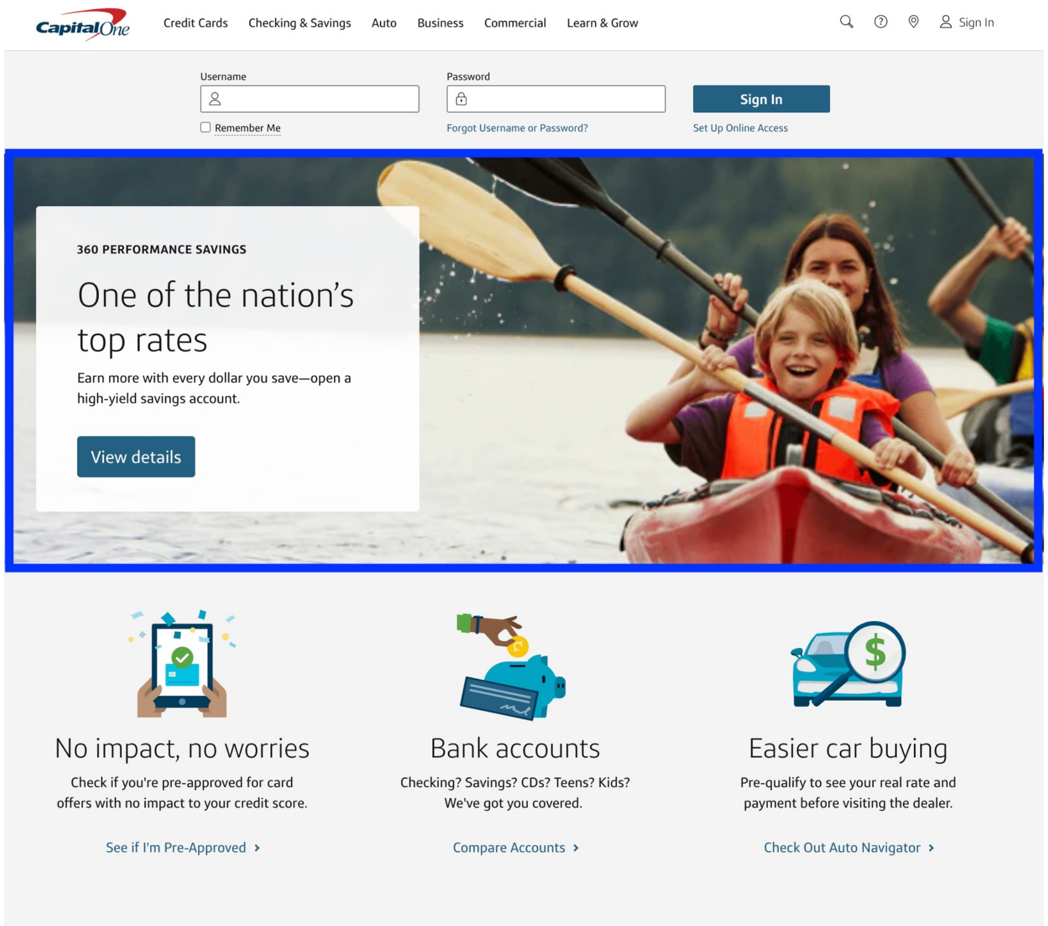
# A Transformer-Powered Recommendation Engine for Personalized Online Advertising

**Kalanand Mishra, Snehita Varma, and Jesse Zymet**
***Capital One Data Science***

September 21, 2022

# We govern personalized recommendation / advertising for the Capital One homepage

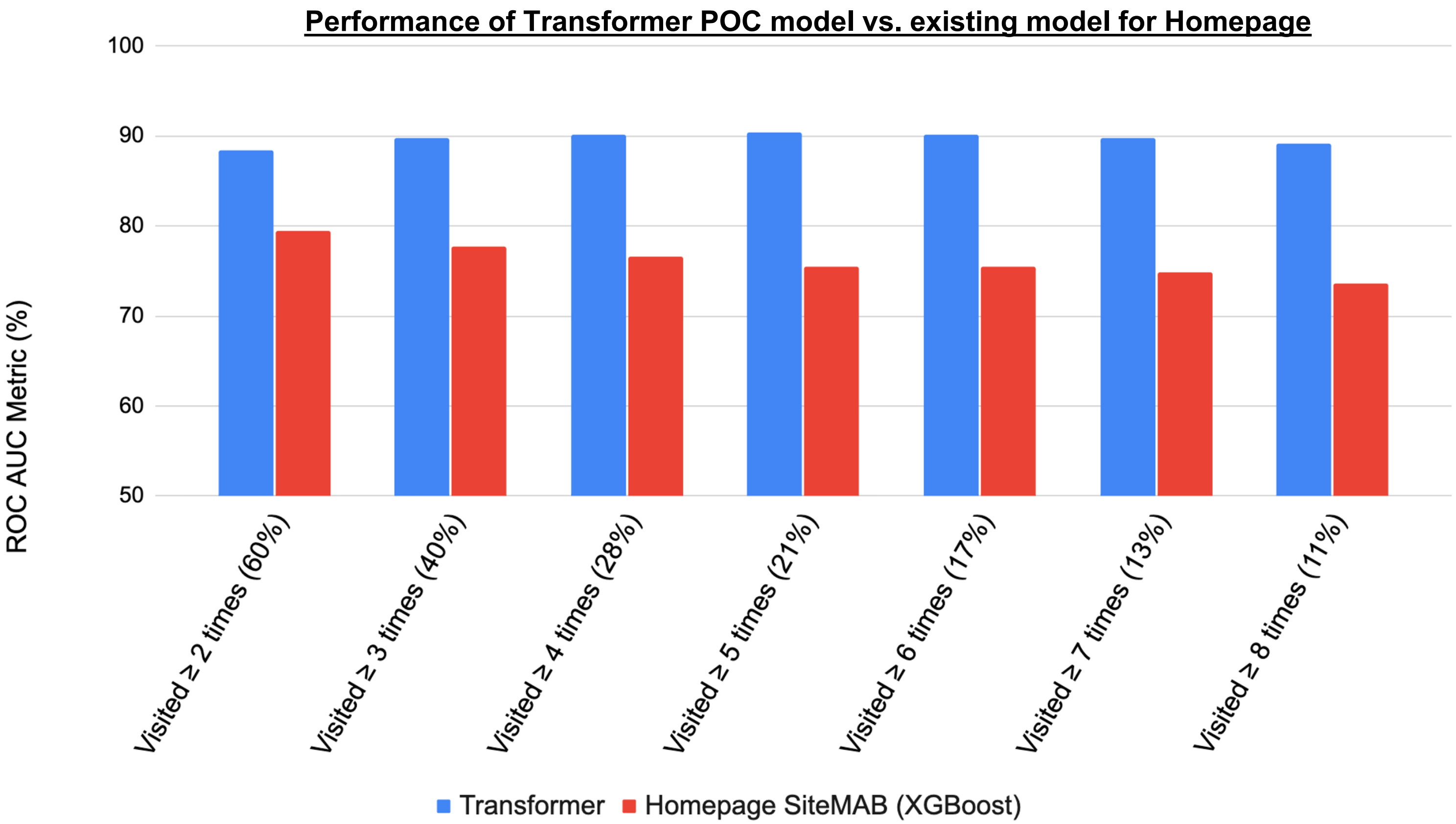



…

# Summary of Presentation

- To personalize the banner shown to a visitor, we employ a contextual multi-armed bandit powered by the XGBoost algorithm
  - System takes visitor contextual features as input, e.g. products they already own
- Algorithm cannot take into account *sequenced data on user experiences / behaviors*
  - E.g., fails to appropriately capture impression sequence leading up to conversion
- Today: POC that replaces XGBoost with the Transformer from Deep NLP
  - Uses self-attention to learn from sequenced data which experiences in a visitor’s journey were relevant to their outcome
- We recruit NVIDIA’s **Transformers4Rec software package** to achieve significant gains in prediction quality

# We employ NVIDIA’s Transformers4Rec library to show big recommendation improvements for repeat visitors to our homepage

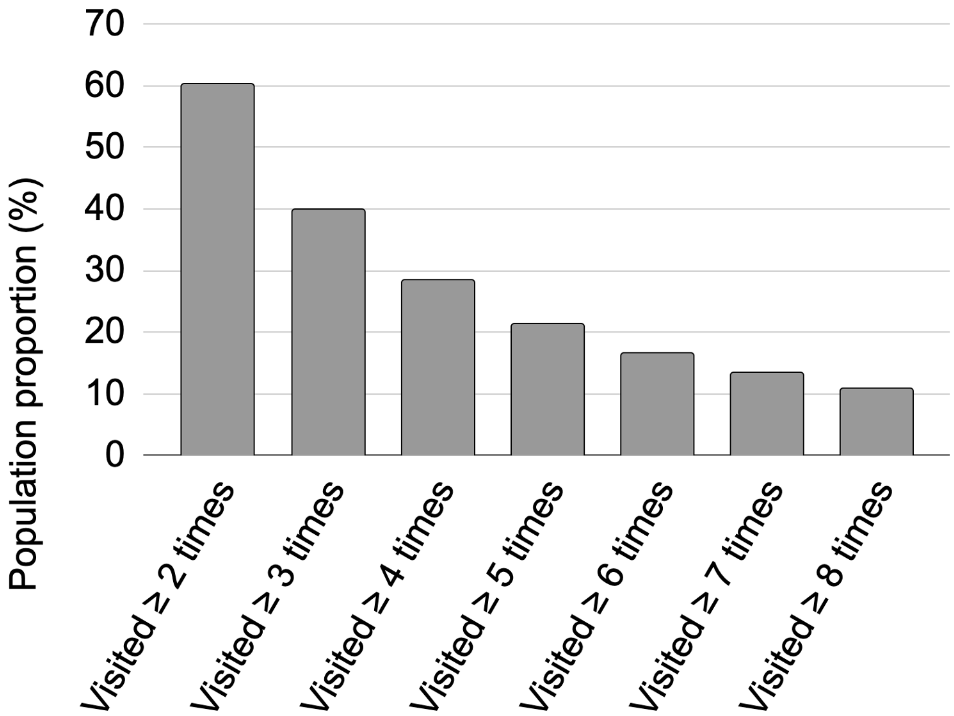


# Agenda

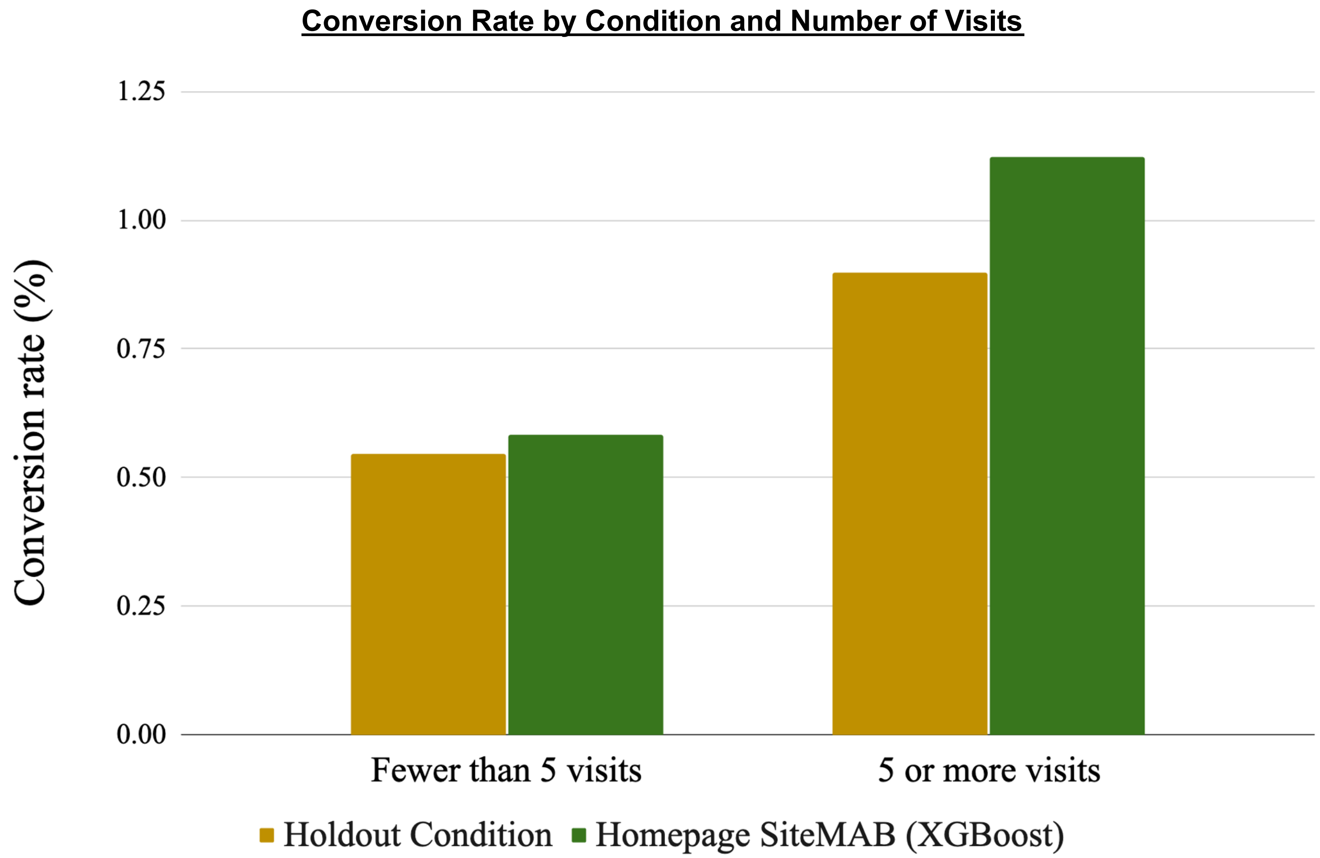
- **Description of the problem**
- **What is a Transformer? Rationale for it in Recommender Systems**
- **Site personalization POC**
- **Performance**
- **Conclusion and future work**

# Repeat visitors are 60% of the Capital One's homepage visitor population

**Proportion of Customers Visiting The Site Multiple Times per Month**



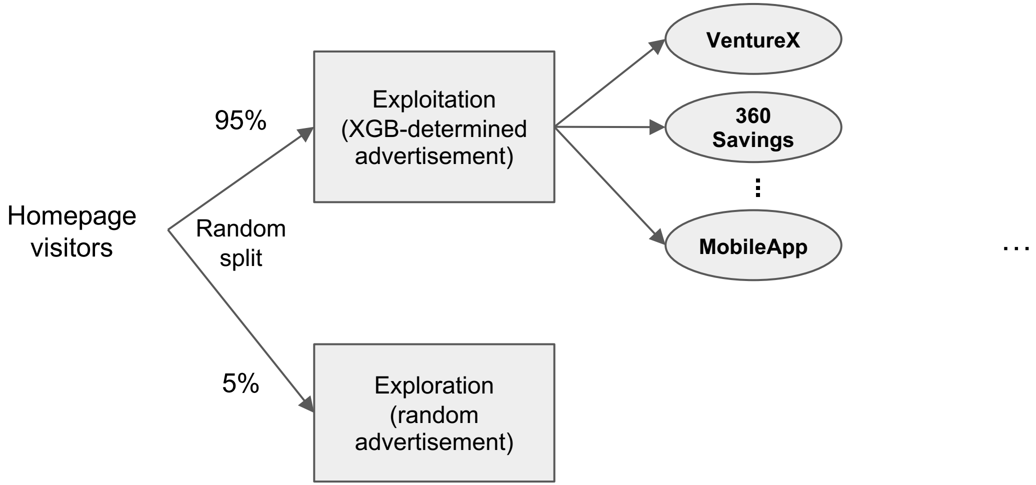
# Frequent visitors convert at higher rates and are major drivers of lifetime value



# Our recommendation engine is a contextual multi-armed bandit with exploitation component consisting of an XGBoost model

- Epsilon-greedy approach:
  - Random 5% of visitors are served random advertisement (exploration)
  - The rest are served a personalized experience powered by XGBoost (exploitation)

- Day's visitors react to served ads; model retrained the next morning with the new data




Capital One

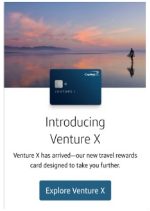
# Our XGBoost model takes into account our visitors' context, but does not capture visitor data as a sequence of experiences


| | | Input Space | | Response |
|---|---|---|---|---|
| **Date** | **Customer** | **Ad** | **Owns Card?** | **Convert** |
| 8/01/2022 | Jane | AutoNav | 1 | 0 |
| 8/02/2022 | Jane | Savings | 1 | 0 |
| 8/03/2022 | Jane | MobileApp | 1 | 0 |
| 8/04/2022 | Jane | VentureX | 1 | 1 |

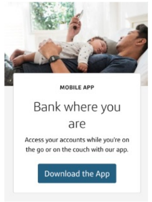
**Session-Level Data Structure**

Binary Classification (XGBoost)

Probability that Jane converts if she sees Banner 1 = 0.03




*Probability that Jane converts if she sees Banner 2* = 0.02




*… etc.*

…

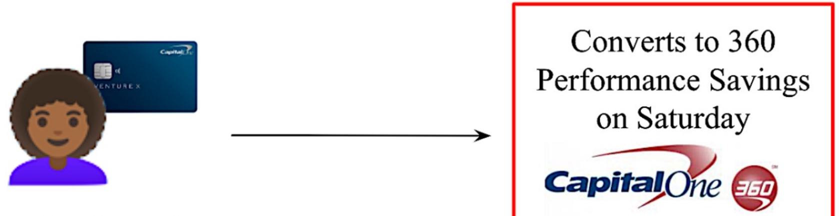
Ranked List of Banners

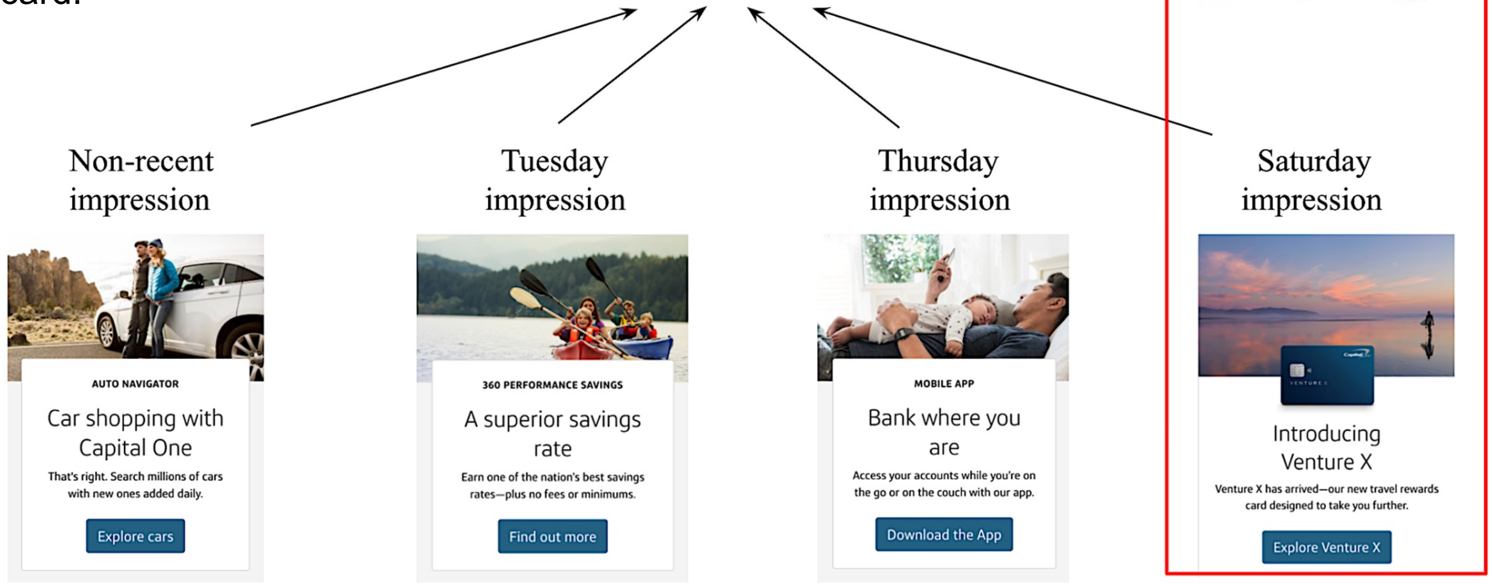
- Train model to predict conversion based on input configuration
- Score candidate ads based on trained parameters, pick highest scoring ad to show next

Capital One

# Our non-sequential model cannot learn which experiences in a visitor's history were relevant to conversion, limiting its ability to serve the best ads

A naive model like ours credits conversion to the Venture X impression, rather than the 360 Savings impression. Ends up learning to show Venture X Card more to people who own a Venture X card!

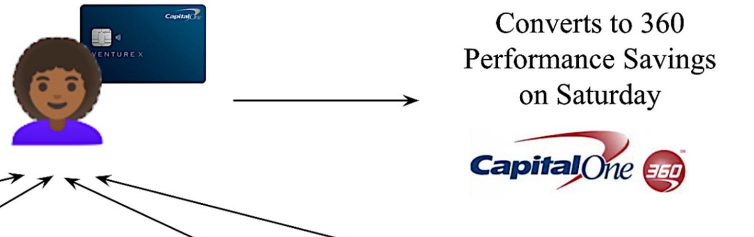


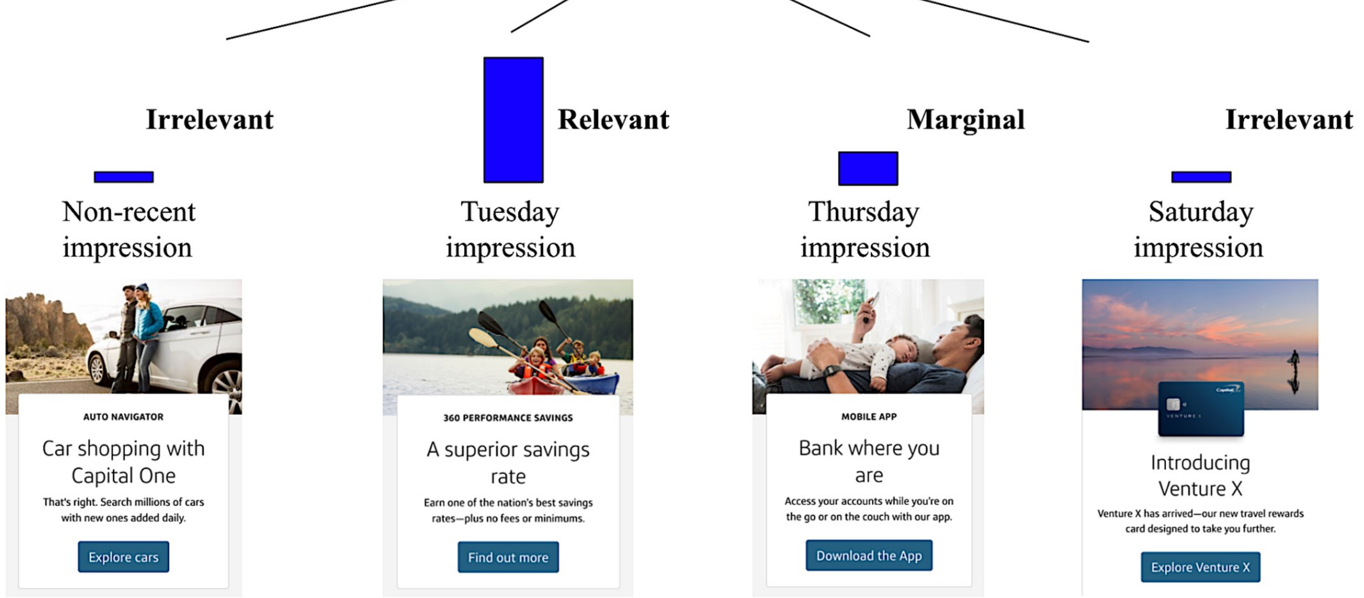



# Sequence modeling would optimize for this population and serve the right banner to the right individual more often, stimulating lift

We want an algorithm that:

- Keeps track of the temporal order of a visitor's history
- Learns how to credit their conversion to the appropriate prior experiences



# Agenda

- Description of the problem
- What is a Transformer? Rationale for it in Recommender Systems
- Site personalization POC
- Performance
- Conclusion and future work

# The Transformer is a deep learning model from NLP – it uses Attention to learn how relevant a word is to sentiment in sentences *with different contexts*

**Bakery Review** **Rating**

"The cake was so fluffy and tender that I loved it."

👍

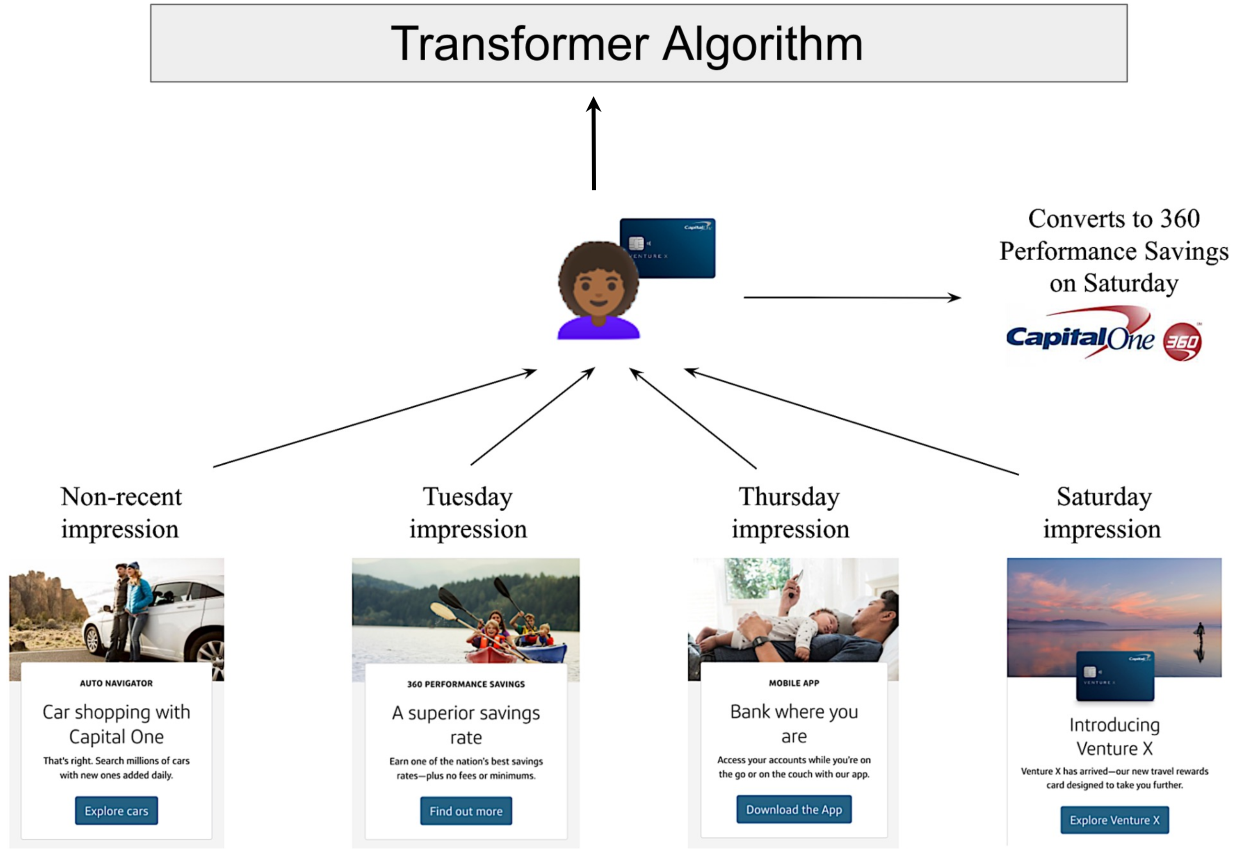
"The cake was so bittersweet that I hated it."

👎

"The cake was so fluffy and sweet that I just hated to love it!"

👍

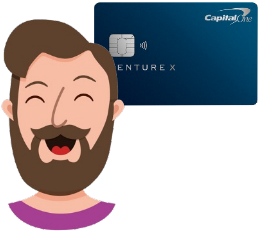
# The same model can be used for personalized recommendation – to learn from Jane's context *and her sequence of experiences leading up to conversion*



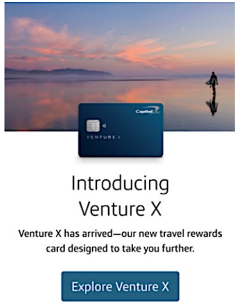
# The Transformer can learn to *pay attention* to the most relevant experiences from Jane's history, so as to serve better recommendations to visitors like her

Transformer Algorithm

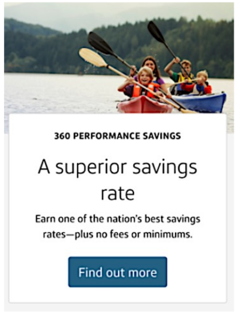
Converts to 360 Performance Savings on Saturday

*Probability Jim converts if he sees Venture X* = 1%




*Probability Jim converts if he sees 360 Savings* = 2%

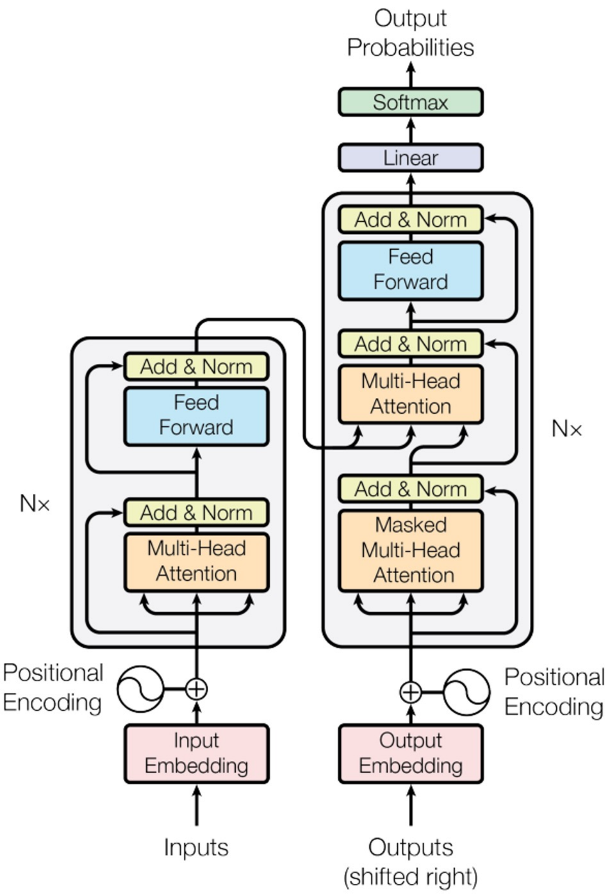


# We employ the Attention-powered Transformer to solve the attribution problem

- The **Transformer** ([Vaswani et al. 2017]) is a state-of-the-art algorithm from Deep Learning and NLP.
  - E.g., [Yang et al. 2019] on text classification

- Transformers have outperformed its competitors on various *recommendation tasks*:
  - Uses Attention to learn from data how to attribute visitor's conversion to appropriate past experiences
  - See e.g. [Sun et al. 2019], [Kang et al. 2018]

- Attention-powered systems have outperformed competitors on *attribution problems like ours*:
  - See esp. [Arava et al. 2018] on comparing RNNs w/ Attn against plain LSTM
  - Also [Ren et al. 2018], [Kumar et al. 2020], etc.




CapitalOne

# Brief overview of concepts underlying Transformers

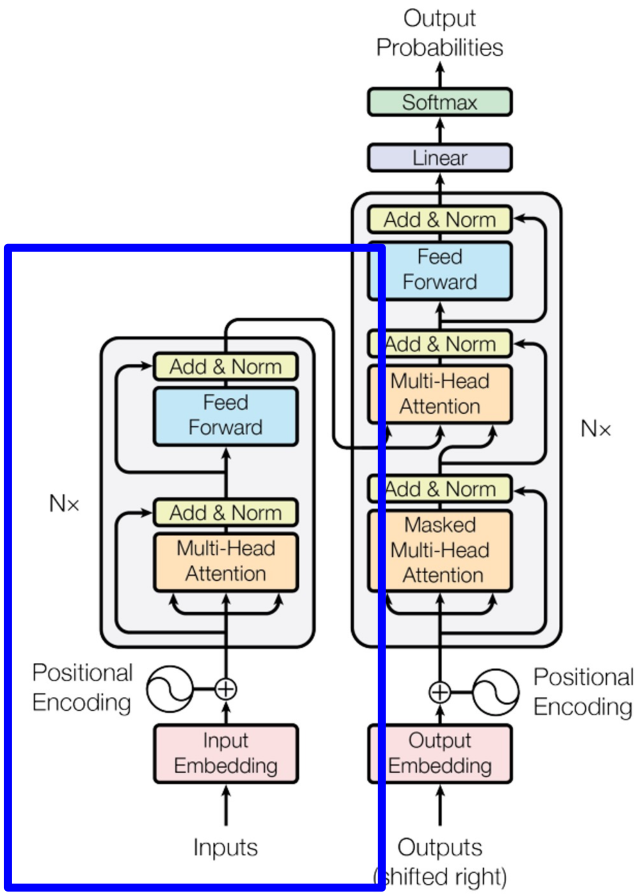
- Unlike RNNs, Transformers are non-sequential algo's:
  - Relative ordering between sequential timesteps captured with *positional encoding*

- Uses *Self-attention mechanism* to capture context / relevance of a timestep within broader sequence

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Self-attention applied iteratively in stack of Attention blocks to capture higher-order interactions

- Originally consisted of encoder & decoder — our model (ALBERT) is encoder-only

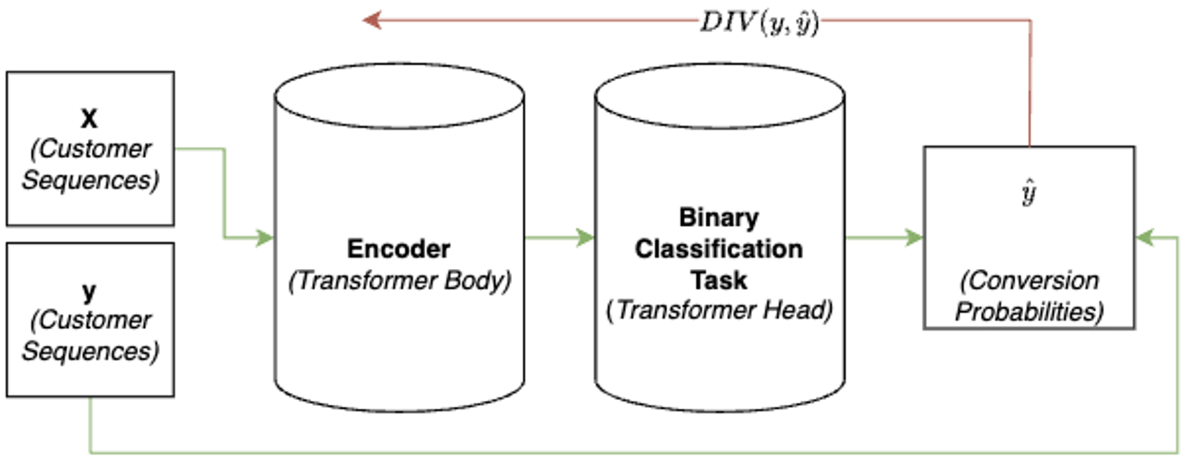



CapitalOne

[Our slides covering the basics of Transformers]

# Our supervised training process differs from how Transformers are typically trained for NLP use cases

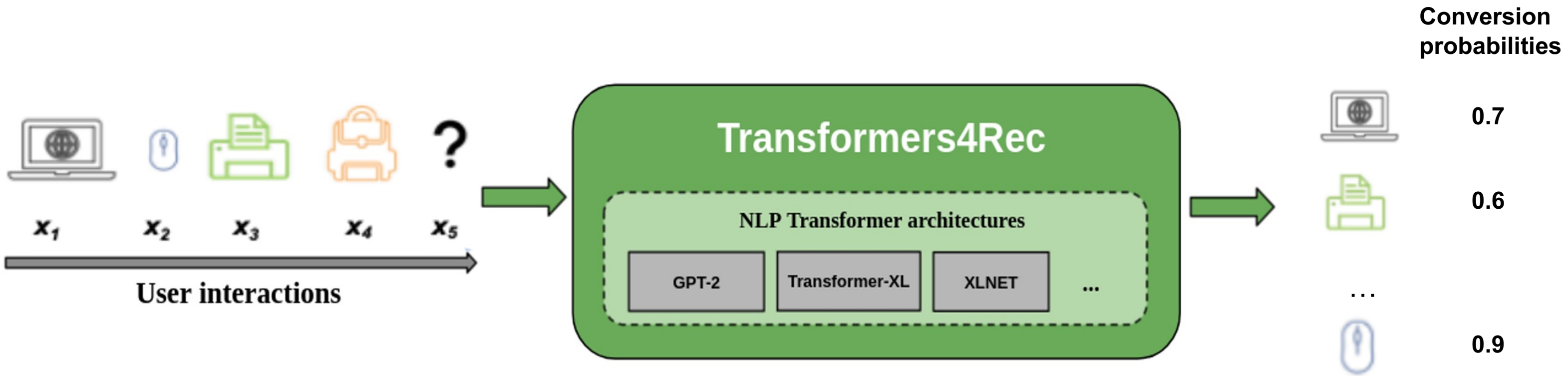
- Though Transformers typically have a pretraining component and masking module, our framework has neither
- Binary targets are explicitly passed in and modeled on directly
  - Transformer gets signal from divergence between true target and predicted label, rather than masked target and prediction over mask



# Agenda

- Description of the problem
- What is a Transformer? Rationale for it in Recommender Systems
- Site personalization POC
- Performance
- Conclusion and future work

# NVIDIA provides GPU-accelerated Deep Learning packages tailored for Recommender Systems




- We evaluated multiple packages - Transformers4Rec (NVIDIA), Recbole, Hugging Face
- Transformers4Rec provides the most out-of-the-box capabilities:
  - ✓ Provides a classification framework that allows us to predict probabilities of conversion based on input features — other packages only provide next-item prediction framework
  - ✓ Integrates user feedback and contextual features
  - ✓ Modular nature gives us flexibility to design Transformer architecture for our unique needs

# We built a Transformer-powered recommendation algorithm that employs NVIDIA's *Transformers4Rec* package

- Data: SiteMAB data for Homepage (June 30 - August 10, 2021)
- ~ 50 features:
  - Previous Site impressions
  - Product ownership
  - Login count
  - Device type (Mobile, Tablet, Mac, etc.)
  - Browser type (Firefox, Safari, Chrome, etc.)
  - Bank page visits
  - Card page visits

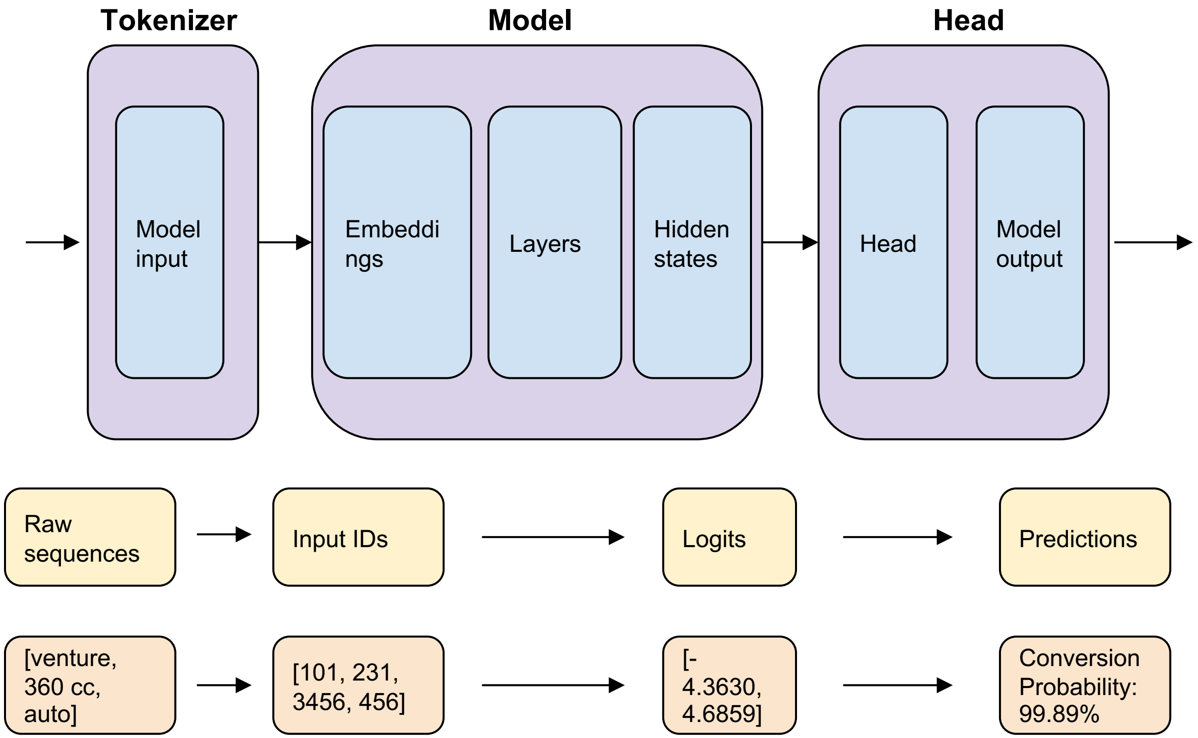



*Image adapted from Hugging Face Transformers Course*

# For data preprocessing we use NVIDIA's *NVTabular* to make use of GPU powered acceleration

- T4rec and NVTabular are part of NVIDIA's Merlin ecosystem that allows for end-to-end GPU-Accelerated and Distributed pipelines.
- NVTabular, as part of the NVIDIA Merlin ecosystem, is a wrapper over the RAPIDS Dask-CuDF library.
- Allows for GPU-Accelerated data operations like transformations, aggregations, slicing & padding.
- It takes us from a tabular representation, like the one we currently model on (SiteMAB), to a sequential one.

Input Space | Response

| Date | Customer | Banner | Day of Week | Convert |
|---|---|---|---|---|
| 8/01/2020 | 1 | 1 | 1 | 0 |
| 8/02/2020 | 1 | 2 | 2 | 0 |
| 8/03/2020 | 1 | 3 | 3 | 0 |
| 8/04/2020 | 1 | 4 | 4 | 1 |

**Session-Level Data Structure**

NVTabular

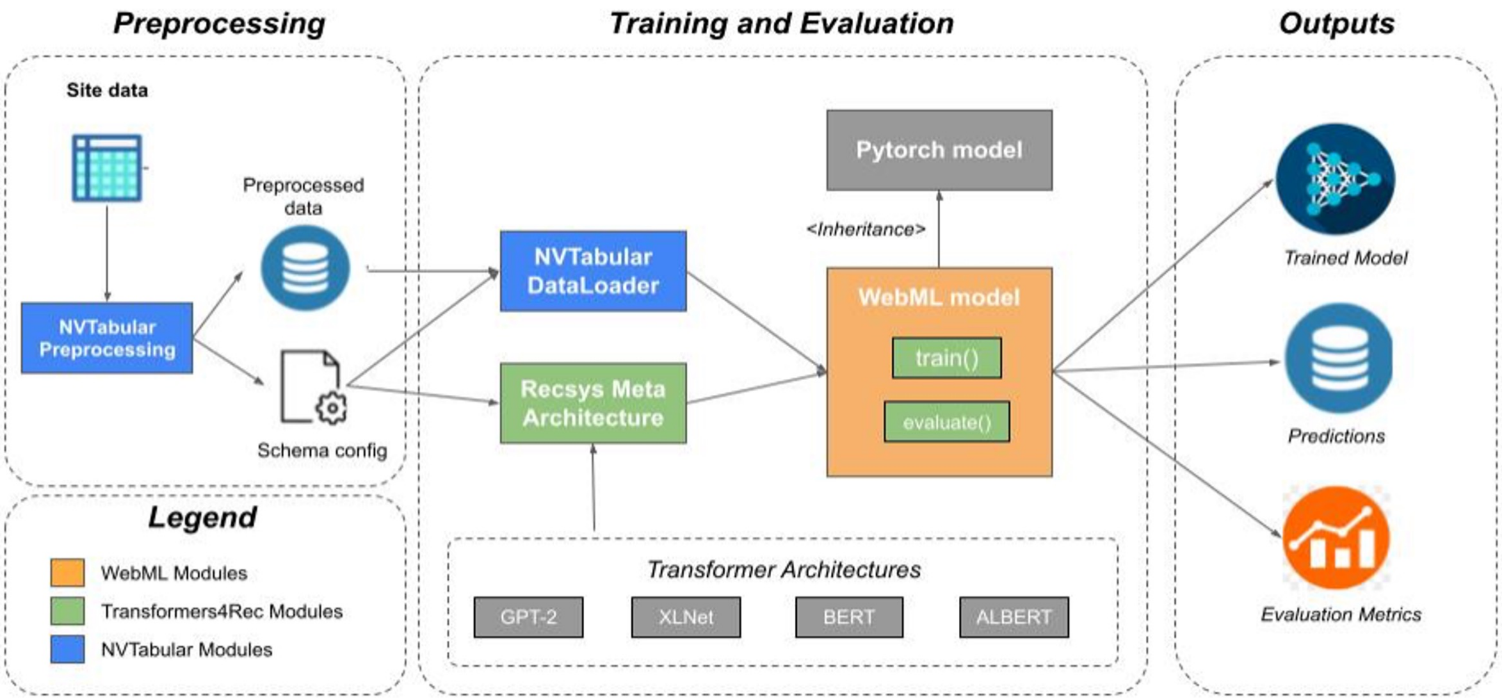
(GPU-Accelerated Data Operations)

Input Space | Response

| Latest Date | Customer | Banner Sequence | DayOfWeek Sequence | Convert |
|---|---|---|---|---|
| 8/04/2020 | 1 | <1,2,3,4> | <1,2,3,4> | 1 |

**Customer-Level Data Structure**

# Overview of our end-to-end Transformer POC pipeline




- POC uses Legoland instances with NVIDIA V100 GPUs with 256 GB memory, processing units that feature the Volta architecture

# Despite its size and complexity, the Transformer can be used for daily batch scoring or real-time scoring

| | Partition Type | Train Size | Test Size | Fitting Time | Evaluation Time | Scoring Time (12 variants - like SiteMAB) |
|---|---|---|---|---|---|---|
| **ALBERT**<br>16 heads<br>12 layers<br>Dimension of embedding = 64<br>Epochs per iteration = 20 | 80% train (downsampled* 50/50)<br><br>20% test (unsampled)<br><br>Partitioned by day | 100k customers | 2M customers | ~30 minutes<br><br>~3.8k customers per minute | ~5 minutes<br><br>~344k customers per minute | **~4.25 hours**** to score 50mm customers daily - 8 V100 Instance Distributed Inference<br><br>~200k customers per minute |

* Downsampling required for two reasons:

1. Fit is relatively slow otherwise (need multi-GPU)
2. Transformer struggles with class imbalance (i.e., the fact that only ~0.1% of customers convert)

** This includes operations to 'enrich' feature vectors to predict at t+1

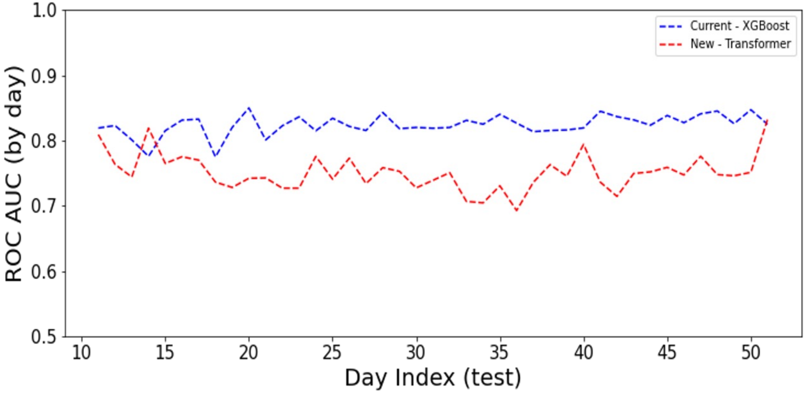
# Agenda

- Description of the problem
- What is a Transformer? Rationale for it in Recommender Systems
- Site personalization POC
- **Performance**
- Conclusion and future work

# In our POC, we tested the Transformer against our current model and measured performance across two segments of Site users

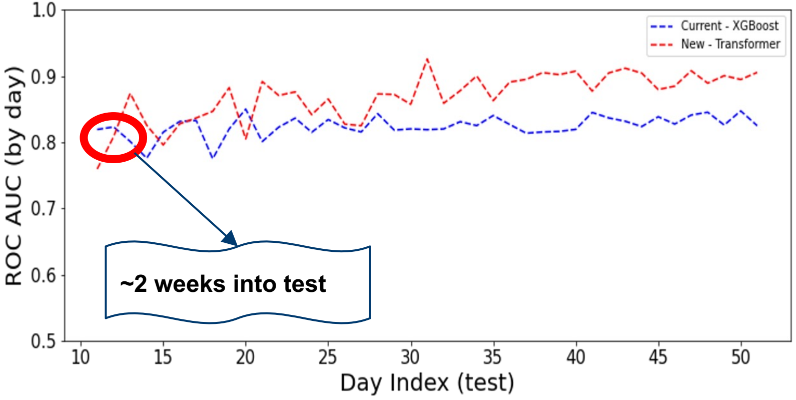
## Segment #1
**single-visit** users (40% of sample)

- **Transformer** underperforms existing model when a user lacks enough history to compose a sequence
- **Does not** exhibit trends of learning throughout time



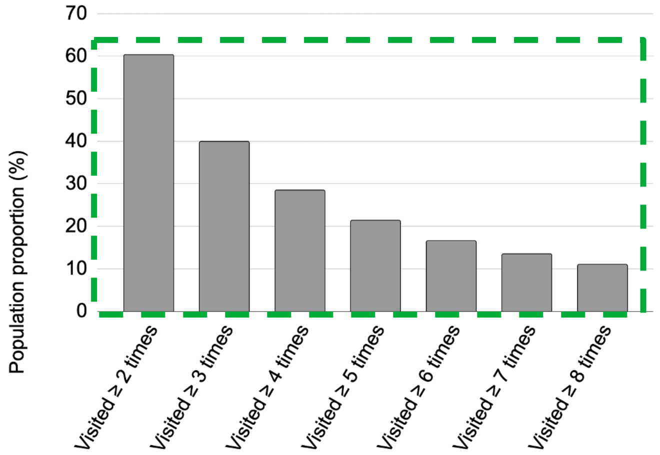

## Segment #2
**repeat-visit** users (60% of sample)

- Performance **improves** over time
- Some **warm up period** needed, with **large lift** thereafter



# Across the segment of repeat visitors, the sequence-capable model outperforms the baseline algorithm by a wide margin

| Number of Visits | Transformer ROC AUC | XGBoost ROC AUC |
|---|---|---|
| ≥ 2 | **0.88** | 0.80 |
| ≥ 3 | **0.90** | 0.78 |
| ≥ 4 | **0.90** | 0.77 |
| ≥ 5 | **0.90** | 0.75 |




CapitalOne

# Agenda

- Description of the problem
- What is a Transformer? Rationale for it in Recommender Systems
- Site personalization POC
- Performance
- Conclusion and future work

# Conclusion and future work

- In our POC, we replaced XGBoost with NVIDIA's Transformers4Rec framework
  - Transformer learns to credit a visitor's conversion to appropriate prior experiences, unlocking sequenced data for classification

- Work under progress today:
  - Apply Transformers4Rec to Capital One authenticated space
  - Pipeline to map sequenced data to embeddings, to use as features in flat models
  - Work with NVIDIA to enable multi-GPU training
  - Look into click sequence, webpage visit sequence, etc.

- Some more theoretical lines of future work:
  - Compare against LSTM, LSTM w/ Attn. (*cf.* Arava et al. 2018)
  - Model absolute timestep rather than relative ordering (Li et al. 2020)
  - Compare incremental vs. batch learning
  - Explore how same / similar system can be used for performance reporting

# Acknowledgment

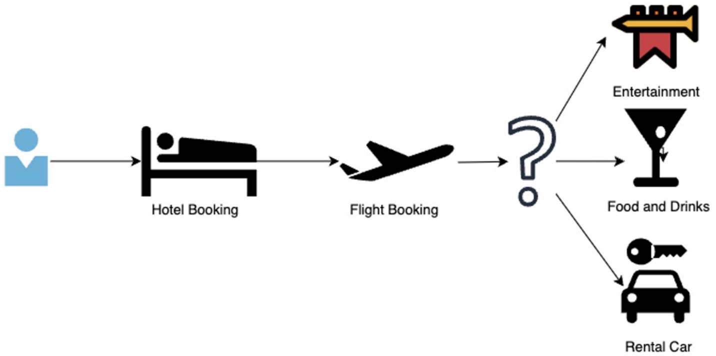

- We thank **Jacob Dineen** (now a CS graduate student at Arizona State University) for invaluable assistance throughout the engineering process


Thank you!

# Appendix

# Transformer4Rec's popular use case is next-item prediction, but ours is classification

- NextItemPrediction is the most popular use-case for T4Rec:
  - Show ad of product you think they will book next *based on products booked in recent past*




  - CapOne customers do not usually book multiple products in quick succession

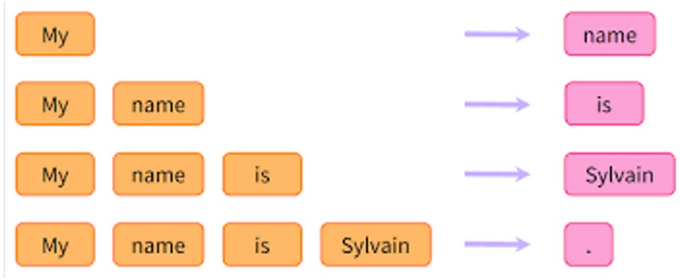
- Instead, we need to classify a sequence of experiences (e.g., impressions) as ending in a conversion or not
  - Then we score ads w/ probability of conversion based on learned parameters, picking highest-scoring ad to show next

# The training procedure differs between the general language case and our application of Transformers

- In general, language models are *pretrained (self-supervised learning)* on a large corpora of text with a specific task
  - Encoder models usually have the task of masked language modeling (MLM)
  - Decoder models usually have the task of casual language modeling (CLM)
- *Pretrained* models are then fine-tuned on downstream tasks, like question answering, sentiment analysis, autocompletion, etc.

**Masked Language Modeling:** Randomly masking some percentage of the input data and 'filling in the blanks'


**Causal Language Modeling:** Predicting the next token following a sequence of tokens




[MLM & CLM]

# “Pooled” crediting strategy fails to capture impression recency

- Accumulate # past impressions for each banner as we move down timesteps, with no signal of recency
- Poor strategy for frequent visitors, esp. those that come to Homepage 8 or more times (> 10% population, significant source of lift)

| visitor | num_times_ saw_autonav | num_times_ saw_savings | num_times_ saw_ventr | ... | has_credit_ card | convert_ to_savings |
|---|---|---|---|---|---|---|
| Jane | 1 | 0 | 0 | ... | yes | no |
| Jane | 1 | 0 | 0 | ... | yes | no |
| Jane | 1 | 1 | 0 | ... | yes | no |
| Jane | 1 | 1 | 1 | ... | yes | yes |

# Tracking which impression was seen at which previous timestep explodes the number of features and leads to training inefficiency

- Features for…
  - Most recent visit: `saw_ventr_frst_rcnt`, `saw_savings_frst_rcnt`, …
  - Second-most-recent visit: `saw_ventr_scnd_rcnt`, `saw_savings_scnd_rcnt`, …
  - Third-most-recent visit: `saw_ventr_thrd_rcnt`, `saw_savings_thrd_rcnt`, …
  - … etc.

- Suppose AutoNav banner is best for a small population *across every timestep*:
  - Fewer data points on COAF conversions
  - Model needs lots of data at each timestep to mimic *time-insensitive generalization*
  - Need algorithm that shares generalizations (i.e., parameters) across timesteps