Predictive Caching@Scale

A scalable ML caching at the Edge

Vaishnav Janardhan

Adit Bhardwaj

Akamai

Experience the Edge

Overview

Problem Introduction

Caching Algorithms

ML for Caching

Traffic prediction

PeSC

System design challenges

Conclusion and Future work

The Akamai Platform

Distributed caching at the Edge

A Global Platform…

- Over 240,000 servers
- In over 2,400 locations
- In over 1,600 networks
- In over 650 cities
- In 138 countries

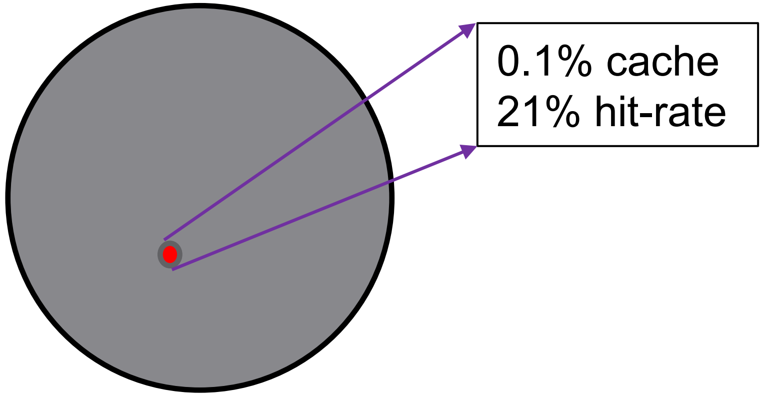
…With Enormous Scale

- 50+ trillion bits per second
- 60+ million hits per second
- 95+ Exabytes delivered per year
- 250M+ attacks defended per day

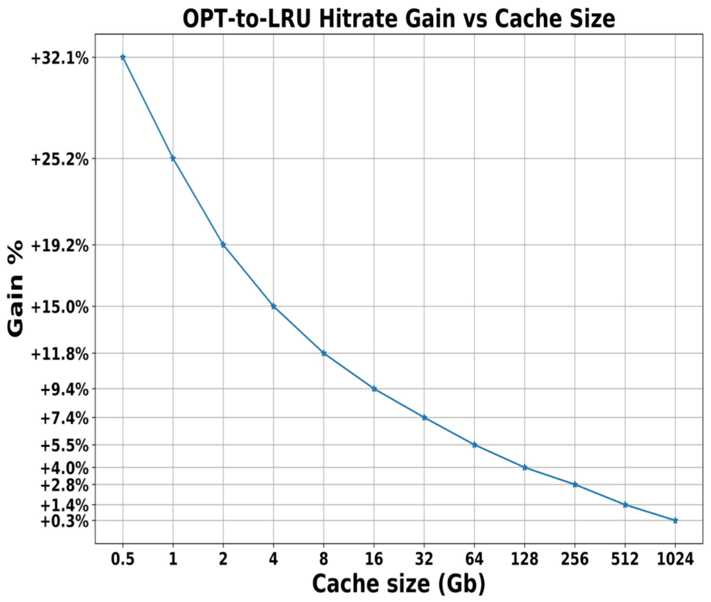
Caching Algorithms

Limitation of classical caching Algorithms

- Classical/Online caching algorithm
  - (LRU, LFU, S4LRU e.t.c.)
  - Are cheap and effective for web-traffic
  - Highly competitive in terms of cache effectiveness
  - Widely applicable and needs no meta information.



- Theoretically optimal Caching scheme, Bélády's
  - Uses future arrival time knowledge
  - Optimal only for single sized object cache
  - Can provide huge performance gains over online schemes.



- Variable object sized optimal can provide 190% mean and 133% median gains

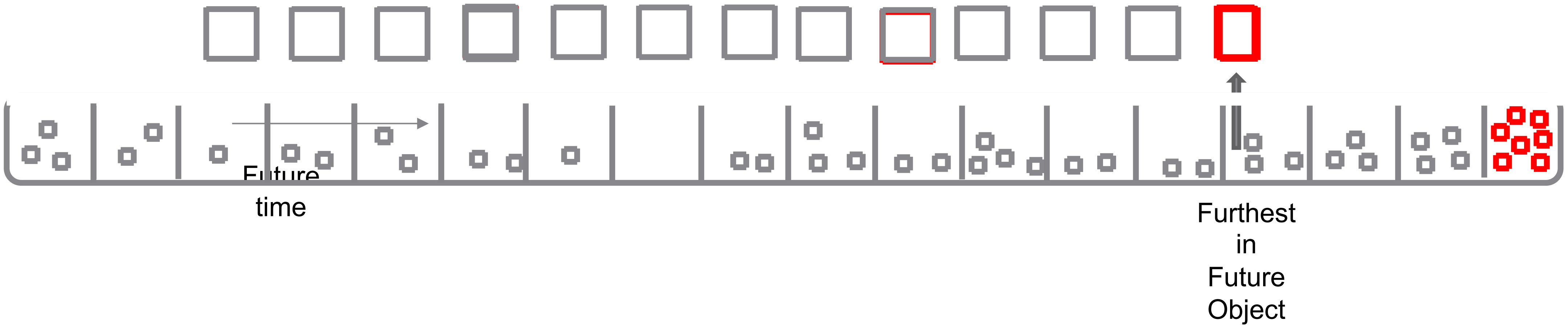
ML for Caching

How to use ML for caching

Previous methods of ML for caching
- Object Popularity prediction
- Using Reinforcement learning
- We developed a variant of **Bélády's** for our PredictiveCaching.

- Mimicking **Bélády's** we just need sequencing
- We don't have to predict actual arrival times but only quantized future arrival bins.
- Only need to predict if the object falls inside or outside of the eviction boundary.

How Bélády's OPT works **Quantized Bélády's**



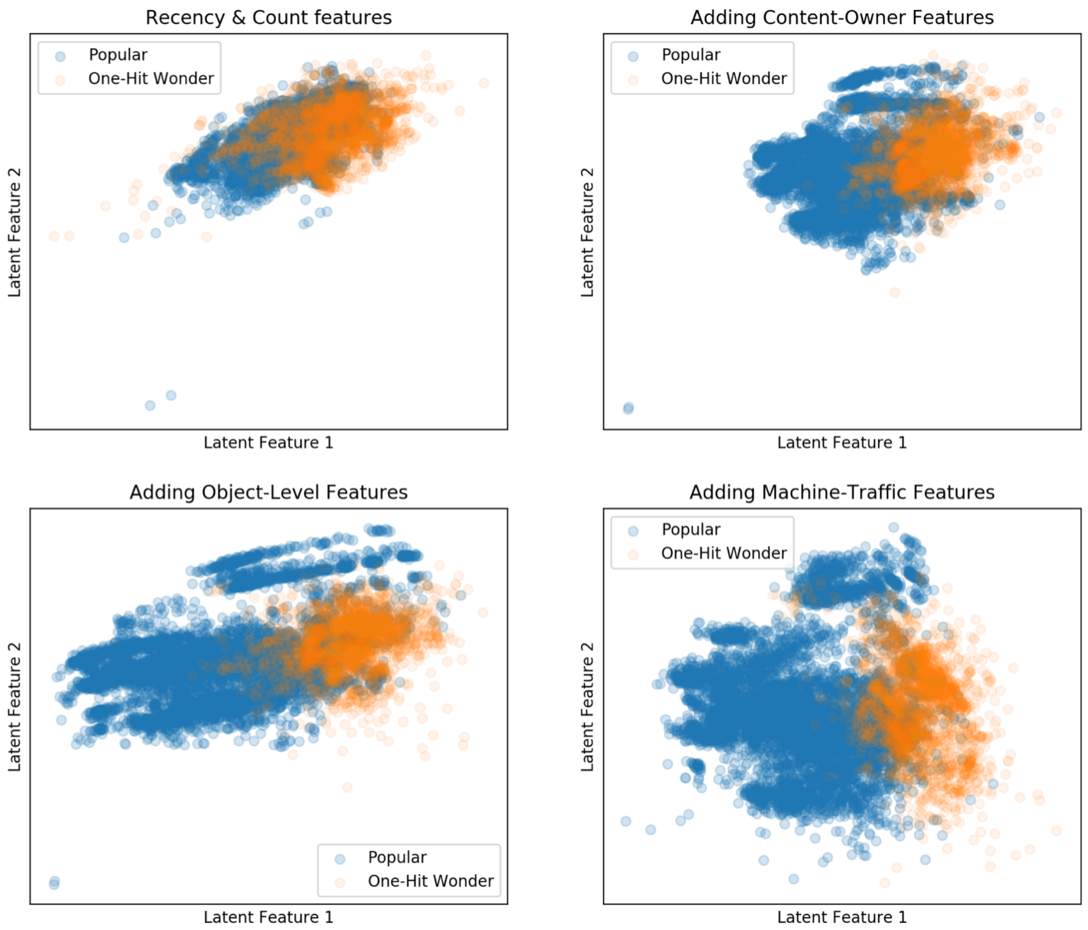
Traffic prediction

Challenges in Predicting Internet Traffic

- Multi-tenancy leads to competing traffic patterns overlapping at the Edge, making predictions challenging.

- We use several informative properties of content to differentiate these patterns:
  - Content level: Owner, size, type etc.
  - Machine level: Traffic throughput, traffic mix ratio, timeofday..
  - Network topology level: cache layer hierarchy, geo location, end-user patterns.

- Feature tuning helps to distinguish unique traffic patterns



Traffic Prediction

Simplifying the Predictions

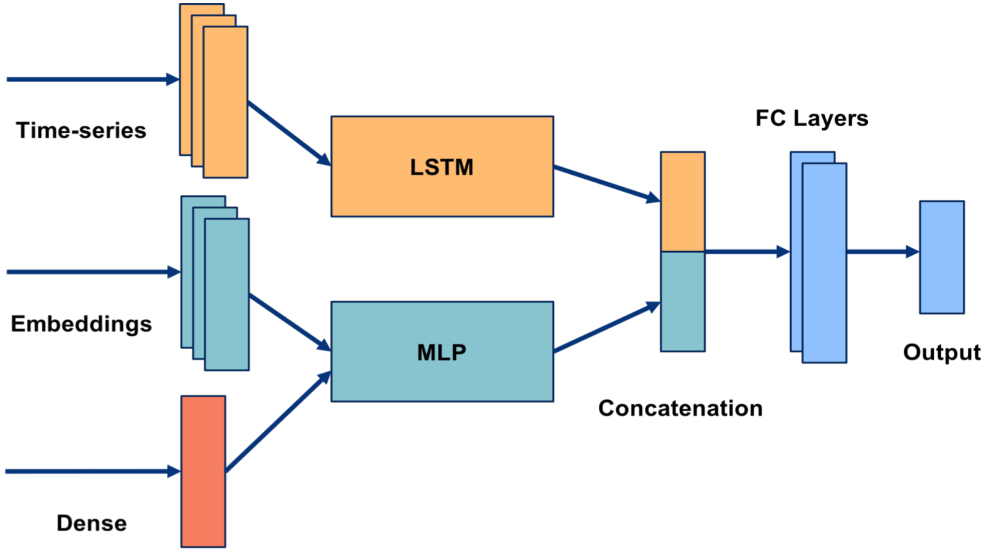
- Next arrival time Prediction → Regression problem
  - Output range [0.001 msec - 2*24*60*60 secs]: Difficult to reach optimal model parameters.
  - Difficult to relate Regression loss to downstream cache hit-rate losses.
- Approximation:
  - Regression → Ordinal Multi-class classification via quantization as sequencing is sufficient to mimic OPT.

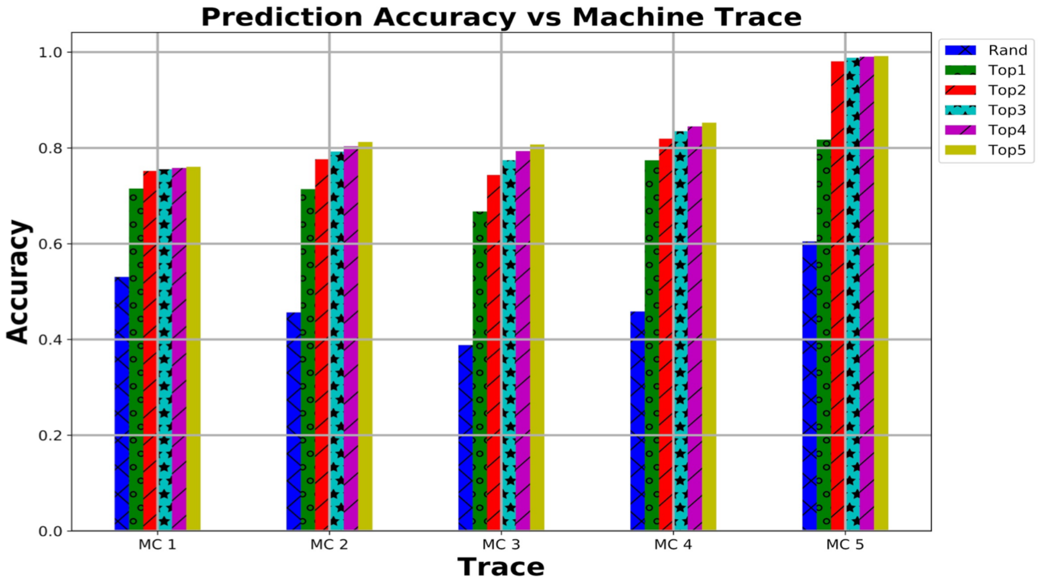
$$L(y,p) = \sum_{c=1}^{m} W_{ij} * y_i * log(p_j)$$

Where,
y: true label
p: predicted probability
m: output classes
i: true class
j: predicted class
W_{ij}: weight of loss for the pair (i,j)

(Order enforcing loss)
|i-j| ↑ ⇒ W_ij ↑

  - Easier to relate mis-classification rate to cache hit-rate performance.
  - Can leverage TopK predictions in caching policy.

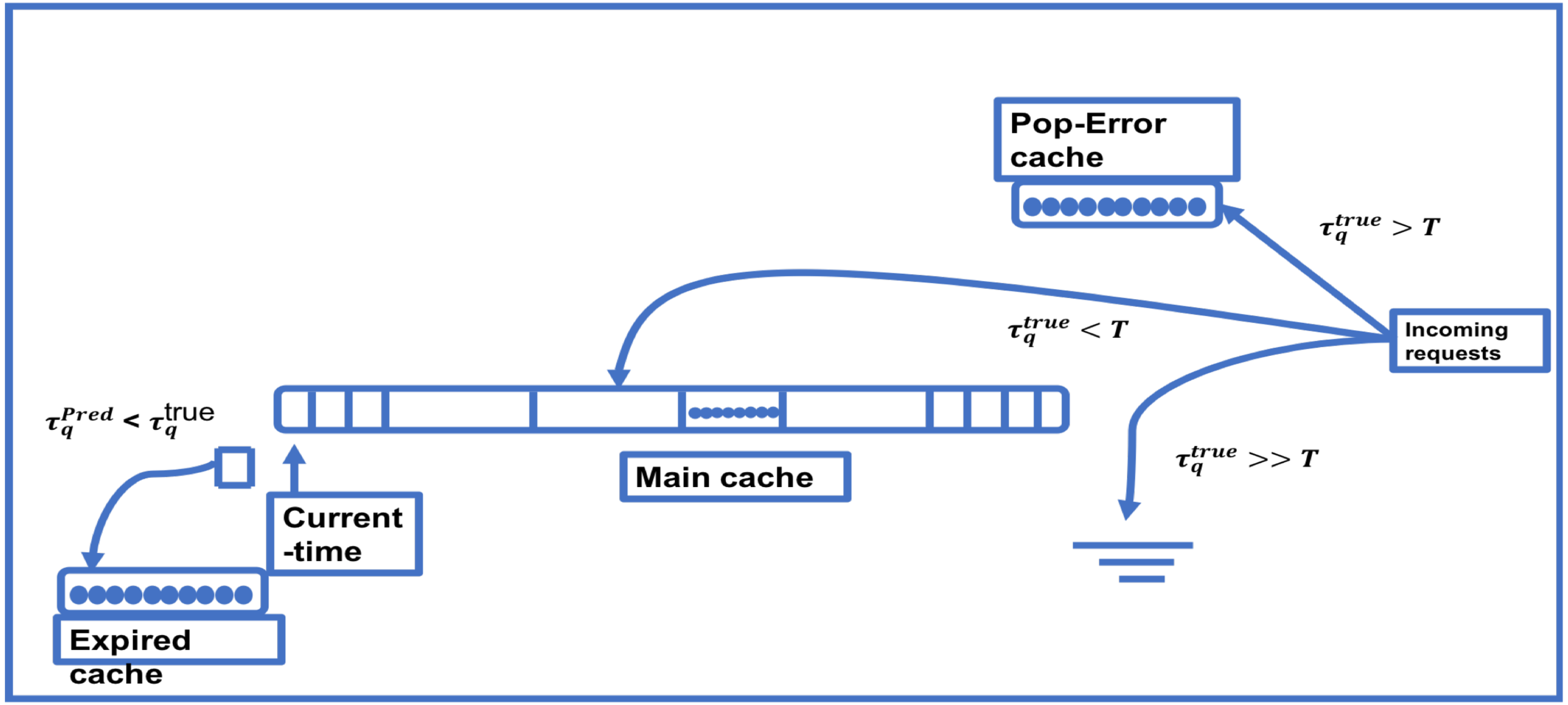




Prediction-Error Segmented Cache (PeSC)

A caching to recover from Prediction errors

- Requirements:
  - Outperform LRU-based policy in an online situation.
  - Robust enough to use unreliable predictions with varying confidence.
- Strategy:
  - Isolate the prediction errors into separate segments of controlled size.
  - Use next most likely predictions from topK predictions to make eviction decisions.

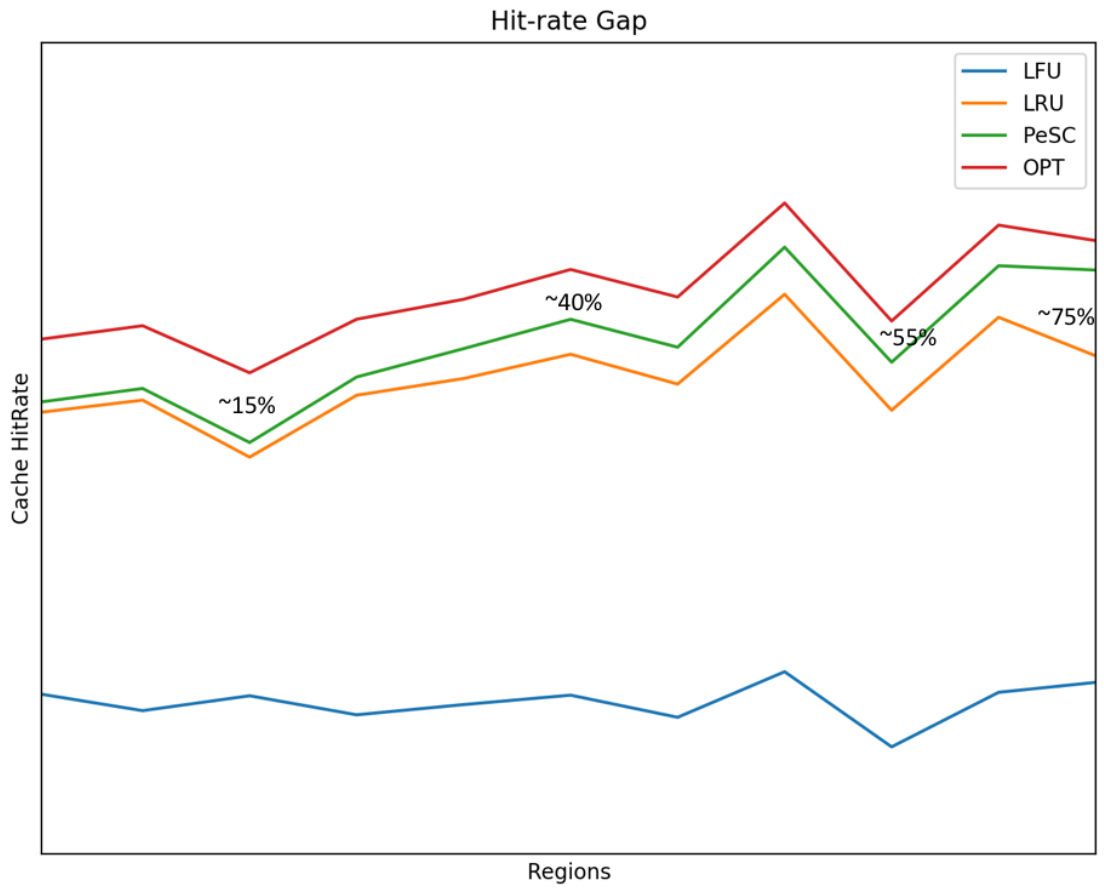


Akamai Experience the Edge

Performance of PeSC

Predictive Caching closing the gap on LRU

- We compare the performance of PeSC we trace the cache hit gap between OPT, LRU and LFU. And the of % cache hit gap recovered by PeSC.
- We turn on the PeSC on several regions and plot the Cache hitrate for 4 schemes.
- 10%-60% of the gap recovered by PeSC depending on the traffic/Region.



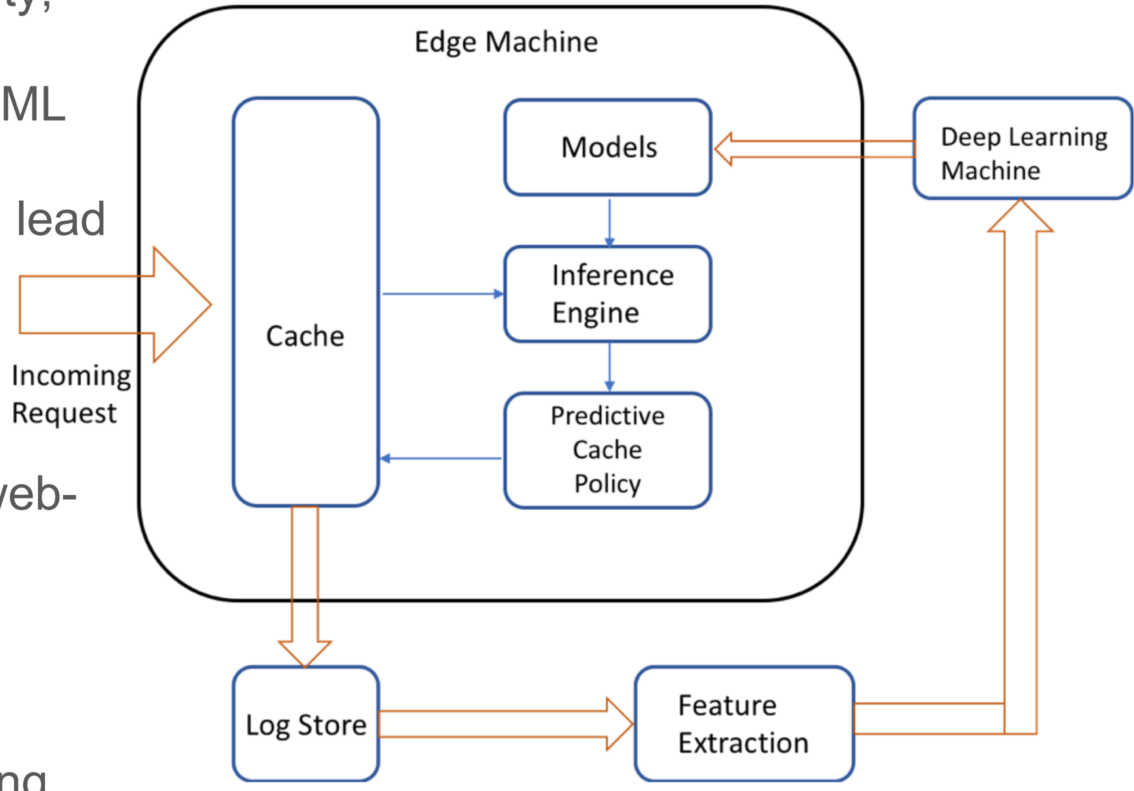
Akamai Experience the Edge

PeSC System Design – Data Pipeline

Building a low overhead data pipeline at the Edge

- Challenges:
  - Proxy servers were not designed to connect cacheability, load-balancing and user-attributes for ML training.
  - Edge machines don't have spare capacity to generate ML training sets.
  - Multi-tenancy leads to missing/corrupt/default data can lead to silent failures in the ML pipeline.

- Solutions:
  - Re-write application modules to connect storage and web-application tiers.
  - Infer and log load-balancing attributes.
  - Feature extraction and transformation modules were changed to work under constrained resources.
  - Deploy data validator: Range check on features, tracking changing distribution of features, monitoring default value imputations, etc.



PeSC System Design – Automation and Training

Building a robust model for the Edge

- Challenges:
  - Training robust models for the multi-tenant Edge workload.
  - Model should be adaptable for changing traffic and concept shift.
  - On-the-fly model hyperparameter selection, capturing dataset silent failures, etc.
  - The large volume of training data and the frequence of re-training the model.
- Solutions:
  - Selecting less sensitive hyperparameters/model design, lr-scheduling, cv-selection over multiple epochs.
  - Continuous learning via partial retraining models on new available data.
  - Targeting most critical PoPs in the network.
  - Pre-training models and loading weights from older models.
  - Down-sampling datasets to reduce the size.
  - Multiplexing GPU machine to handle multiple edge servers.
  - Exploring FP16 training for faster training.

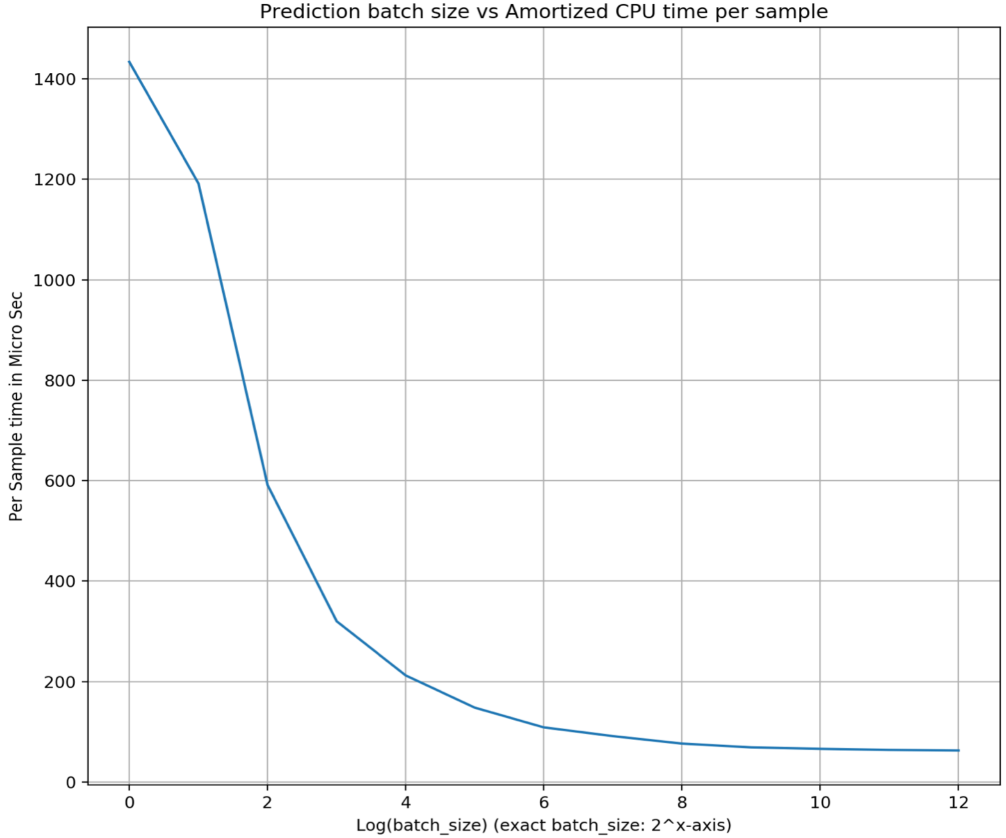
- > 2mil req/hour per Edge
- Retraining ever few hours with 3-7 day of data.

PeSC System Design – Inference

A low-overhead inference at the Edge

- Challenges:
  - Cost of Inference is extremely critical on performance sensitive Edge servers.
  - Inference cost should be comparable to sys-call cost.
  - No hardware accelerators are available at the Edge. Traditional x86 machines.
  - Missing features can lead to silent inference failures.
- Solutions:
  - Lazy-Batched Inference: Decouple content serving and eviction policy logic.
  - Do lazy inference on a batch of requests rather then for each request which reduces amortized cost.
  - Re-writing server application logic to collect and scale features at inference time, which use to be only available at the time of logging.
  - Inference cost of batch size 256 is ~ 100 micro sec.

Inference time per request



Conclusions and Future work

- Demonstrates there is a lot of value in re-thinking limitations of classical algorithms
- We can safely build and use ML, deep inside a high performing web-server or similar real-time applications
- Future work:
    - Building a more general model that works across traffic patterns
    - Reduce the cost of training
    - Building predictive caching for variable object sizes

Akamai Experience the Edge