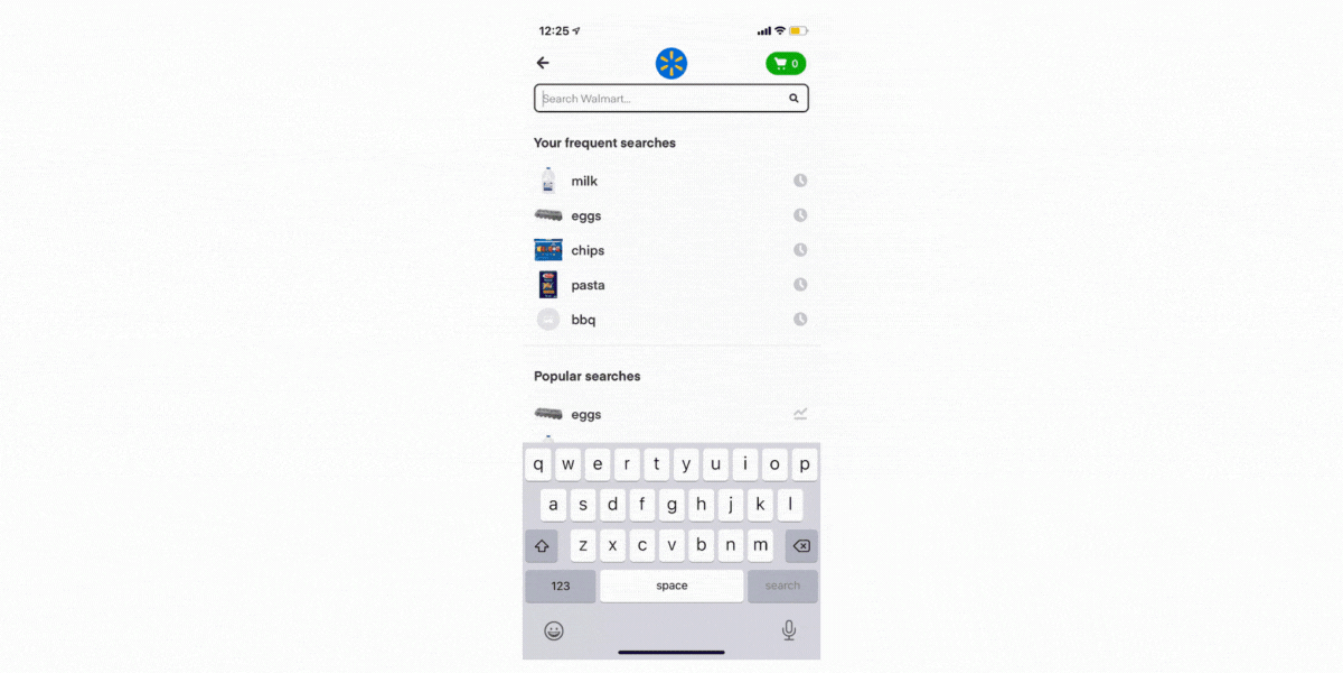
How Instacart Uses Machine Learning-Driven Autocomplete to Help People Fill Their Carts

Esther Vasiete · Follow
Published in tech-at-instacart
10 min read · Aug 24, 2022

Authors: Esther Vasiete Tejaswi Tenneti

For an eCommerce platform like Instacart, Search is an important tool for making shopping feel effortless by helping people find the products they want, even when they are not sure what they're looking for. Autocompleting user's queries when they start typing is a great way of making the search experience more efficient. It assists users in expressing their shopping intent clearly, and helps them explore a wide variety of options. In this post, we describe how we generate and rank query suggestions in Autocomplete and how this shapes a user's search behavior — translating into larger basket sizes.



Autocomplete (AC) is an important surface to ensure that our users have a delightful shopping experience. First, it helps them save time by letting them effortlessly find the products they want. Next, it inspires them to explore the full breadth of our catalog and try new products. Such exploration is important since it helps our customers get more value out of Instacart and engage more with the app beyond their regular shopping list. Finally, it is used to guide them in their journey by deeply understanding their current shopping session and recommending searches that will help our customers find the items they want.

At Instacart, we designed our Autocomplete to create a great search experience by helping our users express their search intent through fewer keystrokes, thus saving time and effort. We also want to guide our users into issuing *meaningful* queries — which means suggesting queries that are likely to produce highly relevant results that closely match the user's needs, correcting any typos, and nudging users towards broad intent queries which lets them fully explore the full breadth of our catalog. Finally, we want to provide relevant suggestions as fast as possible — ensuring a smooth search experience for our users.

Where the suggestions come from

We rely on previous customer searches across Instacart to determine a comprehensive set of suggestions that can be used to predict the customer's search query. The advantage of using a dataset based on our search logs is that it provides rich natural language suggestions that users can easily relate to. Our vocabulary consists of 57k words, extracted from 11.3 Million eligible products and brands — leading to ~785k distinct autocomplete terms across all retailers.

Since user queries need not always be well formed, we apply a set of rules to filter out candidates, such as:

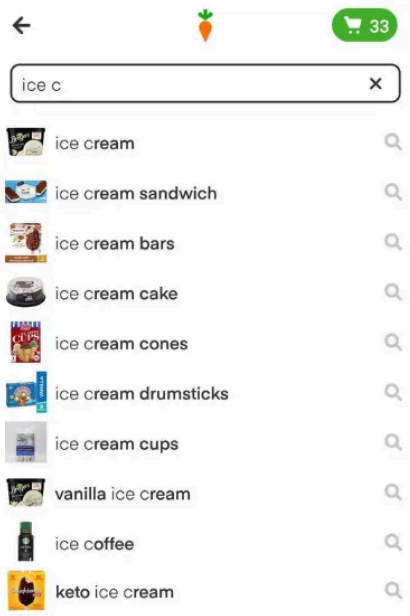
- Removing suggestions with bad words and blocklisted terms
- Removing suggestions with words out of an Instacart-defined vocabulary that is specific to shopping intent on our platform
- Removing suggestions that are too long
- Remove plural duplicates (we don't want to show "egg" and "eggs", so we get the most popular version)

We ensure that our dataset includes a good representation of queries that are popular, trending, seasonally relevant and most importantly — produce highly relevant results if the user selects the suggested query.

Matching user's intent

Given the user input in the search bar, which we refer as **prefix**, we need to retrieve all query suggestions that match the prefix.

For example, for a prefix "ice c" we retrieve all candidates that contain "ice c" with the condition that the match needs to happen at the start of any word in the candidate set to allow for a greater and more diverse set. In this example, then, we would retrieve the following list: "**ice c**ream", "**ice c**ream sandwich", "**ice c**offee", "**ice c**ream bars", "vanilla **ice c**ream", "keto **ice c**ream", etc.



Autocomplete matching is restricted to string matching at the start of any word in the candidate set. In this example, “ice c” prefix will match “ice cream” but also “vanilla ice cream”

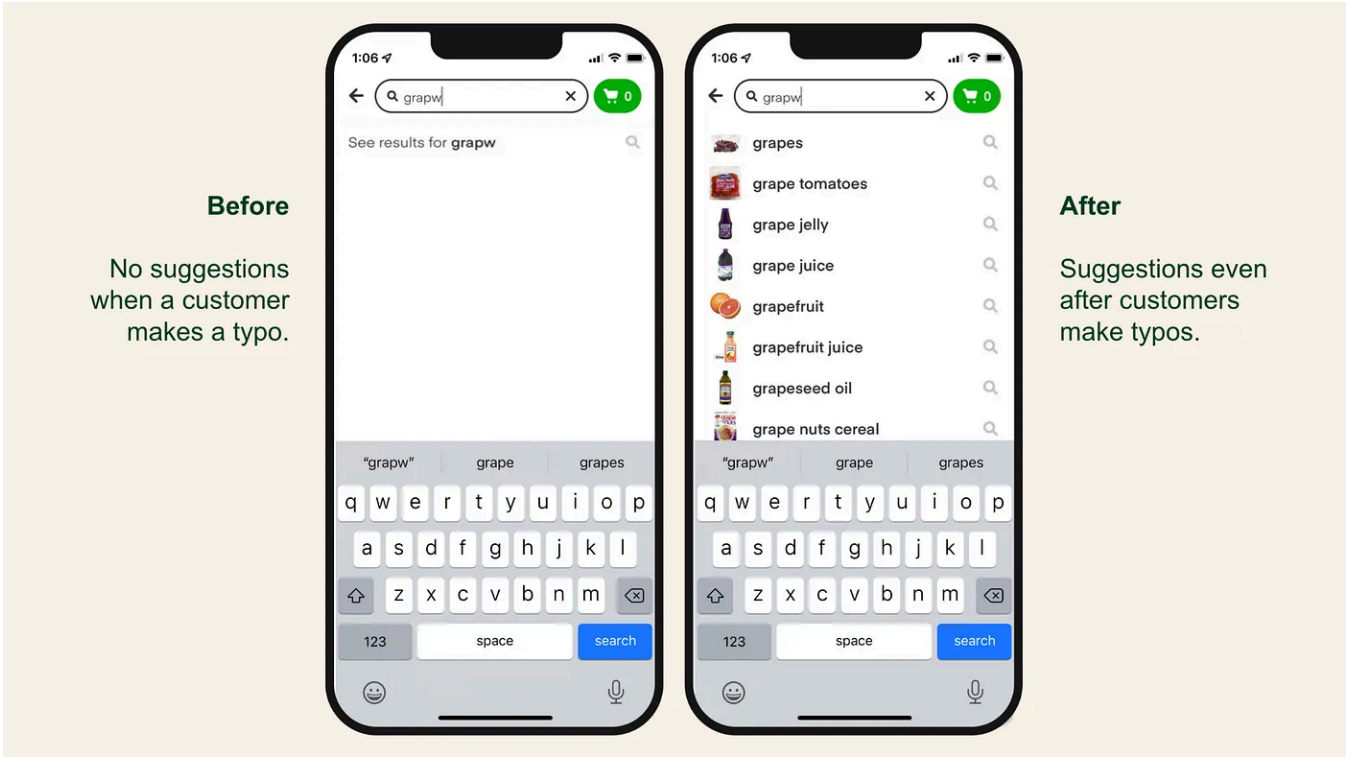
Below, we outline some challenges that we had to overcome to produce high quality suggestions.

Handling Misspellings

Despite imposing some quality measures to our Autocomplete Corpus, such as a filtering non-converted queries, there still are highly converting searches such as “avacado” with a high amount of traffic.

Simply removing misspelled suggestions in this case could cause a decrease in customer ordering avocados– which would be bad for both the customer, and for Instacart.

Enabling fuzzy matching with an appropriately tuned discount factor can help with surfacing the right term when the user’s input contains typos, given that “avocado” has an organically higher ranking score than the misspelled term “avacado”. Fuzzy match increased autocomplete engagement rate by 1%, which in turn improved the percentage of converted queries by 0.5%.



In addition, we plan to apply our Search team’s Spell checker to autocomplete corpus to identify common misspellings while displaying the corrected term instead.

Semantic Deduplication

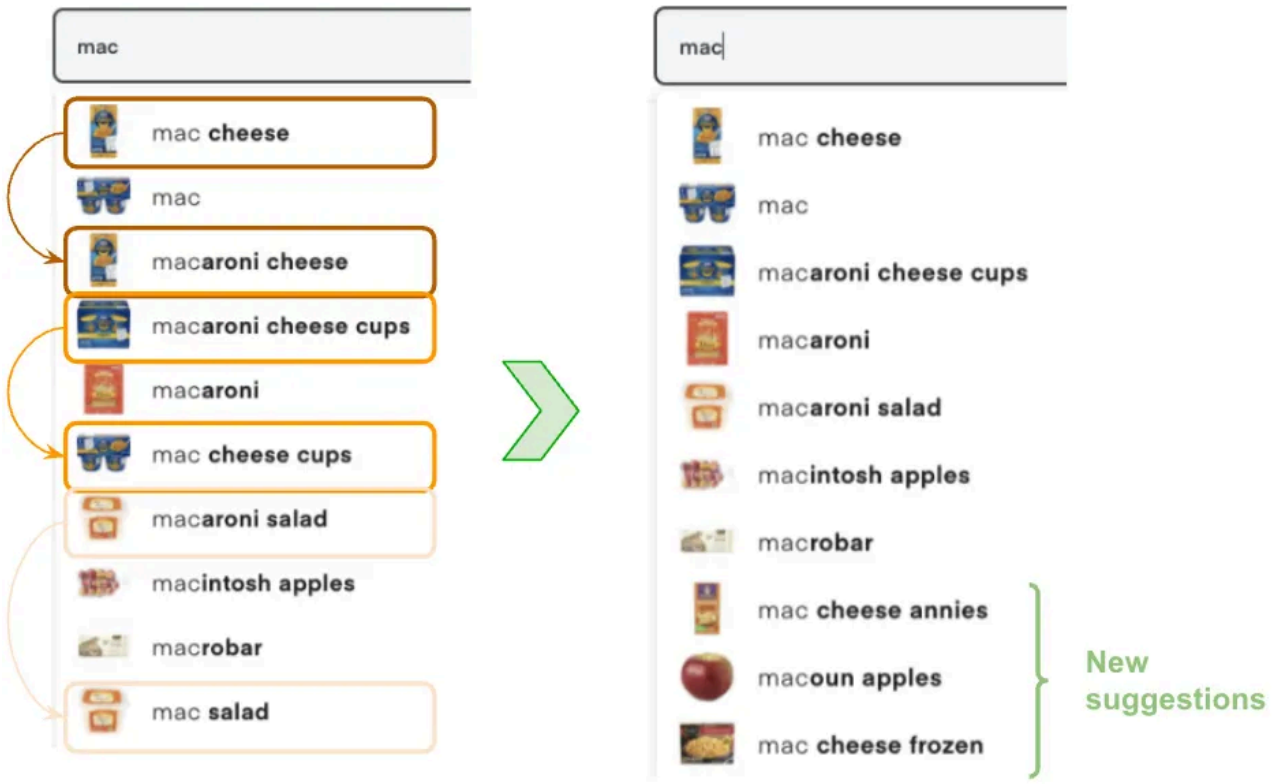
Search query terms generated from user logs will oftentimes include multiple autocomplete terms that are semantically identical. Sometimes, there is an almost exact word match (e.g. “fresh bananas” and “bananas fresh”, singular vs plural forms such as “apple” and “apples” or compound nouns such as “applesauce” vs “apple sauce”). Having semantically identical suggestions burdens the user with more unnecessary choices and decreases the likelihood of surfacing the user’s intent query. In addition, removing duplicates will result in a more diverse set of results.

We tackled semantic deduplication using our Search Embeddings model. Search embeddings are trained to define relevancy between a query and a product. The core of semantic deduplication algorithm is the use of pre-trained query embeddings to identify pairs of terms that are semantically similar by looking at the similarity score defined as the dot product of a pair of query embeddings.

Let’s look at the semantic similarity score between a couple search terms:

- Cheese slices → Sliced cheese: **98%**
- Mayo → Mayonnaise: **97%**
- Mac cheese → Macaroni cheese: **97%**

Via offline analysis and experimentation, we pick a query similarity threshold and apply it to mark query suggestions that have a semantic duplicate at a higher rank.



Leveraging Instacart Catalog to improve Cold Start problem

At Instacart, customers can buy products at multiple retailers, from their favorite nationwide chains to independent neighborhood grocery stores. Our basic algorithm generates a query suggestions dataset based on past searches done at each individual retailer. The reason for this is that the catalog and breadth of products can greatly vary amongst retailers — therefore using a common suggestion dataset for all retailers might produce irrelevant suggestions (suggesting “bananas” at Sephora).

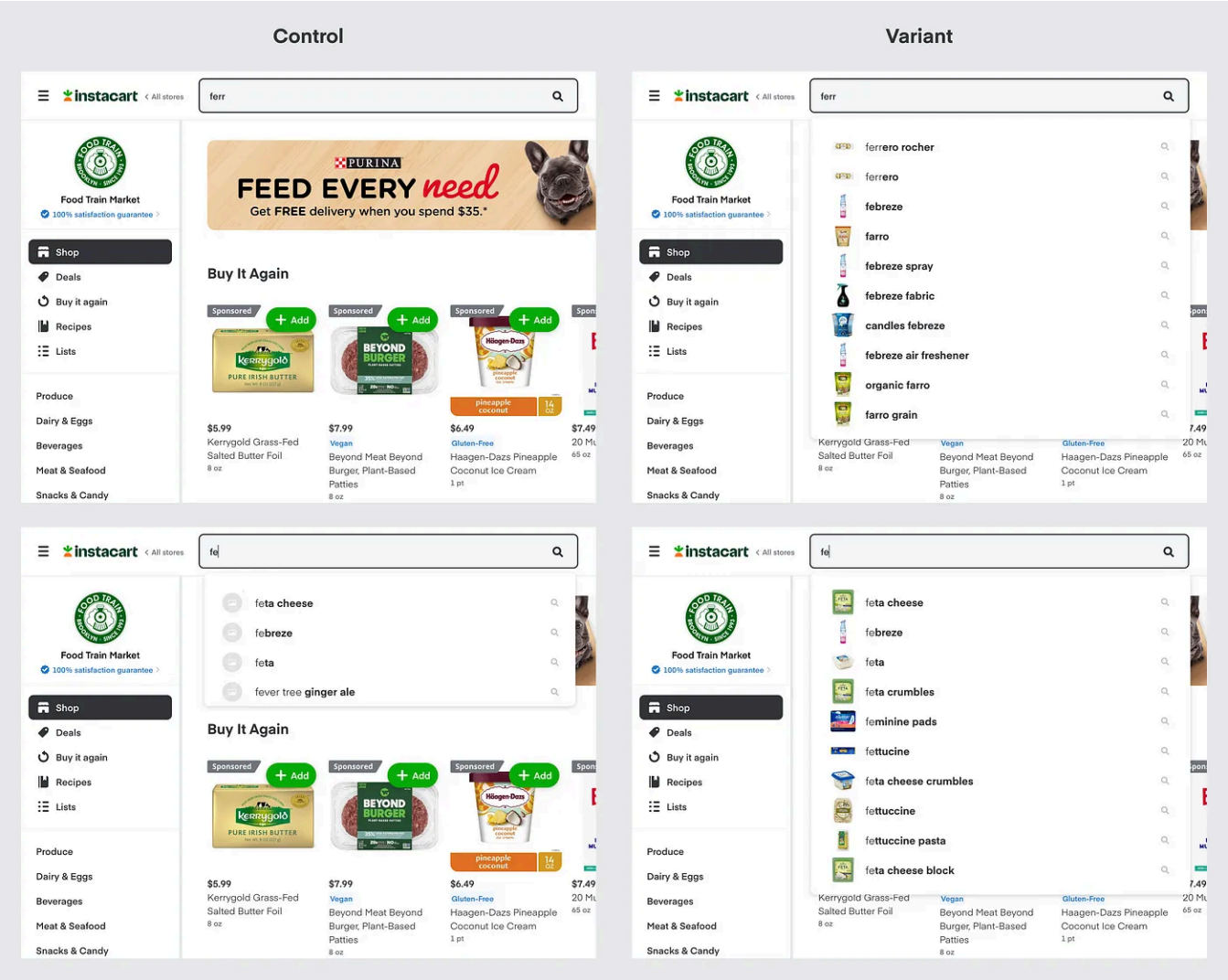
However, this means that we need enough traffic to occur at a retailer before we can generate useful query suggestions for them. This is especially an issue for new or smaller retailers, and generally a problem across all retailers for tail queries.

We overcame this problem by utilizing our standardized Instacart catalog that is shared amongst retailers. The common catalog allowed us to augment our understanding of search terms, products and produce query suggestions to fulfill new and smaller retailers with lower search traffic using products that are commonly shared amongst all retailers. This approach improved the autocomplete rate for New Users increased by 0.8%, as well as increasing basket sizes by 0.7%.

Our next step towards solving the Cold Start problem is to delve into a neural generative language model to extract query candidates terms from each product

available in the catalog (doc2query).

In addition to increasing the coverage of autocomplete terms, we followed a similar approach to increase coverage of thumbnails that are shown next to the suggestions.



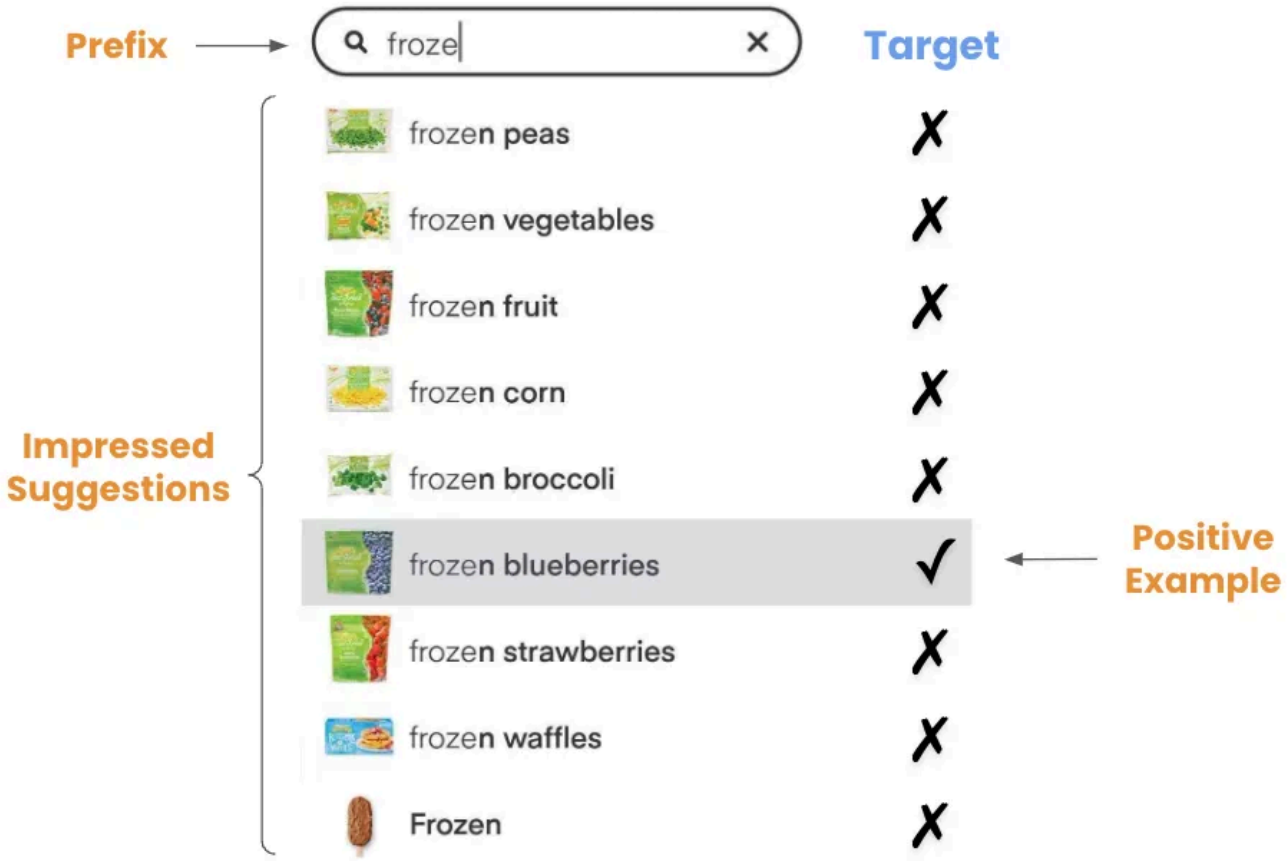
Ranking: Beyond Popularity

Our initial Autocomplete ranking model was based on popularity, which worked fairly well. However, as Autocomplete usage increased, we asked ourselves a lot of important questions — “Can it influence a user’s search behavior”, “Can it drive even more incremental basket sizes”, “Can it inspire users to search for novel queries”. This led us to explore new ranking models to determine the final ordering of suggestions.

Autocomplete Engagement Model

We started with a model whose objective was to surface the most relevant suggestions at the top. Using Autocomplete engagement logs as our training dataset, query suggestions that a user tapped on are used as positive examples, and the

remaining suggestions are used as negative examples. Using this dataset we can now train a learning-to-rank model considering a rich set of signals beyond popularity.



Example of one AC impression dataset. Customer has typed “froze” and clicked “frozen blueberries” (positive example) at display position 6. Other impressed suggestions are marked as negative examples.

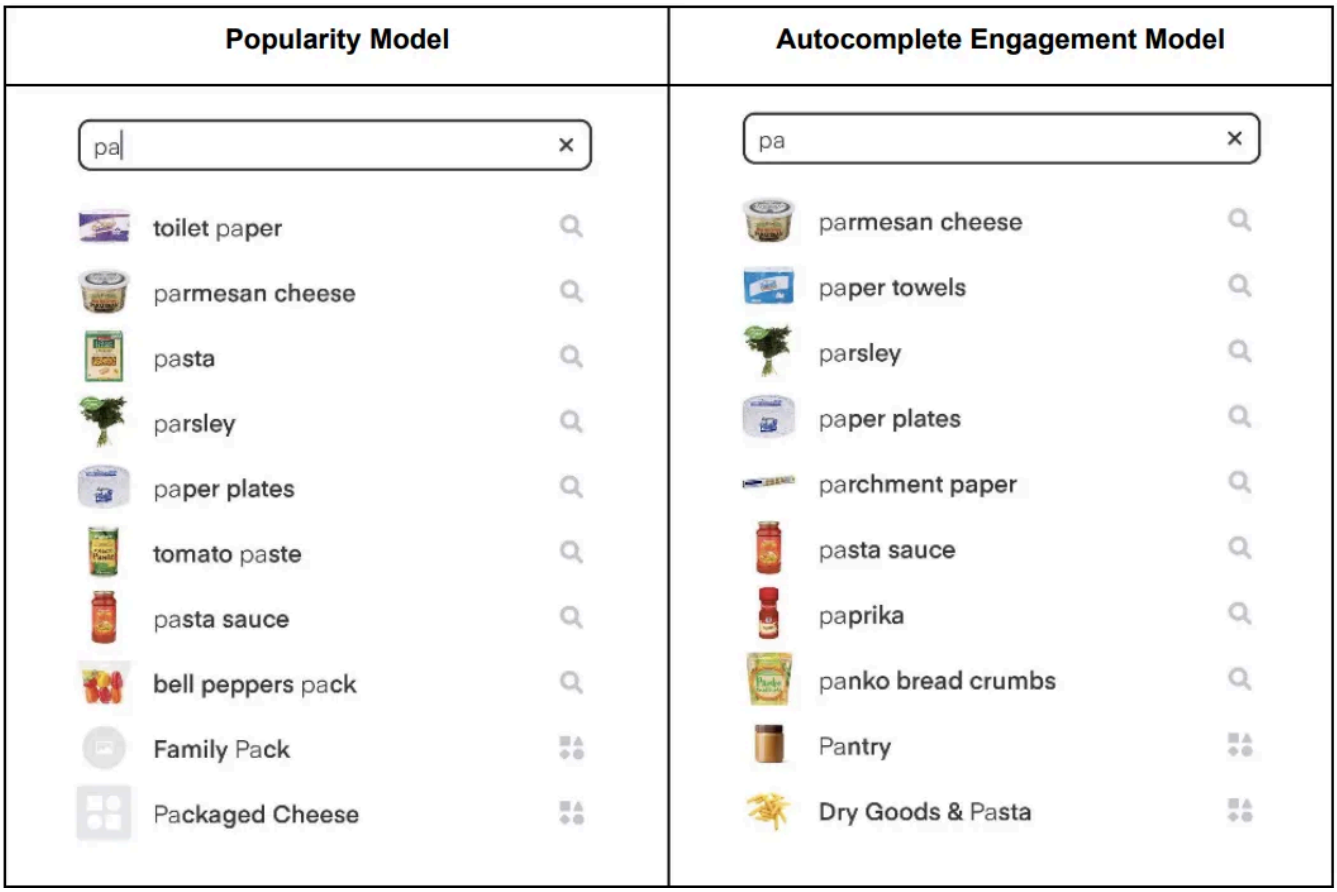
With strict latency being top of mind, we first developed a lightweight ranking algorithm modeled as a binary classification problem with a blend of query features such as popularity and prefix-query interaction features.

The most relevant features are:

- **ac_conversion_rate**, measures the rate at which suggested terms are observed and clicked given a prefix.
- **ac_skip_rate** aims to control for position bias by modeling the rate at which top suggested terms are seen but not engaged by the user.
- **is_start_match**, a boolean feature to indicate whether the prefix matches from the start of the suggested term (i.e. “**p**aper towels” matches prefix “**p**” from the start, but “toilet **p**aper” doesn’t).
- **is_fuzzy_match**, a boolean feature to indicate whether the match is fuzzy.

- **has_thumbnail**, a boolean feature to indicate whether the suggestion contains a thumbnail.
- **normalized_popularity**, a normalized query popularity feature (as a fraction of total searches), at the retailer level.

This model led to an increase in autocomplete engagement rate, and a decrease in the average prefix length, helping customers get to search results faster by typing less.



Showing results for “pa” under Popularity and Autocomplete Engagement models. Note that “toilet paper” is a very popular search query, and therefore ranking on the first position even if a fairly unlikely query for “pa”. An improved experience can be observed in the Autocomplete Engagement model, which uses prefix-query features.

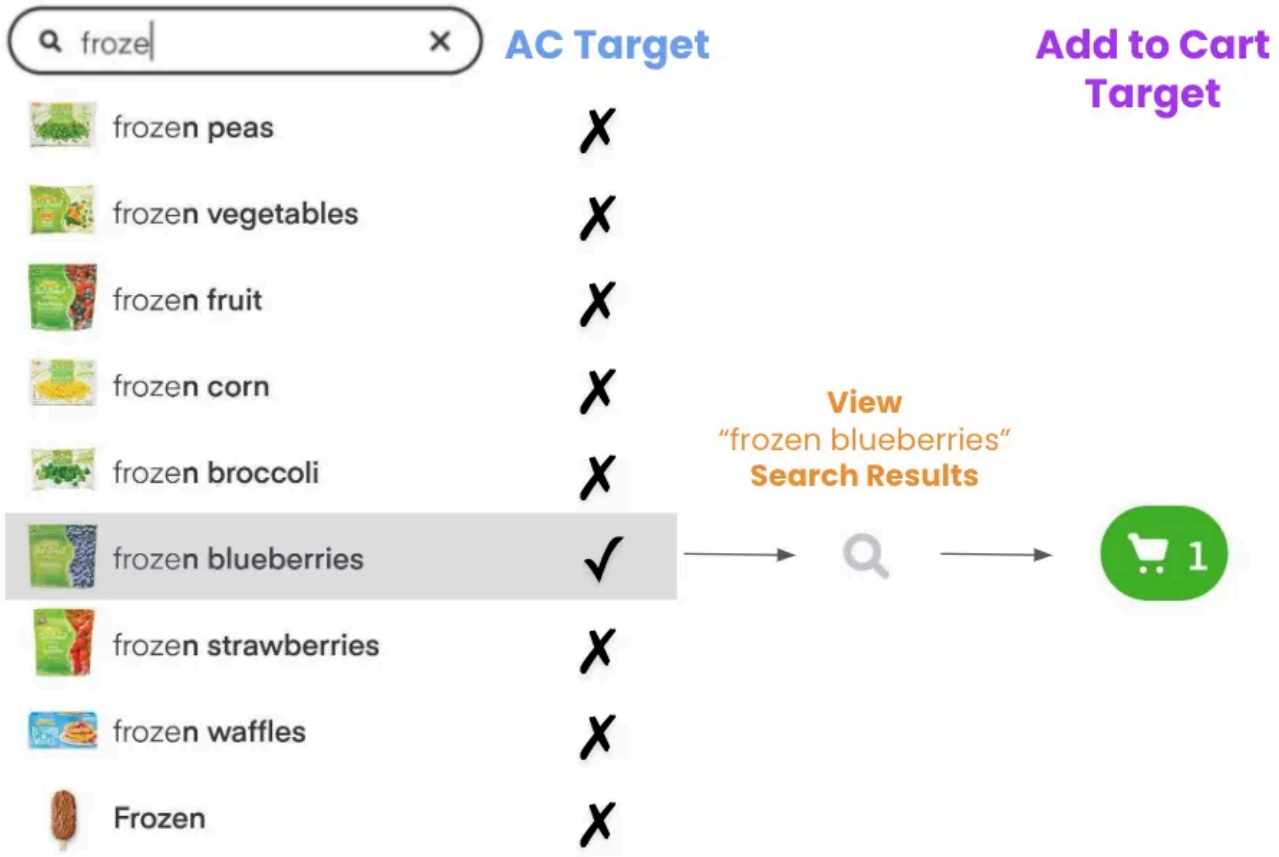
The effectiveness of this improved shopping experience trickled down to downstream core metrics with +2% in items added to the cart per search and +1.6% in Ad revenue.

Multi-objective Ranking

We showed that Autocomplete is a great lever to guide customers to a better search experience and improve overall search performance. We can further drive business value by jointly optimizing for autocomplete engagement (help users find what they

are looking for faster) and add-to-carts (drive search traffic towards higher-converting queries).

The goal of Search is to fulfill an intent that the user expresses with a search query. We understand that an intent has been fulfilled when there has been one (or more) items added to a cart associated with that search query. In an oversimplification of the Search journey, there are three steps between expressing an intent and an add to cart. In the first step, the user is typing their intent in the search bar (prefix) and dynamically sees a ranked list of suggestions. If the user selects one of the suggestions (step 2), we have an *Autocomplete conversion*. In step 3, the user lands on the search results page showing suggested items for the selected term, and has now the choice to add an item to the cart — we call this event a *search conversion* or *add to cart*.



In an Autocomplete engagement model, selected suggestions are marked as positive examples, whilst all other suggestions are used as negative examples (AC target). Selected Autocomplete suggestions can then have zero, one or more add to carts (Add to Cart target).

In the Autocomplete engagement model, we are only looking at steps 1 and 2 and training a ranking model that optimizes for Autocomplete conversions. Instacart's ultimate goal in Search is to help the customer build a cart. For this reason, we

created a new training dataset that focuses on all three steps and built a multi-objective model with the ultimate goal of increasing add to carts.

We tried a few multi-objective approaches such as a **label aggregation** approach, where the problem is reduced to a single objective by aggregating multiple targets into a single one. In our autocomplete scenario, an add to cart can only happen if an autocomplete impression is converted and therefore we can simply swap the AC target to the Add to Cart target. However, we ran into data sparsity problems and opted for a **model aggregation** approach — a fusion of models tuned independently for each objective.

With the assumption that steps 2 and 3 are independent, we can estimate the probability of add to cart given a prefix as a nested product of an AC engagement model (described in previous section) and an Add to Cart model, written as:

$$\underbrace{p(atc=1|prefix)}_{\text{add to cart score}} = \underbrace{p(term|prefix)}_{\text{AC Engagement model}} * \underbrace{p(atc=1|term,prefix)}_{\text{Add to Cart model}}$$

$$where \quad p(atc=1|term,prefix) \simeq p(atc=1|term)$$

Multi-objective ranking via model aggregation — a fusion of models tuned independently for each objective. Here, we have a AC Engagement model and an Add to Cart model, trained separately and combined for final ranking.

In the above equation, we actually make the assumption that the add to cart model is independent of the prefix (conditional independence), and therefore estimate the add to cart model as p(atc=1 | term), the conditional probability of an add to cart given a term. This has the added advantage of being able to use our entire Search logs for feature extraction, and not just those searches that were triggered via autocomplete.

Note that we treat the Add to Cart model as a binary classification task in this formulation, but that more than one add to cart can occur given a search term. After applying feature selection, features in this model include historical features as measured in the past N days:

- **Search Conversion Rate**, the total number of searches that have at least one conversion divided by the total number of searches of that term.
- **Add to Cart Rate**, the total number of add to carts divided by the total number of searches of that term.
- **Zero Result Rate / Low Result Rate,** the rate at which a search term yields zero or a low number of results.

These features are computed at the retailer level, but global versions have also been introduced to boost signal for tail queries and small retailers.

The model was launched after observing a +2.7% increase in Autocomplete engagement rate, +0.3% search conversion rate and an increase in GTV (Gross Transaction Value) per user as a result of driving more search traffic to high quality searches.

Parting thoughts

Autocomplete is more than a navigation tool that anticipates the user's intents as they are typing. It is also a powerful avenue to shape search traffic towards meaningful queries and increase discoverability of Instacart's product offerings.

To best guide and inspire the user through the process of building a cart, Autocomplete needs to take into account a variety of factors beyond popularity and search metrics. We are working towards making our Autocomplete personalized and utilize contextual information such as local trending products and a user's current items in the cart to help navigate and delight the customer with relevant and inspiring suggestions.

Shoutouts!

Nitin Pasari, Jesse Shevin, Silas Burton.

Machine Learning | Search | Autocomplete | Multi Objective | Instacart

Follow

Published in tech-at-instacart

6.9K Followers · Last published Nov 8, 2024

Instacart Engineering

Follow

Written by Esther Vasiete

105 Followers · 8 Following

Machine Learning @Instacart

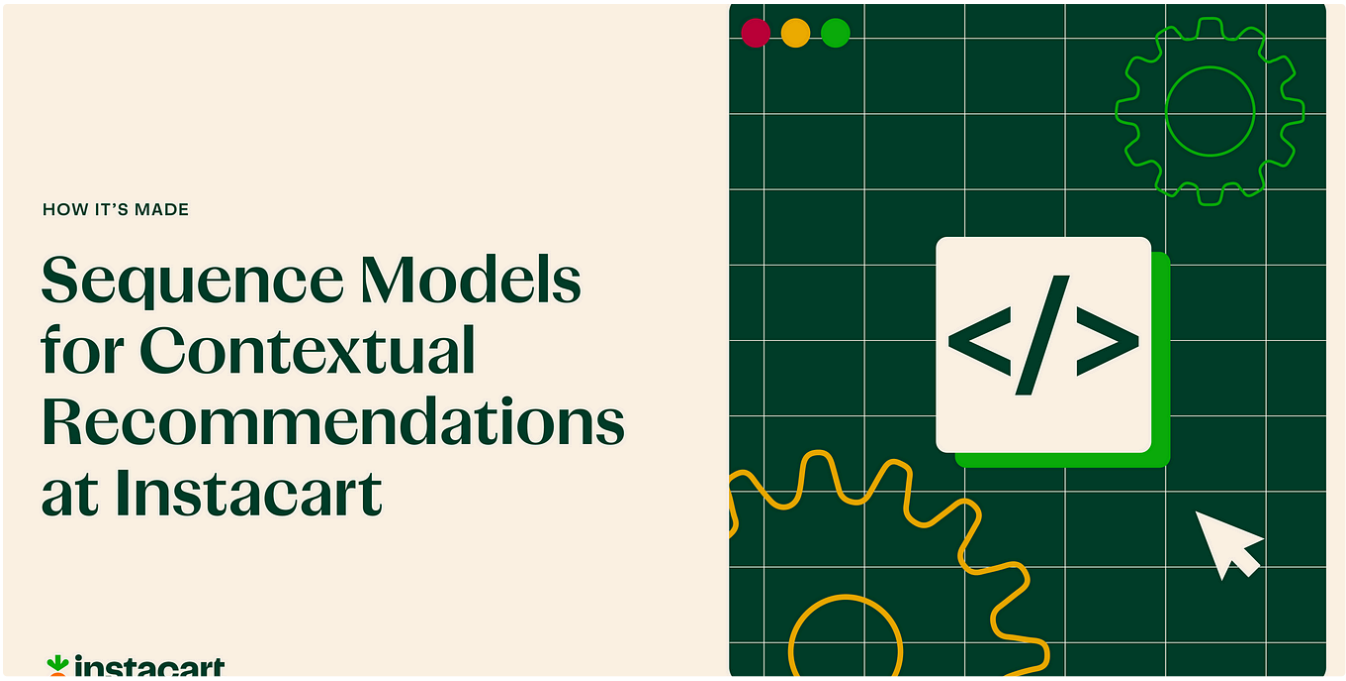
Responses (7)

navneet chaudhary

What are your thoughts?

See all responses

More from Esther Vasiete and tech-at-instacart

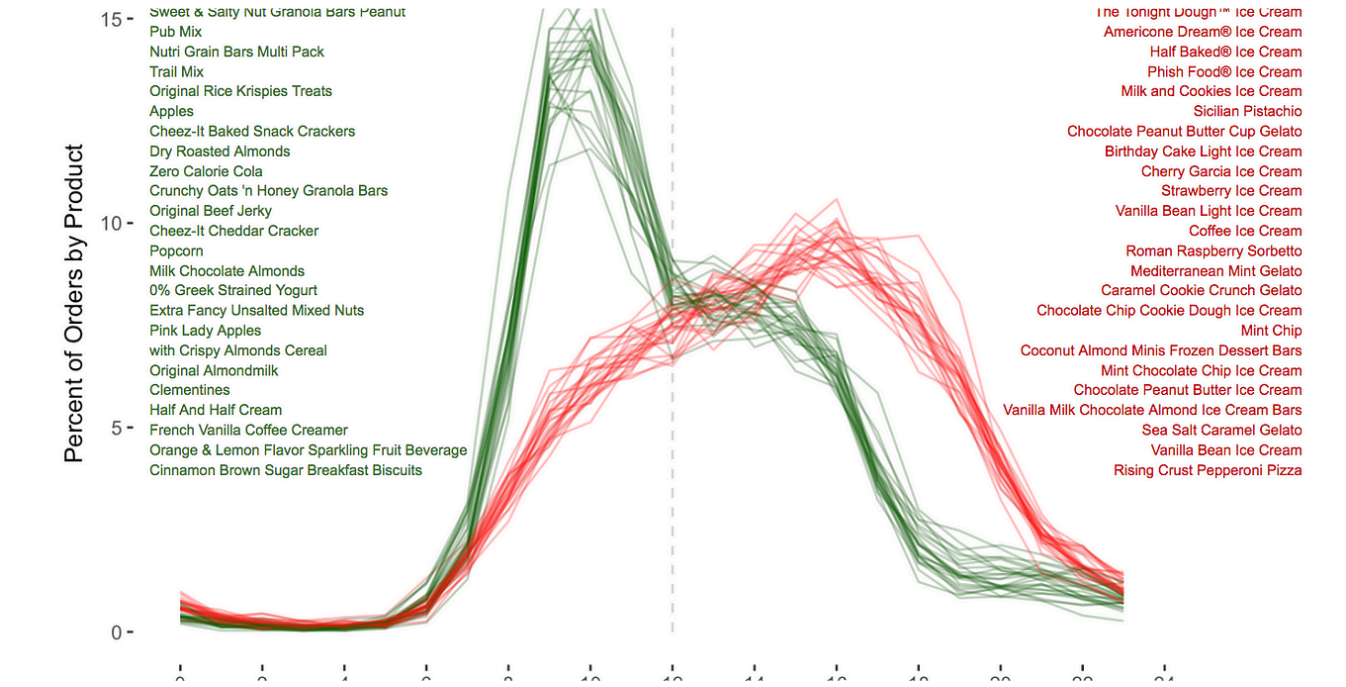


In tech-at-instacart by pradeep karuturi

Sequence models for Contextual Recommendations at Instacart

Authors: Pradeep Karuturi, Young Rao, Sharath Rao, Shishir Kumar Prasad Key contributors: Brian Lin, Cheng Jia, Karuna Ahuja, Shrikar...

Oct 23, 2024 261 1



In tech-at-instacart by Jeremy Stanley

3 Million Instacart Orders, Open Sourced

Curious about the food Americans eat? Look no further.

May 3, 2017 2.2K 36

In tech-at-instacart by Vinesh Gudla

Optimizing search relevance at Instacart using hybrid retrieval

Vinesh Gudla, Prakash Putta, Ankit Mittal, Andrew Tanner, Tejaswi Tenneti

Sep 11, 2024 505 3

In tech-at-instacart by Nick Shieh

Real-time Fraud Detection with Yoda and ClickHouse

Authors: Nick Shieh, Shen Zhu, Xiaobing Xia

Mar 18, 2024 94 2

See all from Esther Vasiete

See all from tech-at-instacart

Recommended from Medium

In tech-at-instacart by Cheng Jia

One model to serve them all

How Instacart deployed a single Deep Learning pCTR model for multiple surfaces with improved operations and performance along the way

Dec 20, 2023 268

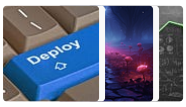
In The Airbnb Tech Blog by Hongwei Harvey Li

Wisdom of Unstructured Data: Building Airbnb's Listing Knowledge from Big Text Data

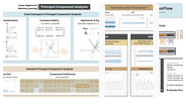
How Airbnb leverages ML/NLP to extract useful information about listings from unstructured text data to power personalized experiences for...

Nov 15, 2023 362 10

Lists



Predictive Modeling w/ Python
20 stories · 1852 saves



Practical Guides to Machine Learning
10 stories · 2221 saves

Natural Language Processing
1973 stories · 1616 saves

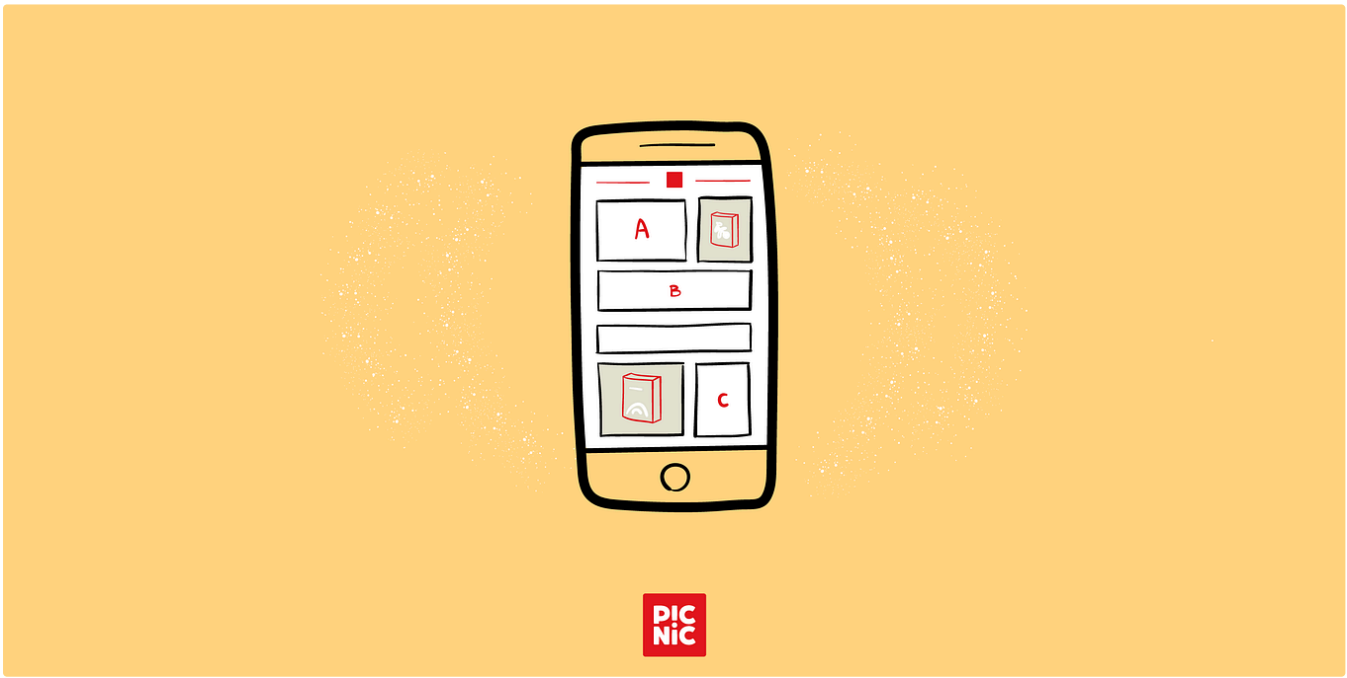
The New Chatbots: ChatGPT, Bard, and Beyond
12 stories · 562 saves

In Coupang Engineering Blog by Coupang Engineering

Accelerating Coupang's AI Journey with LLMs

By ML Platform Team

Nov 15, 2024 92

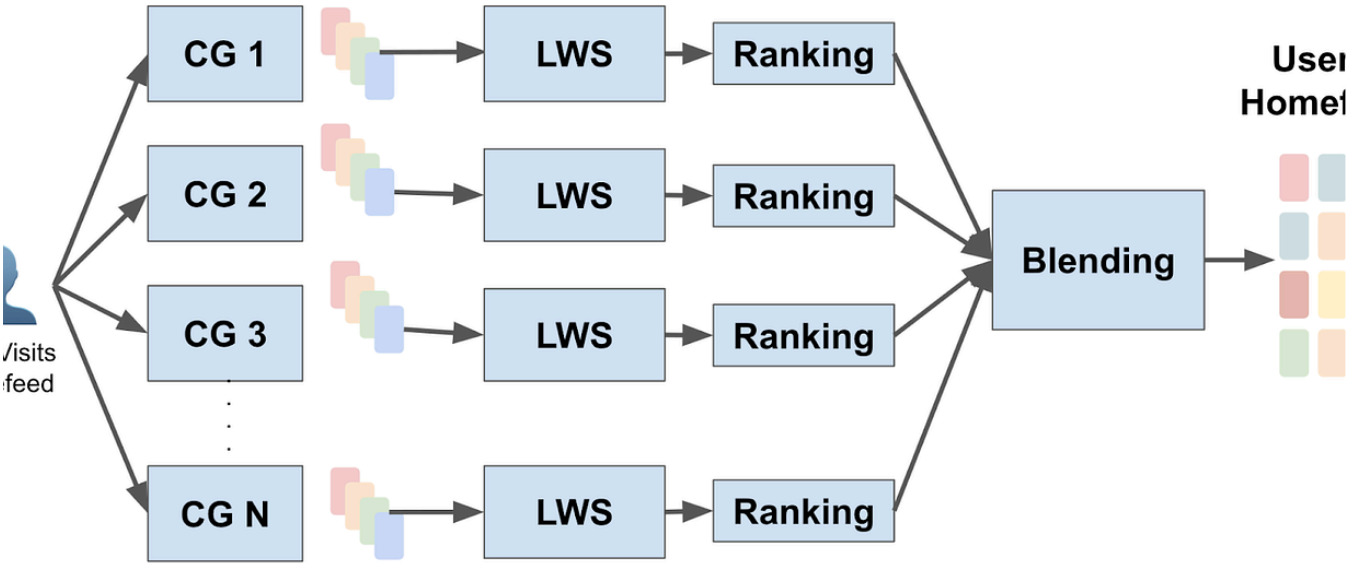


In Picnic Engineering by Lucas Twisk

Picnic's Page Platform from a Mobile perspective: enabling fast updates through server-driven UI

After introducing our Page Architecture initiative in this previous post, we'll now dive deeper into how we transformed the mobile app —...

Jan 30 42

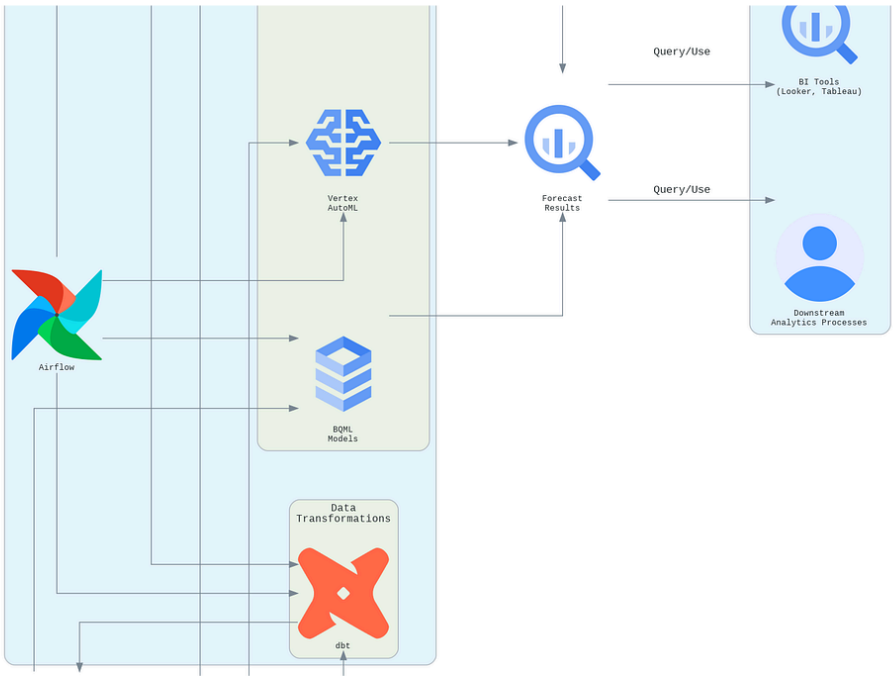


In Pinterest Engineering Blog by Pinterest Engineering

Establishing a Large Scale Learned Retrieval System at Pinterest

Bowen Deng | Machine Learning Engineer, Homefeed Candidate Generation; Zhibo Fan | Machine Learning Engineer, Homefeed Candidate...

Jan 31 207



In Just Eat Takeaway-tech by Stan Chen

Powering Global Forecasting at Just Eat Takeaway.com with BigQuery ML & Vertex AutoML

A Practical, Iterative Approach to Building a Reliable Forecasting Platform

Jan 27 17