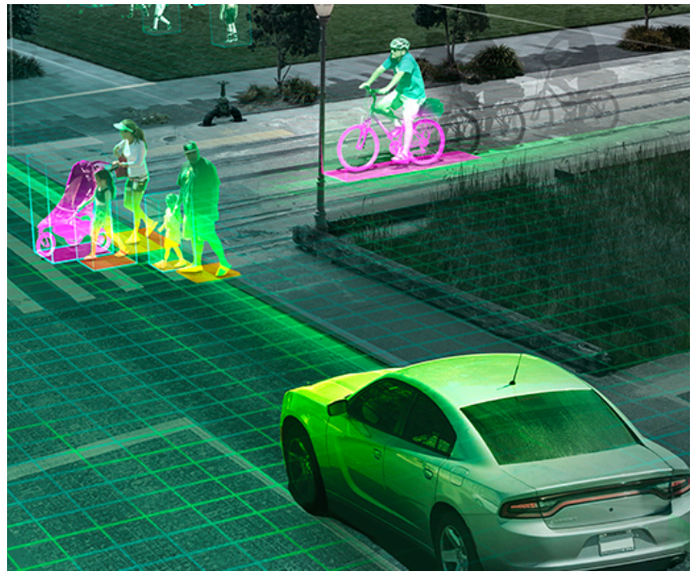


Wait, what is a feature?

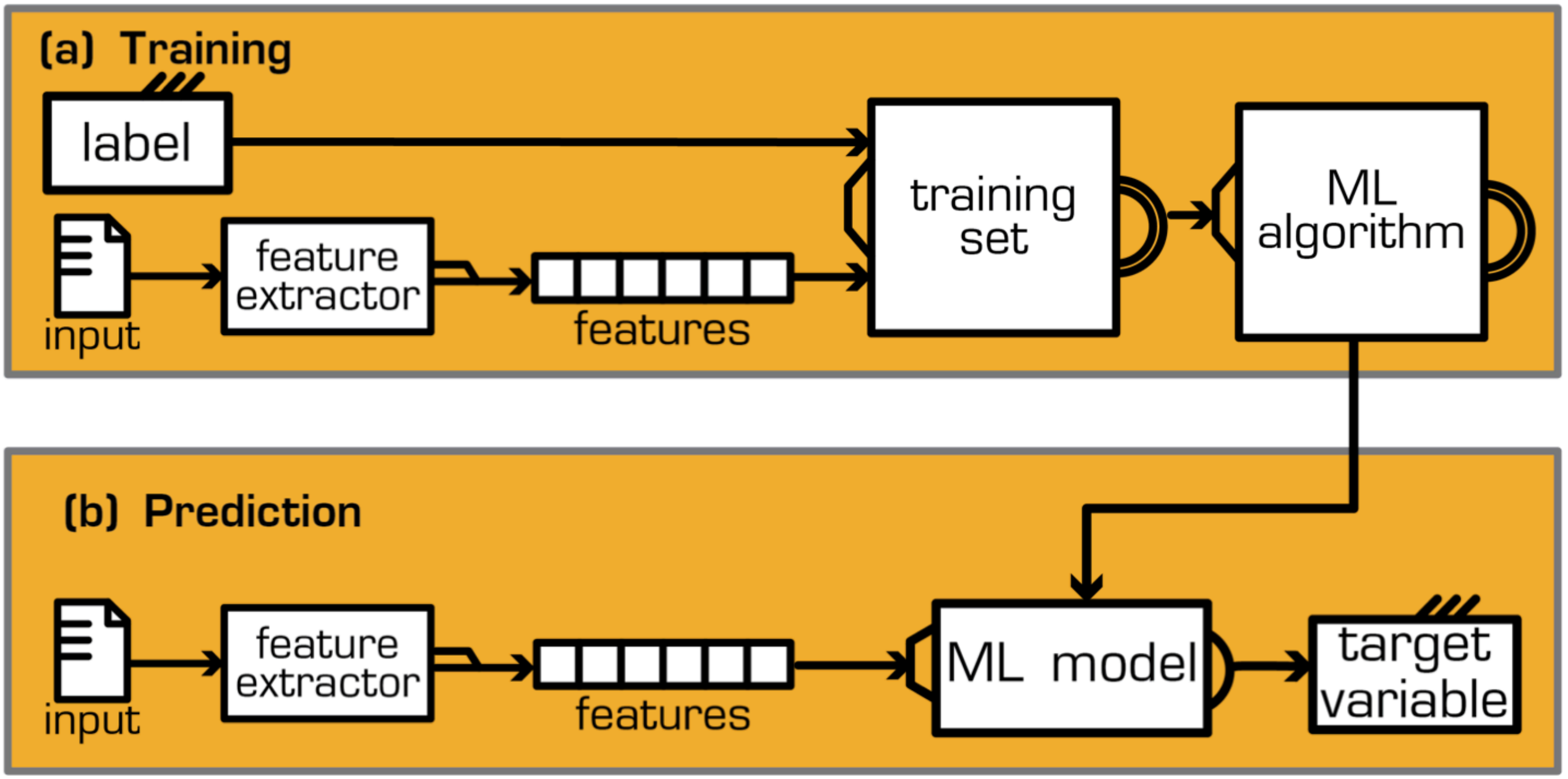
- A feature is an individual measurable property or characteristic of a phenomenon being observed.

What are the important features in each of these domains?





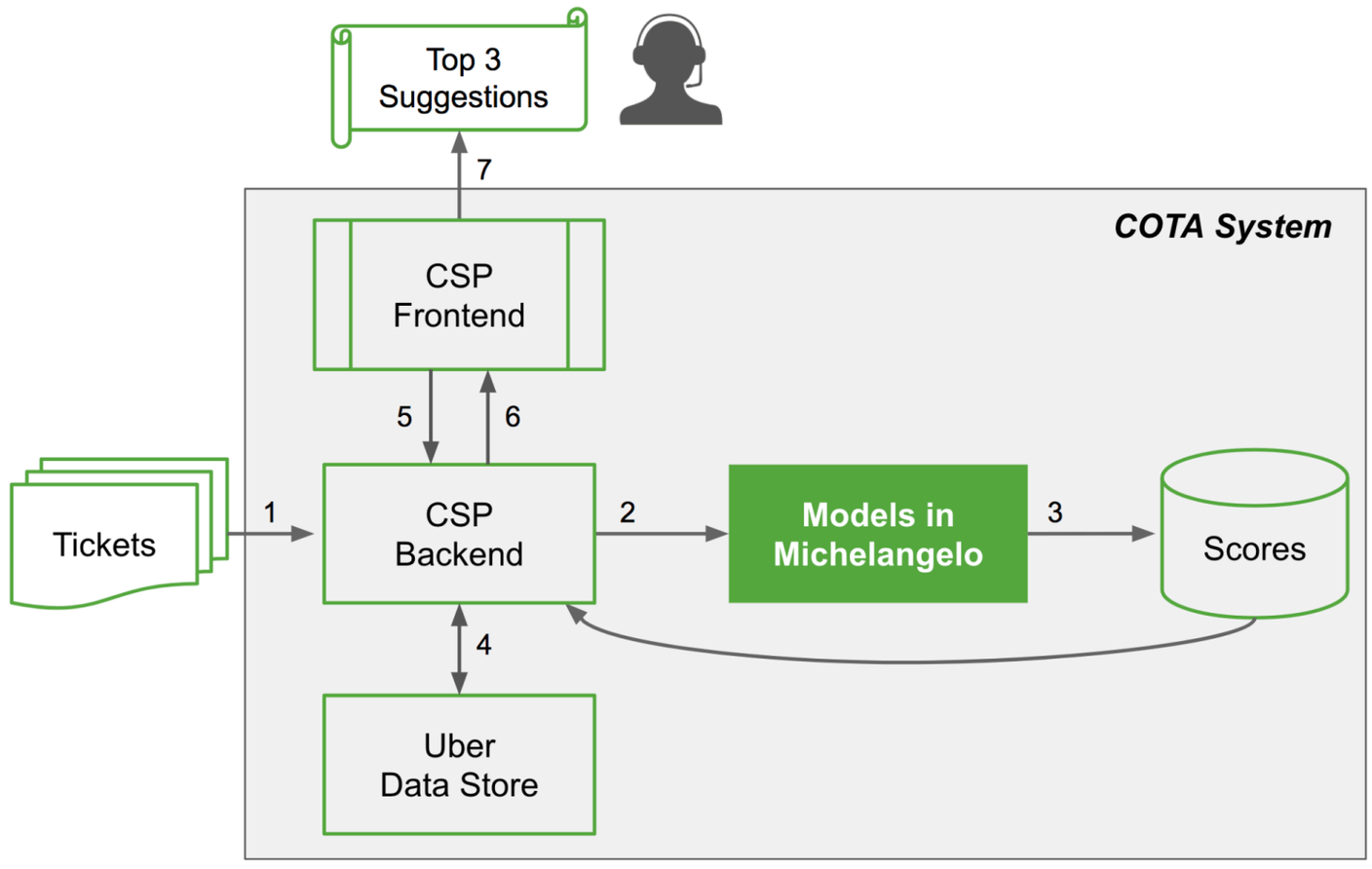
How features will be used in training and testing?



COTA Architecture

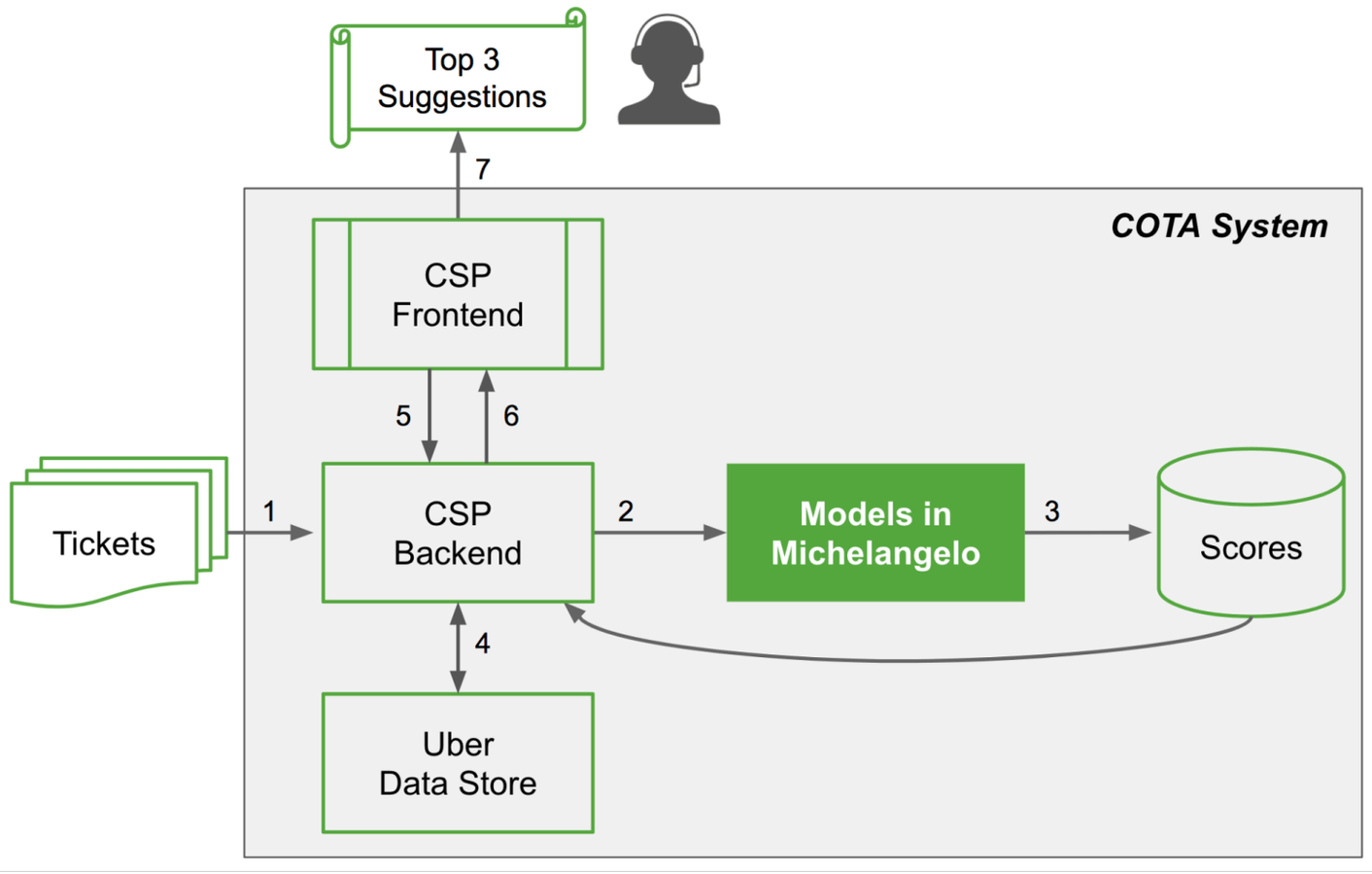
3. The model **predicts scores** for each possible solution.

4. The back-end service receives the predictions and scores, and **saves them to our Schema-less data store**.



COTA Architecture

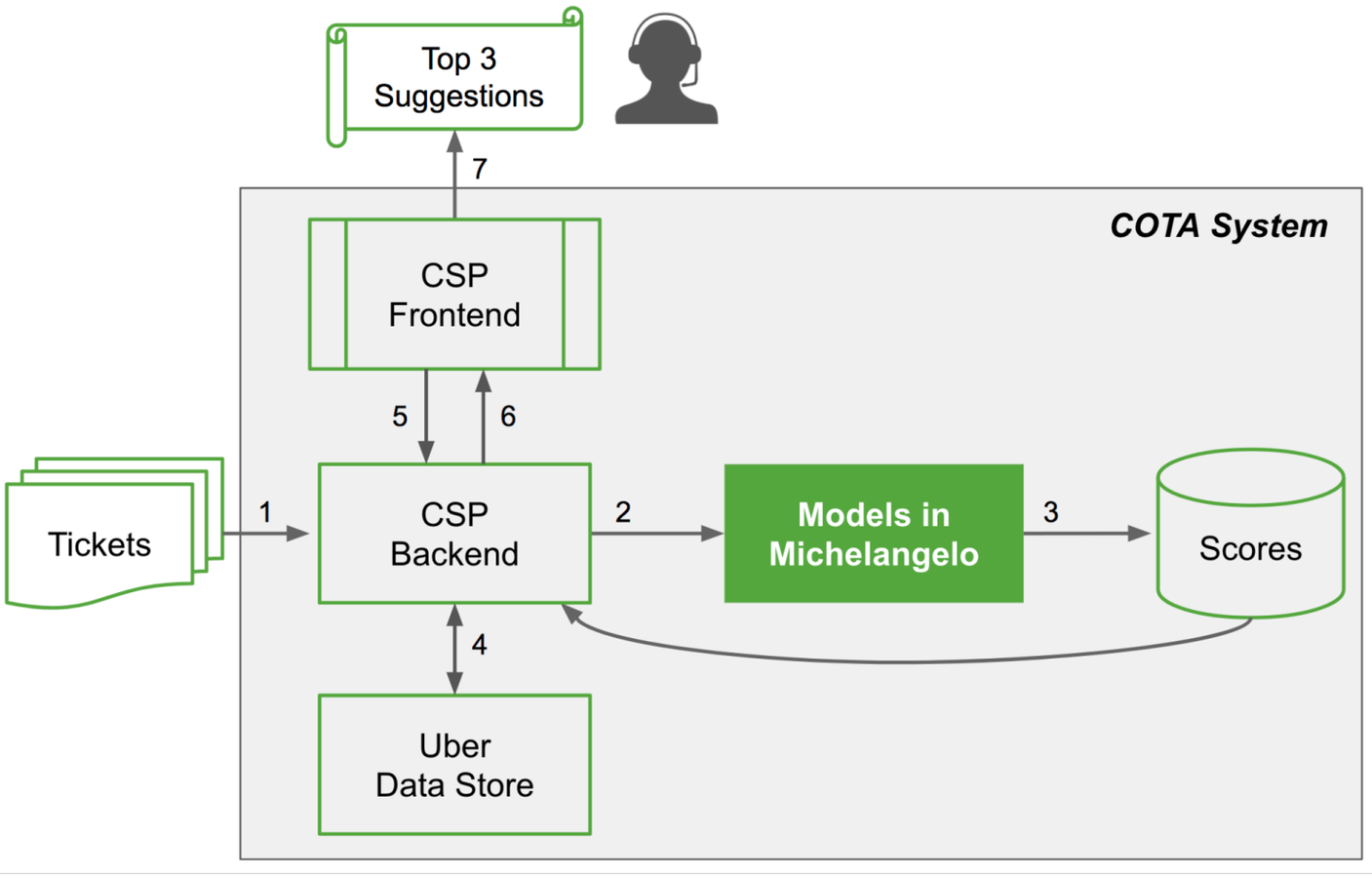
5. Once an agent opens a given ticket, the front-end service triggers the back-end service to check if there are any updates to the ticket. If there are no updates, the back-end service will retrieve the saved predictions; if there are updates, it will fetch the **updated features** and go through steps 2-4 again.



COTA Architecture

6. The back-end service **returns the list of solutions** ranked by the predicted score to the frontend.

7. The top three ranked solutions are **suggested** to agents; from there, agents make a selection and resolve the support ticket.



So what?

- Results are promising;
- COTA can **reduce ticket resolution time** by over 10 percent
- While delivering service with **similar or higher levels of customer satisfaction**

Let’s have a look at the backend

- To accomplish the goal, machine learning model leverages features extracted from
  - customer support messages,
  - trip information,
  - and customer selections in the ticket issue submission hierarchy outlined earlier.

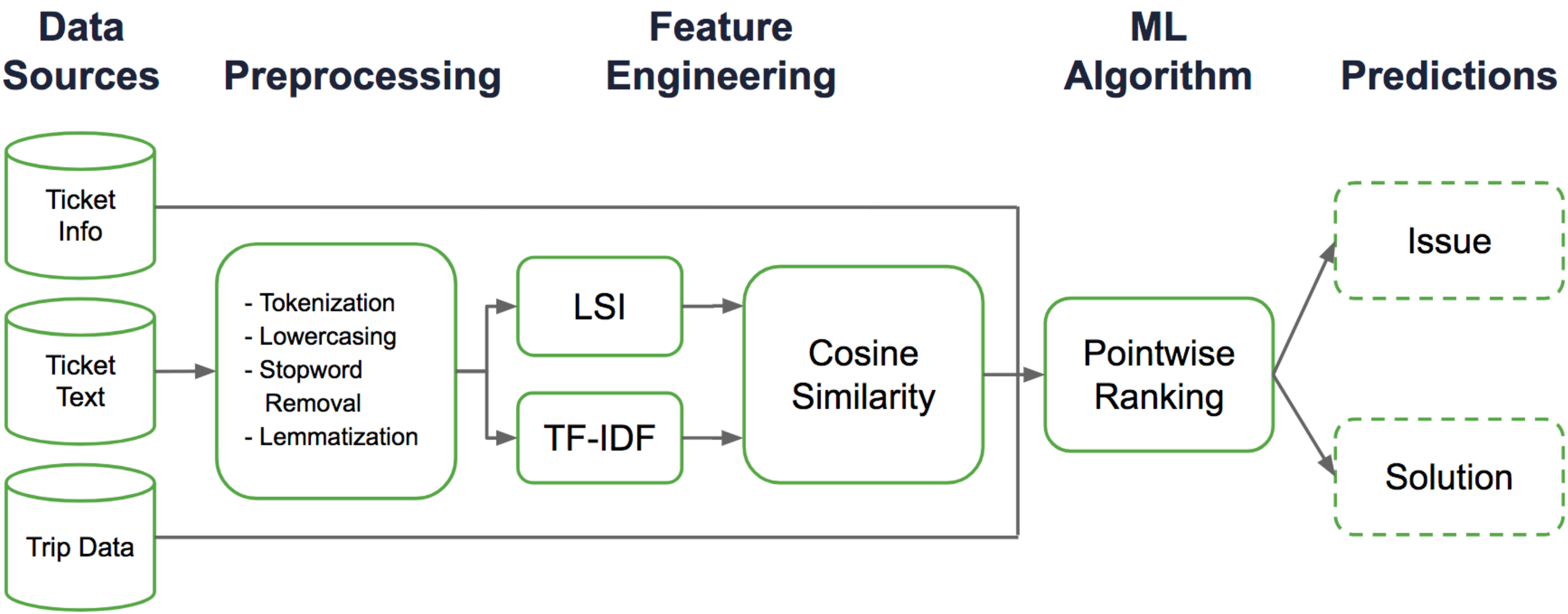
Any guess what is the most important feature of the ticket you submit to Uber?

Any guess what is the most important feature of the ticket you submit to Uber?

- The most valuable feature for identifying issue type is the **message** customers send to agents about their issue.

- Uber built a **NLP pipeline to transform text across several different languages** into useful features.

The NLP pipeline



What NLP pipeline does?

- NLP models can be built **to translate and interpret different elements of text**, including phonology, morphology, grammar, syntax, and semantics.

- Depending on the building units, NLP can also register **character-level, word-level, phrase-level, or sentence/document-level language** modeling.

- Traditional NLP models are built by **leveraging human expertise** in linguistics to engineer handcrafted features. With the recent upsurge in end-to-end training for **deep learning models**, researchers have even begun to develop models that can decipher full chunks of text without having to explicitly parse out relationships between different words within a sentence, instead using raw text directly.

What NLP pipeline does?

- Uber analyzes text at the **word-level** to better understand the **semantics of text data**.

- One popular approach to NLP is **topic modeling**, which aims to understand the meaning of sentences using the **counting statistics of the words**.

- Although topic modeling does not take into account **word ordering**, it has been proven very powerful for tasks such as information retrieval and document classification.

Preprocessing

- **Cleaning** the text by removing HTML tags.

- **Tokenizing** the message's sentences and remove stopwords.

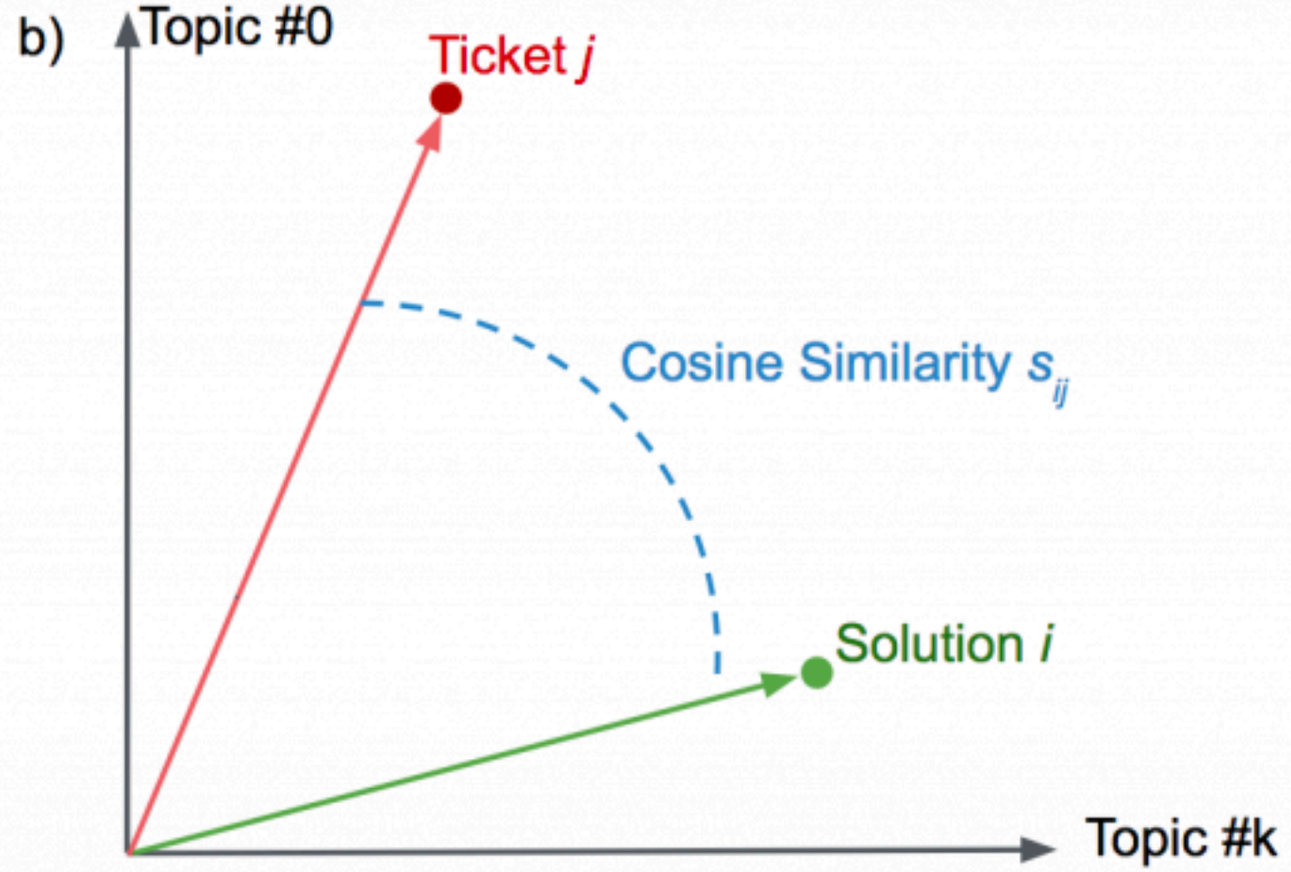
- Conducting **lemmatization** to convert words in different inflected forms into the same base form.

- Finally, **converting** the documents into a collection of words (a so-called bag of words) and build a dictionary of those words.

Topic modeling

Topic Modeling: TF-IDF and LSA to extract topics from rich text data in customer support tickets processed by our customer support platform.

Feature Engineering: All the solutions and tickets are mapped to the topic vector space, and cosine similarity between solution and ticket pairs are computed.



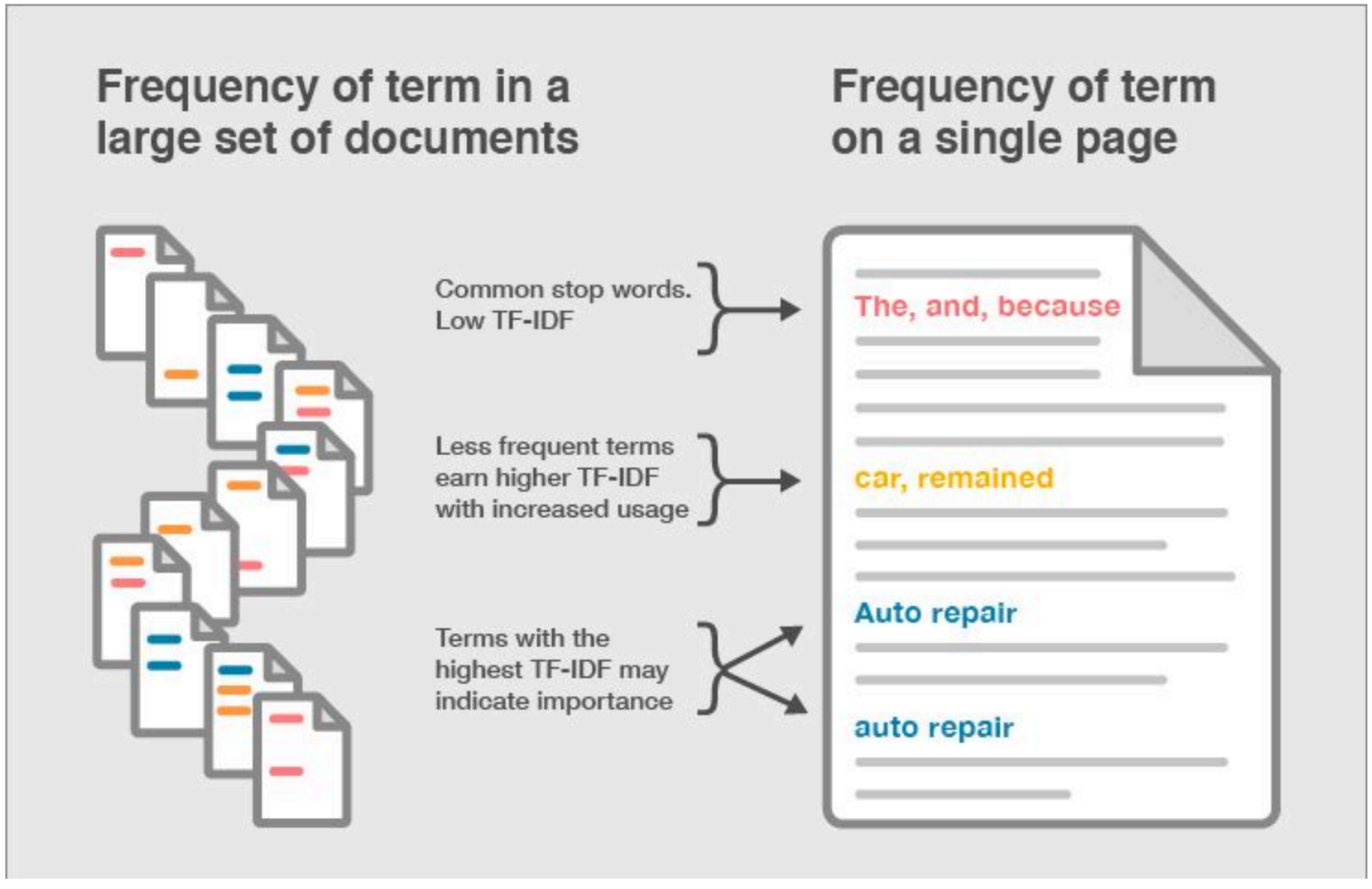
TF-IDF, ha?

- Tf - Term Frequency
- Idf - Inverse document frequency

Quiz!

- Would you weight common words higher , or rare words?

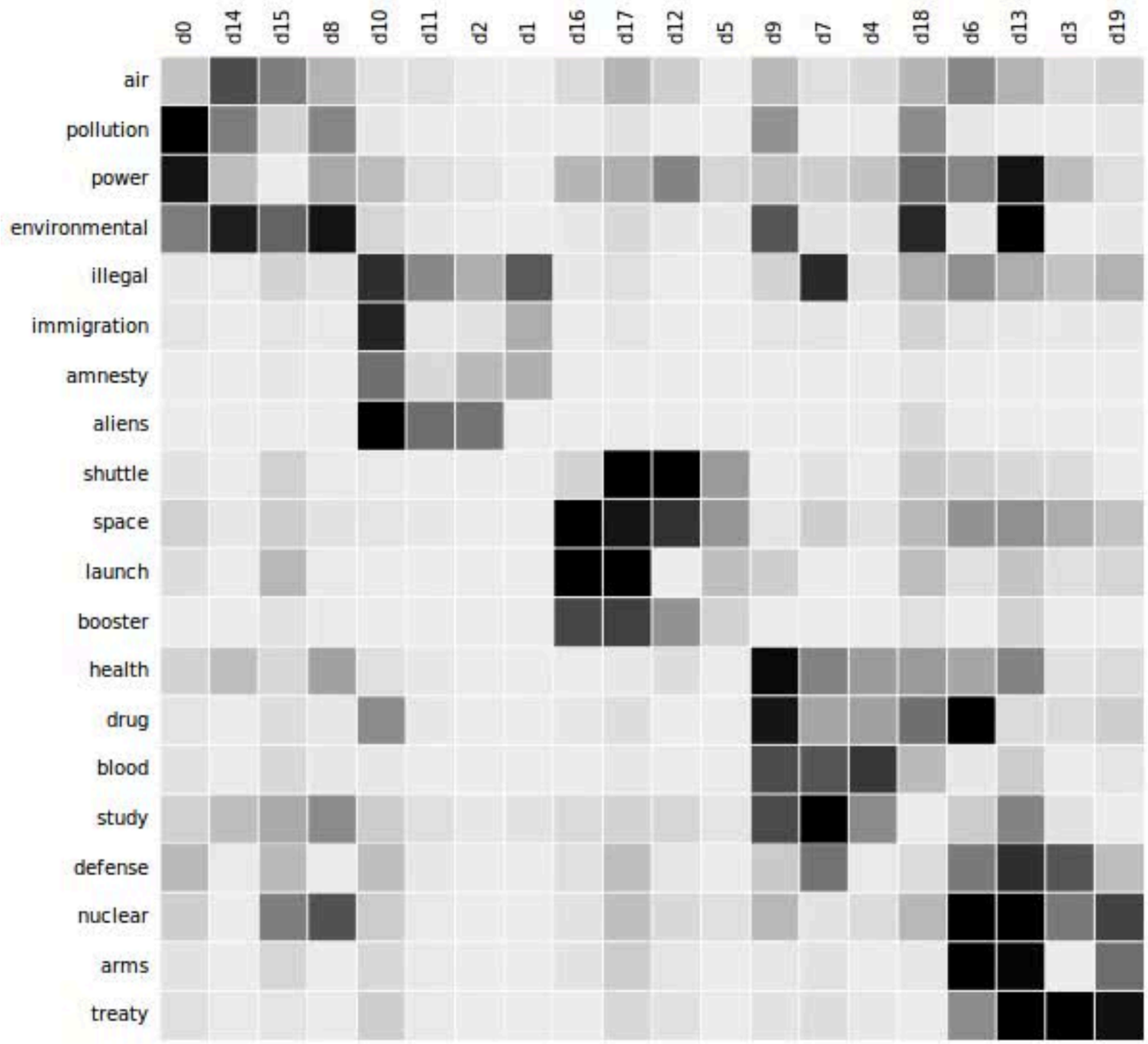
TF-IDF, ha?


Frequency of term in a large set of documents
Frequency of term on a single page
Common stop words. Low TF-IDF
Less frequent terms earn higher TF-IDF with increased usage
Terms with the highest TF-IDF may indicate importance
The, and, because
car, remained
Auto repair
auto repair

LSA, ha?

- LSA - Latent Semantic Analysis

LSA, ha?


d0
d14
d15
d8
d10
d11
d2
d1
d16
d17
d12
d5
d9
d7
d4
d18
d6
d13
d3
d19
air
pollution
power
environmental
illegal
immigration
amnesty
aliens
shuttle
space
launch
booster
health
drug
blood
study
defense
nuclear
arms
treaty

Feature engineering

- Topic modeling enables us to directly use the **topic vectors as features** to perform downstream classifications for issue type identification and solution selection.

Do you guess what could possibly go wrong?

Training becomes challenging at scale

- This direct approach suffers from a **sparsity of topic vectors**;

- In order to form a meaningful representation of these topics, we typically need to keep hundreds or even **thousands of dimensions of topic vectors** with many dimensions having values close to zero.

- With a very **high-dimensional feature space and large amount of data to process**, training these models becomes quite challenging.

How Uber solved the challenge?

- Performing further feature engineering by computing **cosine similarity** features.

- Using solution selection as an example, we collect the **historical tickets** of each solution and form the bag-of-word representation of such a solution.

Cosine similarity, ha?

$$\mathbf{A} \cdot \mathbf{B} = \| \mathbf{A} \| \ \| \mathbf{B} \| \cos\theta$$

Cosine similarity, ha?

$$\textbf{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Pointwise ranking

- Combined cosine similarity features together with other ticket and trip features that matches tickets to solutions

Do you guess what is the challenge now?

Solution space becomes large

- With over **1,000** possible solutions
- For **100s** of ticket types,

- Solution space becomes a challenge for the ranking algorithm of distinguishing the fine differences between these solutions.

How Uber solved the challenge?

- Learning to rank!

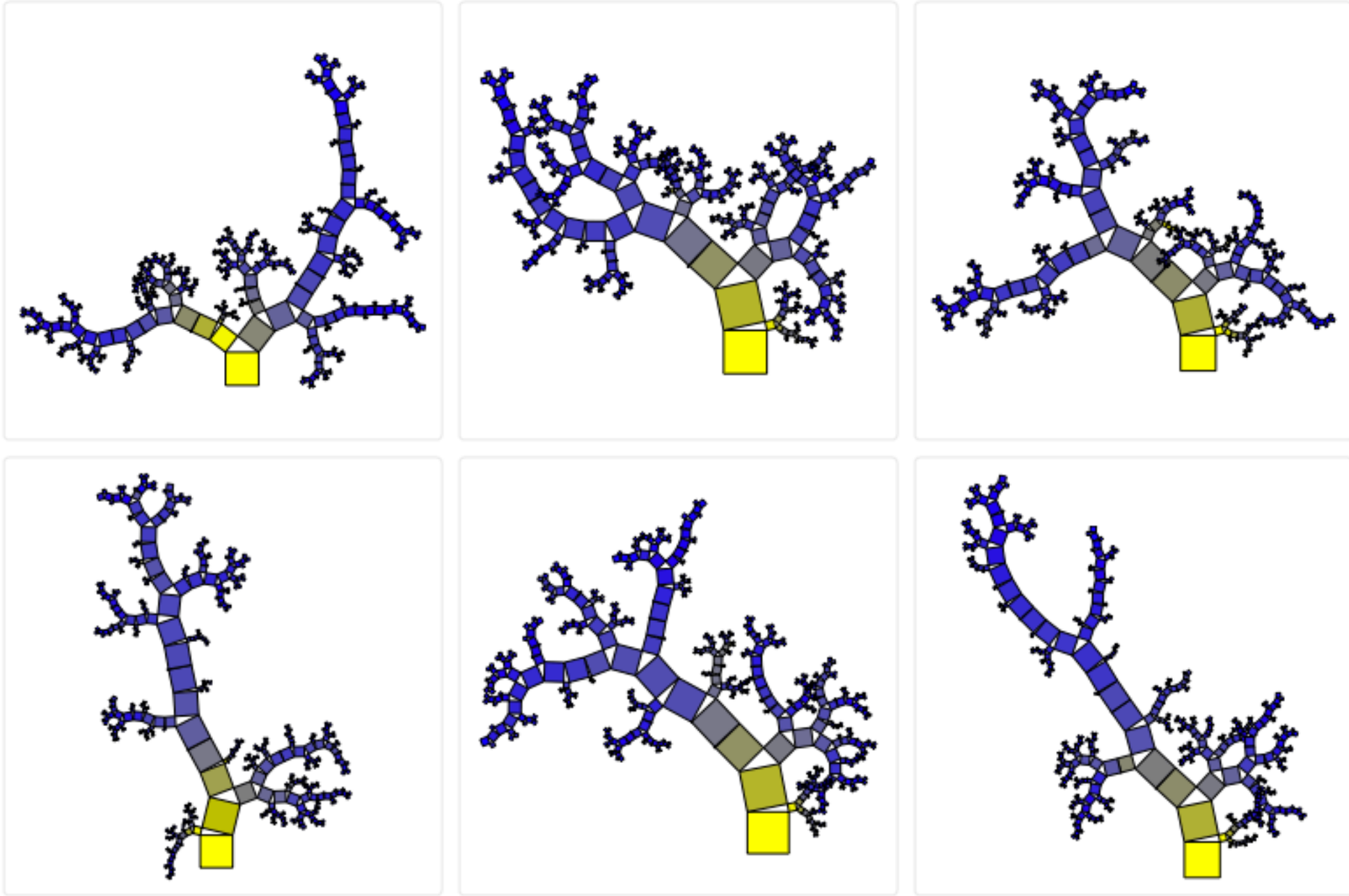
How Uber solved the challenge?

- Label the correct match between solution and ticket pair as **positive**

- Sample a random subset of solutions that do not match with the ticket and label the pairs **negative**

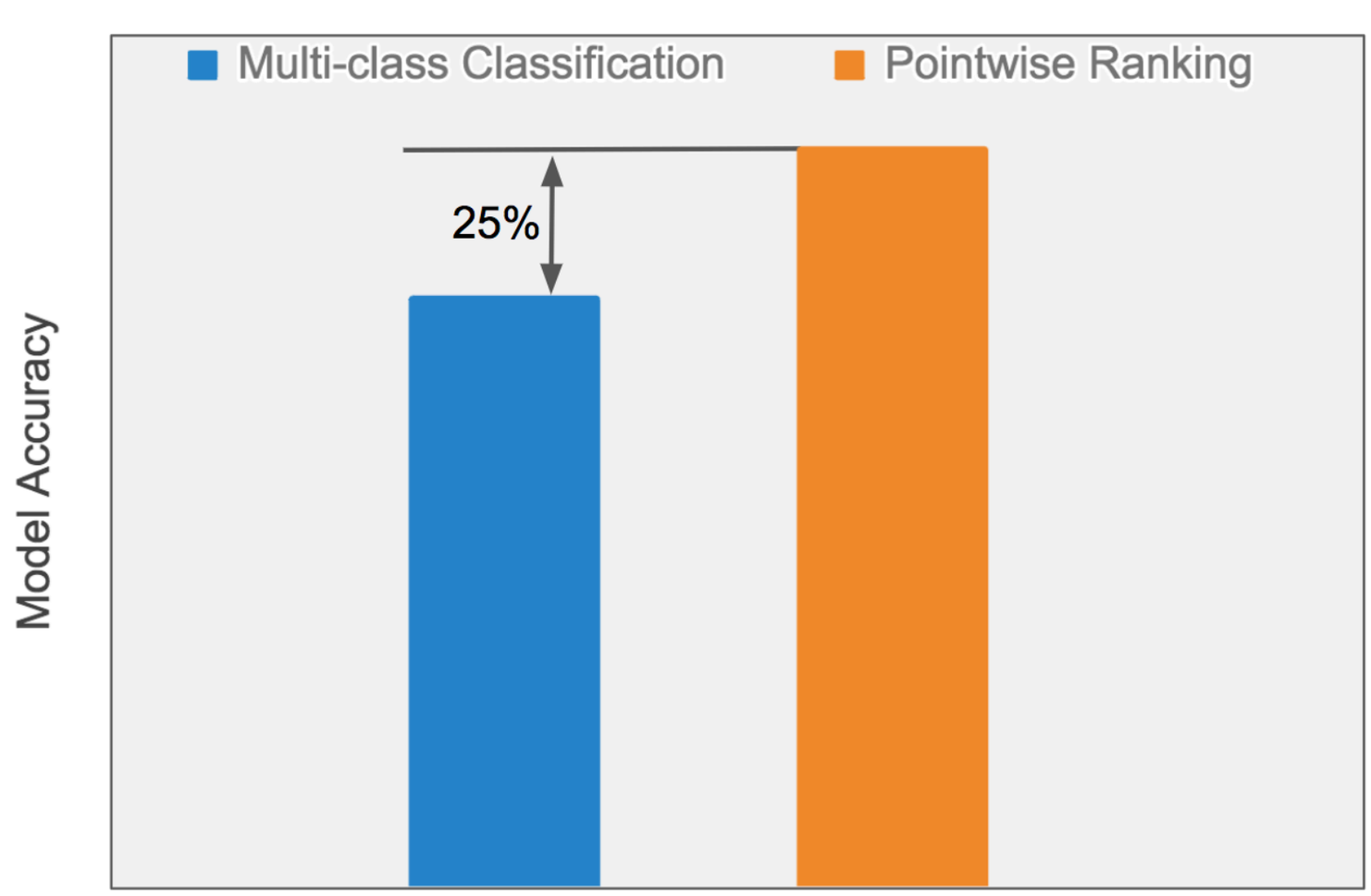
- Using the cosine similarity as well as ticket and trip features, they built a **binary classification** algorithm that leverages the **random forest** technique to classify whether or not each solution-ticket combination matches.

Random forest, ha?



Results

- 25 percent relative improvement in accuracy
- Speeds up the training process by 70 percent



Easier and faster ticket solving = better customer support

But wait, how they actually measured they were successful for handling customer issues?

Uber performed A/B tests

- To measure COTA's impact on customer support experience, Uber conducted several controlled A/B test experiments online on English language tickets.

- Included thousands of agents and randomly assigned them into either control or treatment groups.

- Agents in the control group were exposed to the original workflow, while agents in the treatment group were shown a modified user interface containing suggestions on issue types and solutions.

- We collected tickets solved solely by either agents in the control or treatment group, and measured a few key metrics, including **model accuracy, average handle time, and customer satisfaction score**.

Summary of results

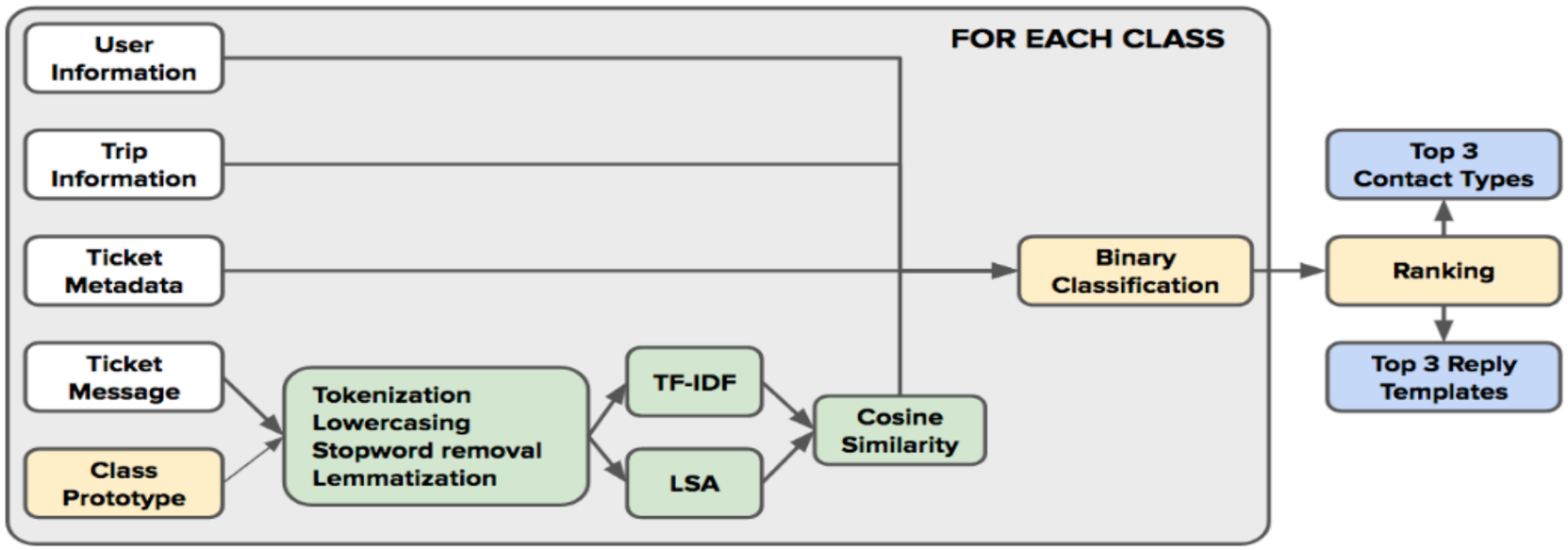
- Measured the online model performance for both groups and compared them with offline performance. The model **performance is consistent** from offline to online.

- Measured customer satisfaction scores and compared them across control and treatment groups. Customer satisfaction often increased by a few percentage points. This finding indicates that COTA delivers the **same or slightly higher quality of customer service**.

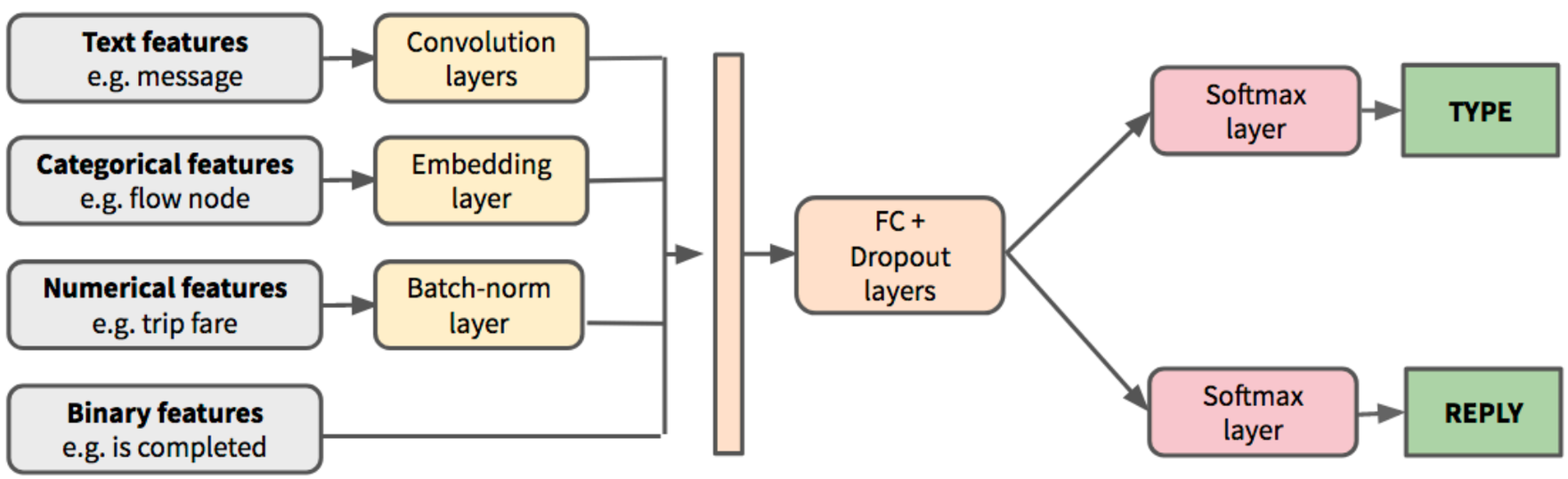
- To determine how much COTA affected ticket resolution speed, we compared the average ticket handling time between the control and treatment groups. On average, this new feature reduced ticket **handling time by about 10 percent**.

Next?
Moving to COTA v2 with deep learning

a)



b)



Project Example

Deep learning experiments with various architectures

- Convolutional neural networks (CNNs),
- Recurrent neural networks (RNNs),
- And several different combinations of the two, including hierarchical and attention-based architectures.

Training process

- Trained models in a multi-task learning fashion, with a single model capable of both identifying the issue type and suggest the best possible solution.

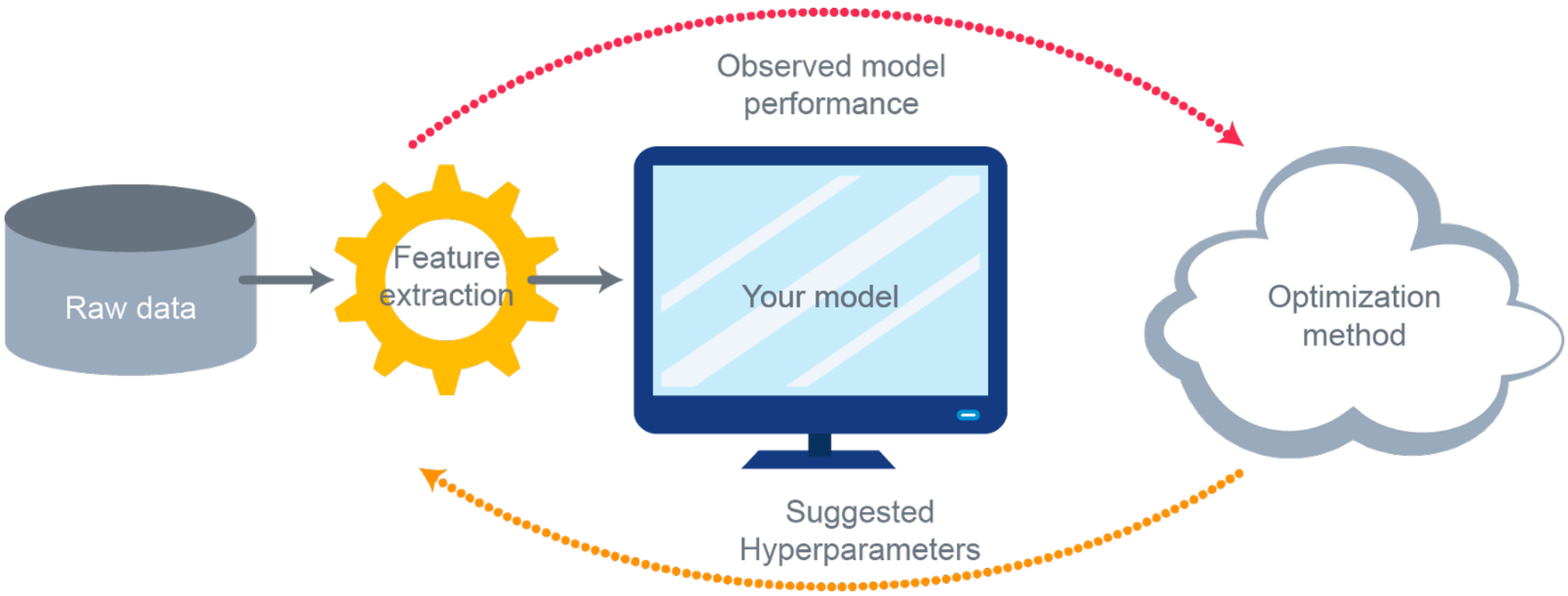
- Since issue types are organized into a hierarchy, we determined that we could train the model to predict the path in the hierarchy with a recurrent decoder using **beam search**.

Hyperparameter optimization to select the best model

- Performed large scale hyperparameter optimization for all types of architectures, training them in parallel on our GPU cluster.

What is Hyperparameter optimization?

NAME	DESCRIPTION	TYPE	MIN	MAX
batch_size	SGD parameter	int	8	32
conv_1_filter_size	Architecture parameter	int	2	10
conv_1_num_filters	Architecture parameter	int	32	256
conv_2_filter_size	Architecture parameter	int	2	10
conv_2_num_filters	Architecture parameter	int	32	256
conv_3_filter_size	Architecture parameter	int	2	10
conv_3_num_filters	Architecture parameter	int	32	256
log_beta_1	Adam SGD parameter	real	-4.6	-0.7
log_beta_2	Adam SGD parameter	real	-13.8	-0.7
log_decay	Adam SGD parameter	real	-23	-2.3
log_epsilon	Adam SGD parameter	real	-23	-13.8
log_lr	Adam SGD parameter	real	-23	0

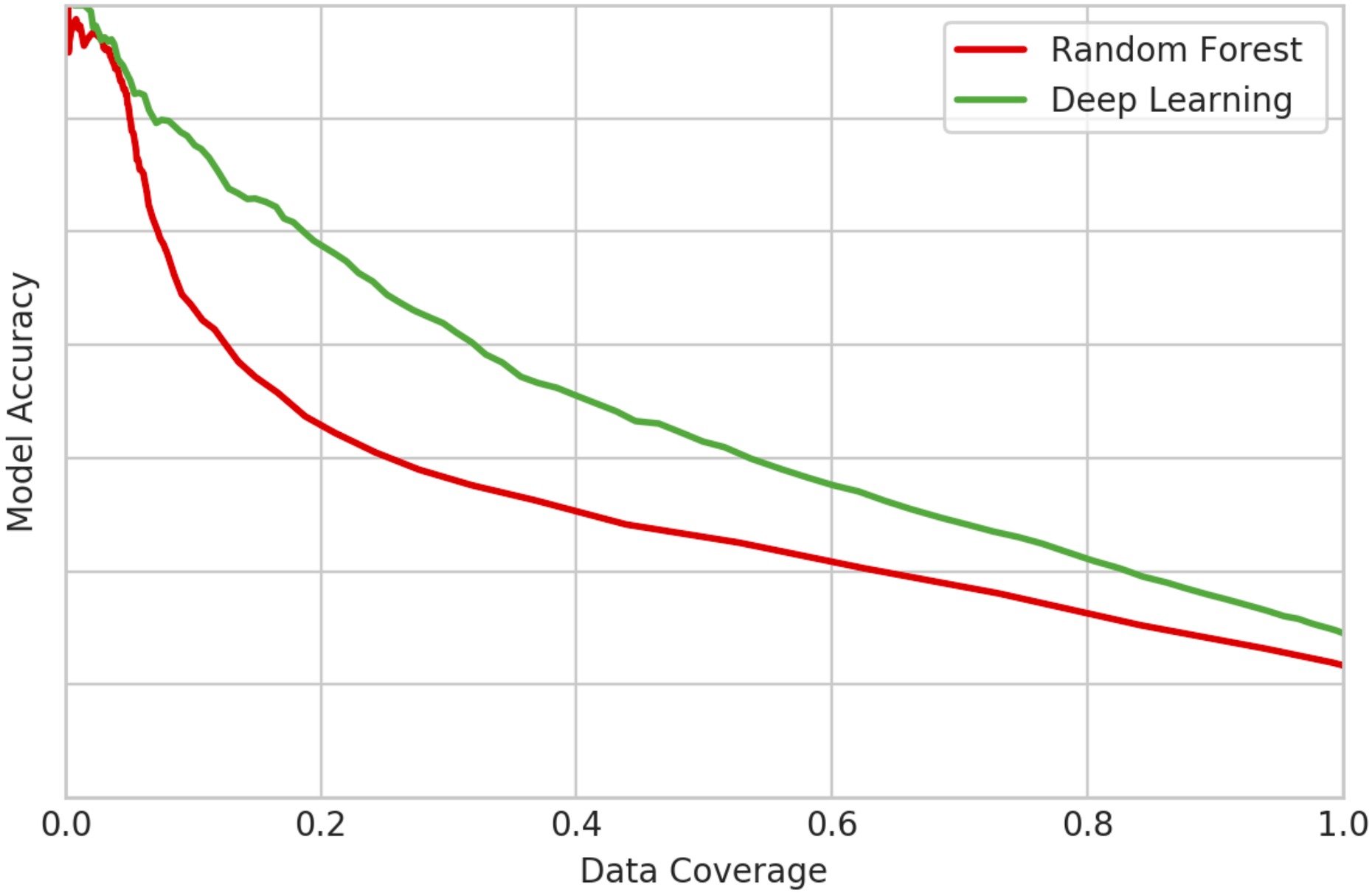


Why hyper-parameter optimization is challenging?

Performance tradeoffs

- The final results suggest that the most accurate architecture is one that **applies both CNNs and RNNs**,
- Uber chose a simpler CNN architecture that was slightly less accurate but had more **advanced computational properties** in terms of training and inference time.
- In the end, the model Uber settled on provides about 10 percent greater accuracy with respect to the original random forest model.

Tradeoff between accuracy and data coverage



Summary

- We studied what an ML system looks like via a real-world system

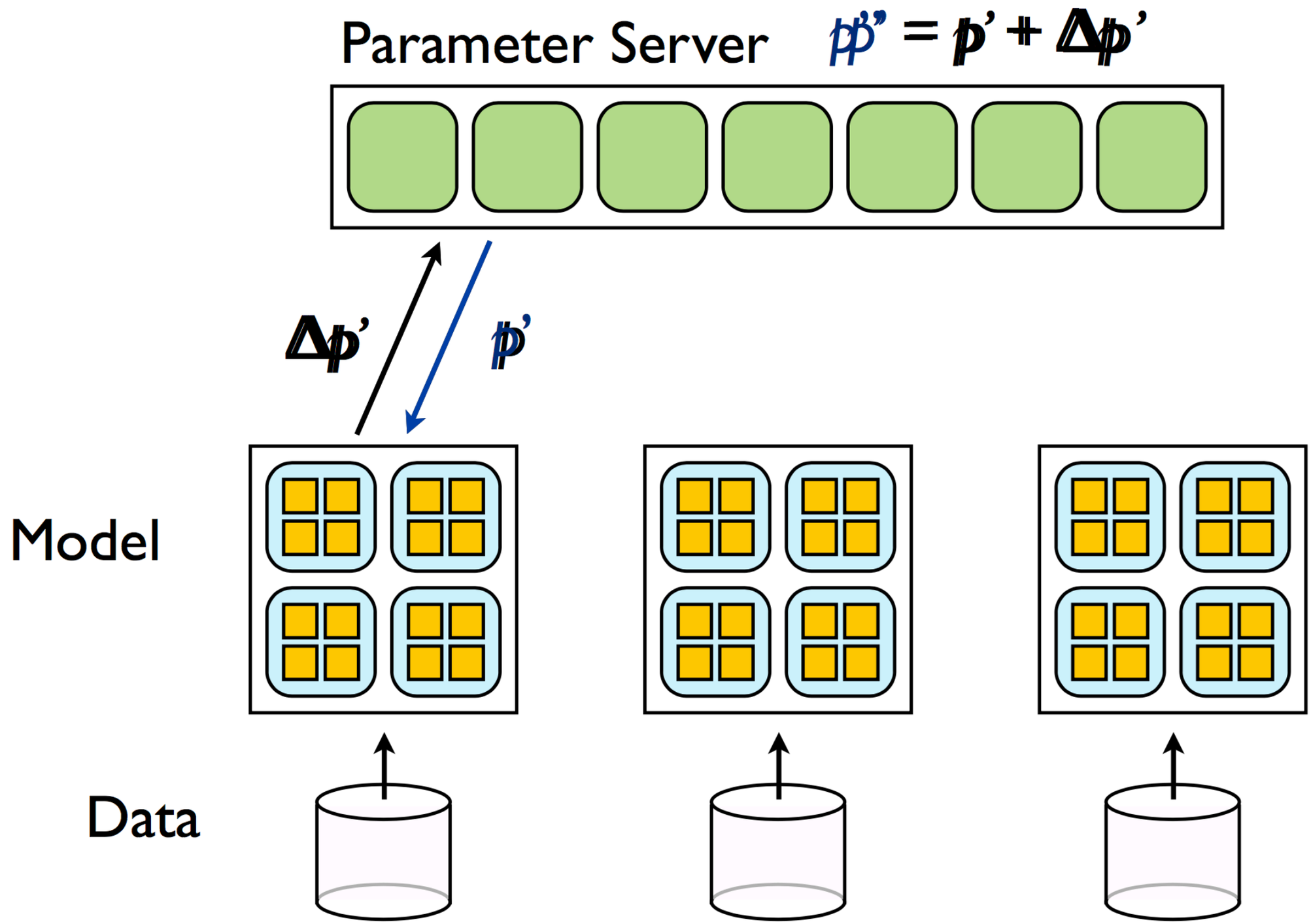
- We reviewed the architecture of the ML system and challenges of building such system

- We reviewed some solution that helped Uber to scale and answer to customer issues more quickly

Projects

- **Project2 (Distributed ML)**: How the choice of configuration parameters in distributed ML setting affect performance indicators of training and inference (training time, inference time, energy)?

Data parallelism



Data parallelism