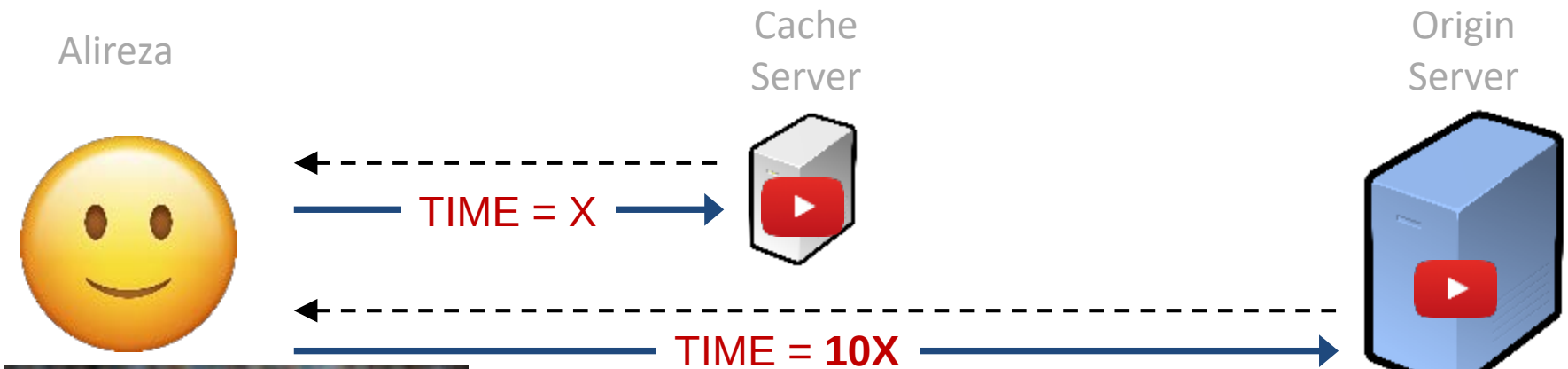
UNIVERSITY OF MINNESOTA

DEEPCACHE: A Deep Learning Based Framework For Content Caching

Content Caching is Important!



reduces user perceived ***latency***

increases user's ***QoE***

reduces ***costs***

But caching is NOT easy!

Caching Is Not Easy

Resources *(e.g. storage space)* at Cache Servers are ***limited***

All content objects cannot be cached at a cache server !

Content agnostic caching node should make precise **cache decisions**

Caching Decisions include:

1. what content to cache?
2. when to cache?
3. what happens when cache is full?

Caching Policy

Popular examples include: Least Recently Used (LRU) and its variants
Least Frequently Used (LFU) and its variants } *Reactive*
Static Caching (or prefetching) → *Proactive*

Challenges in Caching

- Heterogeneity in content (reusable info.) types and sizes

  (e.g. web pages *vs.* Videos *vs.* music …)

- Content-object requests patterns frequently change over time

  (thus content-object popularity also changes)

- Accounting for diversity in content life spans

  (short-lived *vs.* long-lived content-objects)

- Handling burstiness and non-stationary nature of real-world content-object requests

Drawbacks of Existing Approaches

- Although *reactive caching* acts faster, it ignores future object popularity.

- While *proactive caching* accounts for possible future object popularity, it assumes object requests patterns are stationary.

Drawbacks of Existing Approaches

- Although *reactive caching* acts faster, it ignores future object popularity.
- While *proactive caching* accounts for possible future object popularity, it assumes object requests patterns are stationary.
- Adaptive caching policies (e.g. TTL-based) handle the heterogeneity and burstiness of object requests, but still rely on the history of object requests and ignores possible future object request patterns.

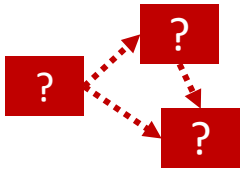
Existing approaches don't generalize to different conditions

No Single Caching Policy Likely Works Well in All Settings!

Goal

AI comes to rescue!

Is it possible to develop a *self-adaptive* and *data-driven* caching mechanism (via deep learning) that is able to cope with *diverse* and *time-varying* content object characteristics (e.g. arrival patterns, popularity, life spans) and improve cache efficiency?



how to model?

input features?

interpret and apply?

The Rationale…

Understanding content object characteristics is key to building any caching mechanism.

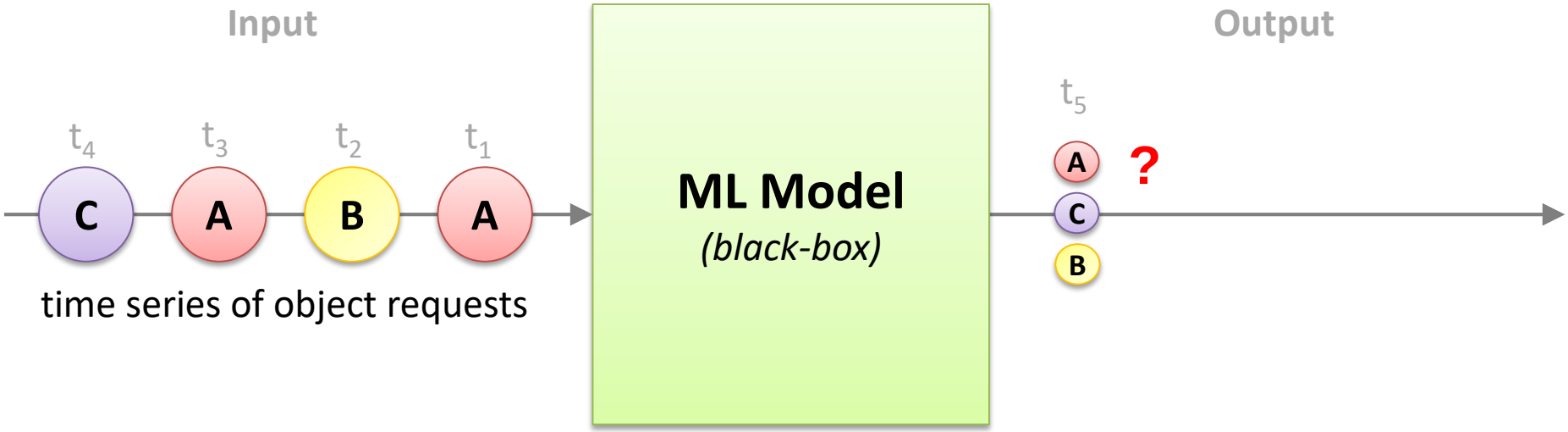
If one can accurately predict vital content object characteristics ahead of time, cache efficiency can be improved.

This work focuses on predicting content object popularity *ahead of time* such that caching can be improved *(e.g. by prefetching them)***.**

Does this make sense? El-Clasico example ..

A Naïve Simple Approach

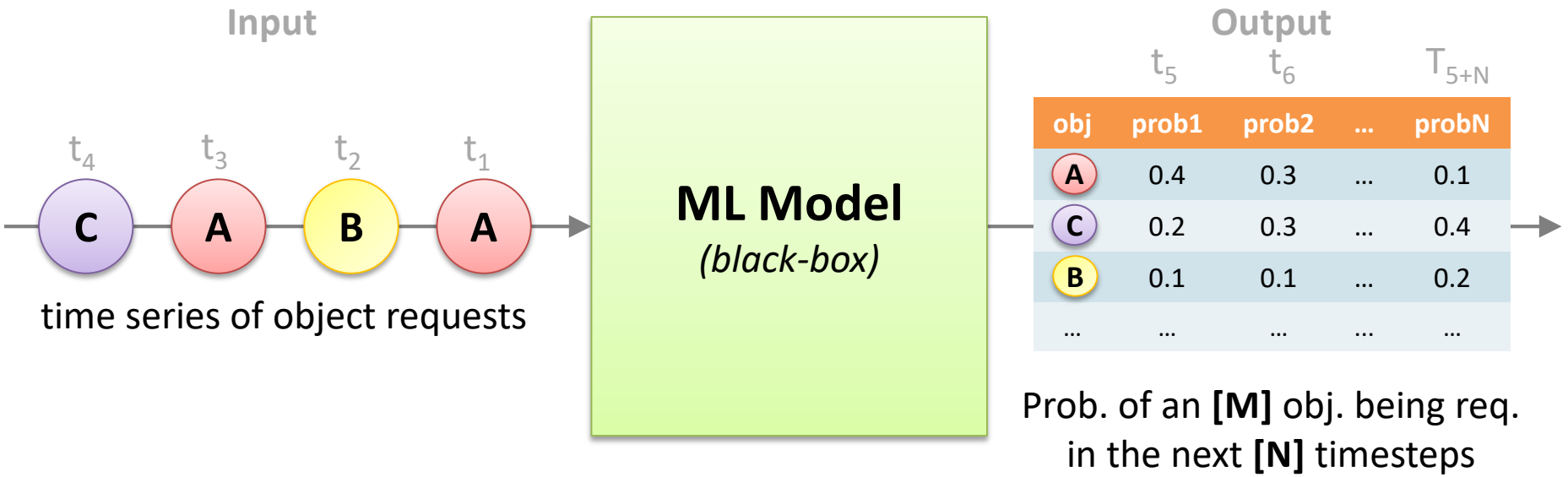
Goal: Predict *which* object will be requested ("*next-item* prediction")



- ✓ Easy to understand
- ✗ Which object will be requested and at what time will in general likely be random!
- ✗ Will be hard to predict accurately due to the inherent randomness of request processes

It's Better to Predict …

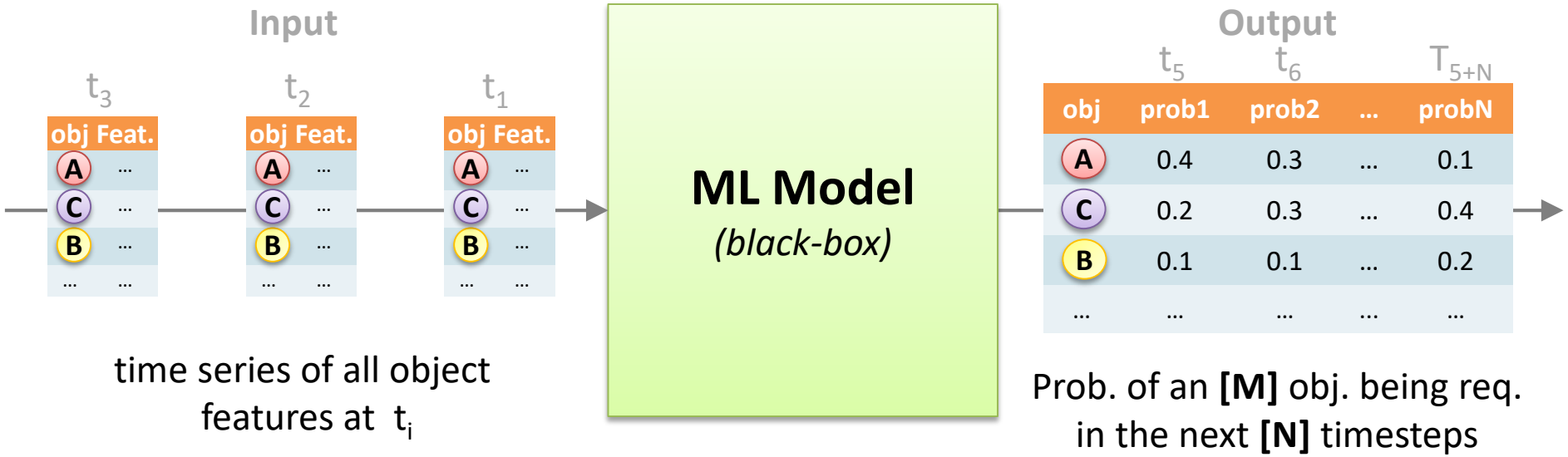
Goal: Predict Content Object Popularity in *Next Time Window*



- ✓ Also easy to understand
- ✗ Model does not *explicitly* account for possible object request *correlations*
- ✗ Model may also learn *erroneous* "relationships" between object requests

Can We Do Better?

Goal: Predict Content Object Popularity within *Next Time Window* *with fixed prob. vector by accounting for possible object correlation*

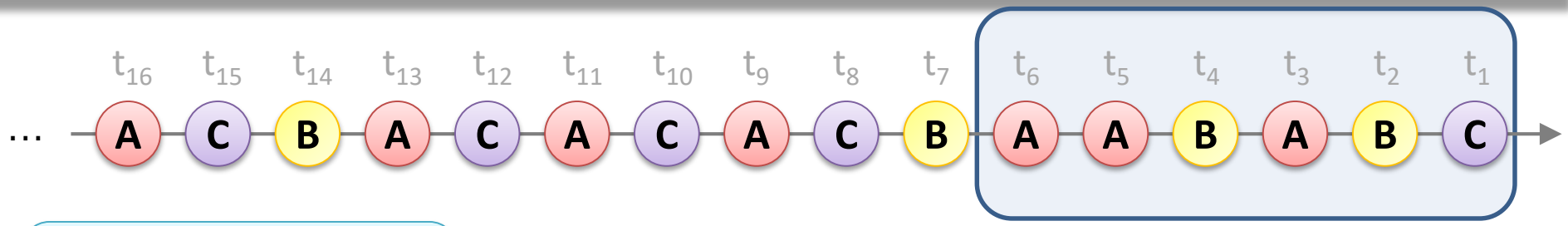


time series of all object features at t_i

Prob. of an **[M]** obj. being req. in the next **[N]** timesteps

- ✓ Still easy to understand
- ✓ Able to disambiguate correlation/independence among objects *within time window*
- ✗ But may still erroneously infer/miss relationships among objects *beyond* time window
- ✗ Fixed number of content objects, new ones cannot be added

Prop. Approach – Sequence Construction



Configurable Parameters
Window size (W) = 6
Step size (S) = 3

$P(W_A^2 | W_A^1) = 0.5$

Input: Given a time series of object requests

- For every object *(say A)*
  - Compute the probability of object A in W *(i.e. # req. for A / total req.)*
  - Slide the window by step size S
  - Repeat till end of time series;
- We get a series of sequence of probabilities for object A

Construct sequences of probabilities for all content objects from a large synthetic set of content objects having diverse characteristics

Proposed approach – Seq2Seq Model

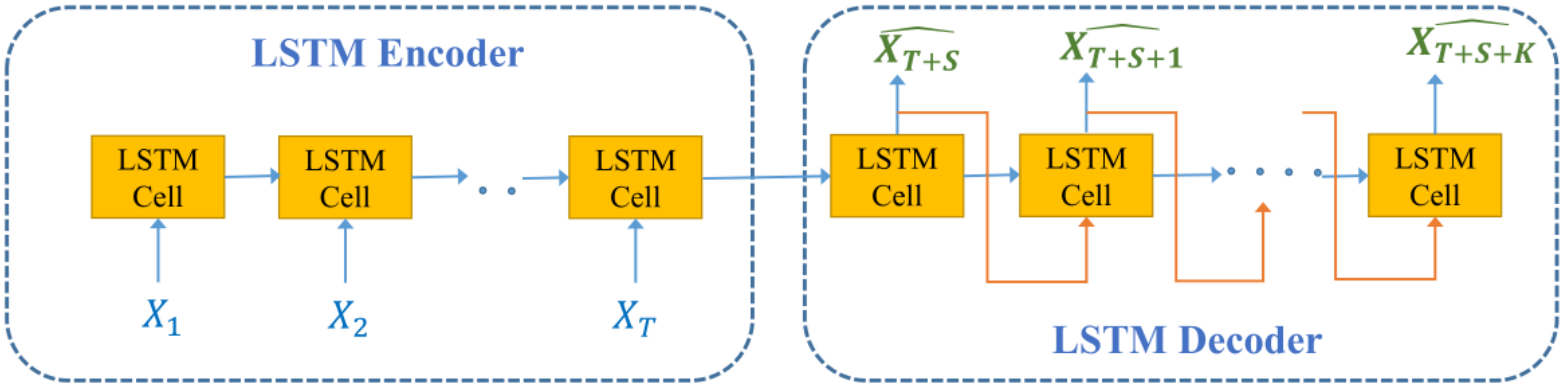
- Model the problem as **Seq2Seq** learning
  - Seq2Seq modeling was first developed for machine translation but has proven to be a powerful tool in other domains as well.
- Train the model using sequences generated from the large corset pus of synthetic object requests
- Once modeled, given a sequence of probabilities of an object, model is able to predict the future probabilities (i.e. popularity) of the object at various time steps.
- Once the future probabilities of all the objects are known, one can apply a caching policy *(e.g. pick top K popular objects)*

Why Choose Seq2Seq?

- Enable jointly predicting several characteristics of objects together
  - Provide flexibility in terms of predicting varieties of outputs together with possibly varying input/output sequence lengths
  - Can predict object's popularity over multiple timesteps in the future
  - Can also predict sequential patterns among objects
  - Can classify sequence into pre-defined categories

- For seq2seq modeling, LSTM-based models are the most successful
  - Can capture any long-term and short-term dependency among objects
  - Designed to avoid vanishing and exploding gradient problems when building deep layer neural network models

Content Popularity Prediction Model



- LSTM Encoder-Decoder takes a sequence of objects represented by their input feature vectors and predicts the next **k** outputs associated with each object.

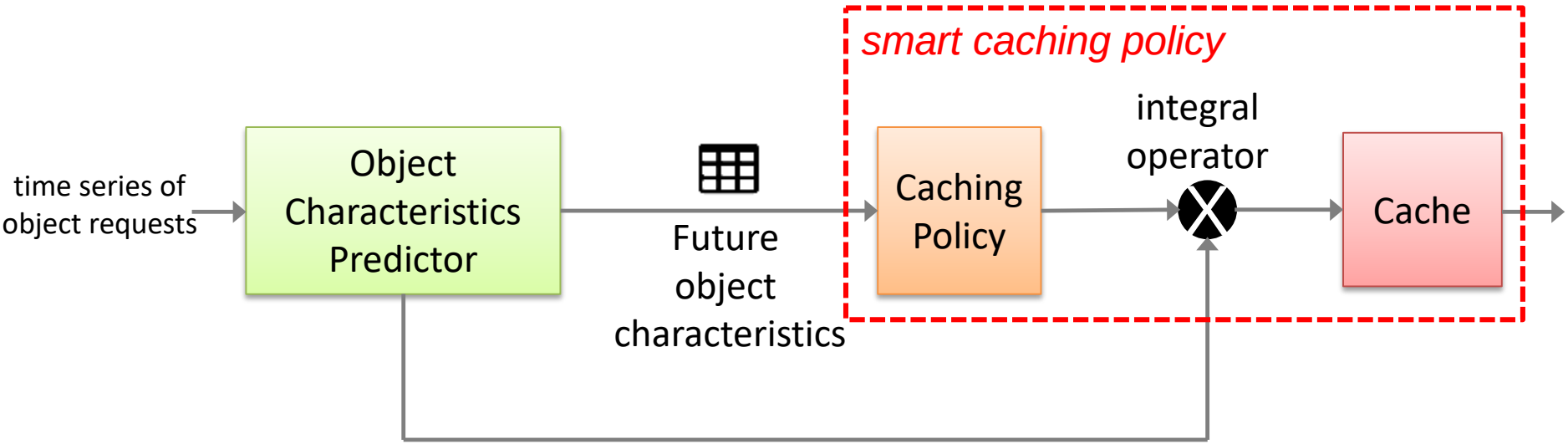
- In our case:
  - Input feature vectors are defined as object probabilities of occurrence computed based on a pre-defined time (or sequence length) window.
  - Output is defined as a sequence of next **k** future probabilities associated with each object.

- Once trained, LSTM Encoder-Decoder can predict (for example) next **k** hourly probabilities for each object that can be utilized in making cache policy decisions.

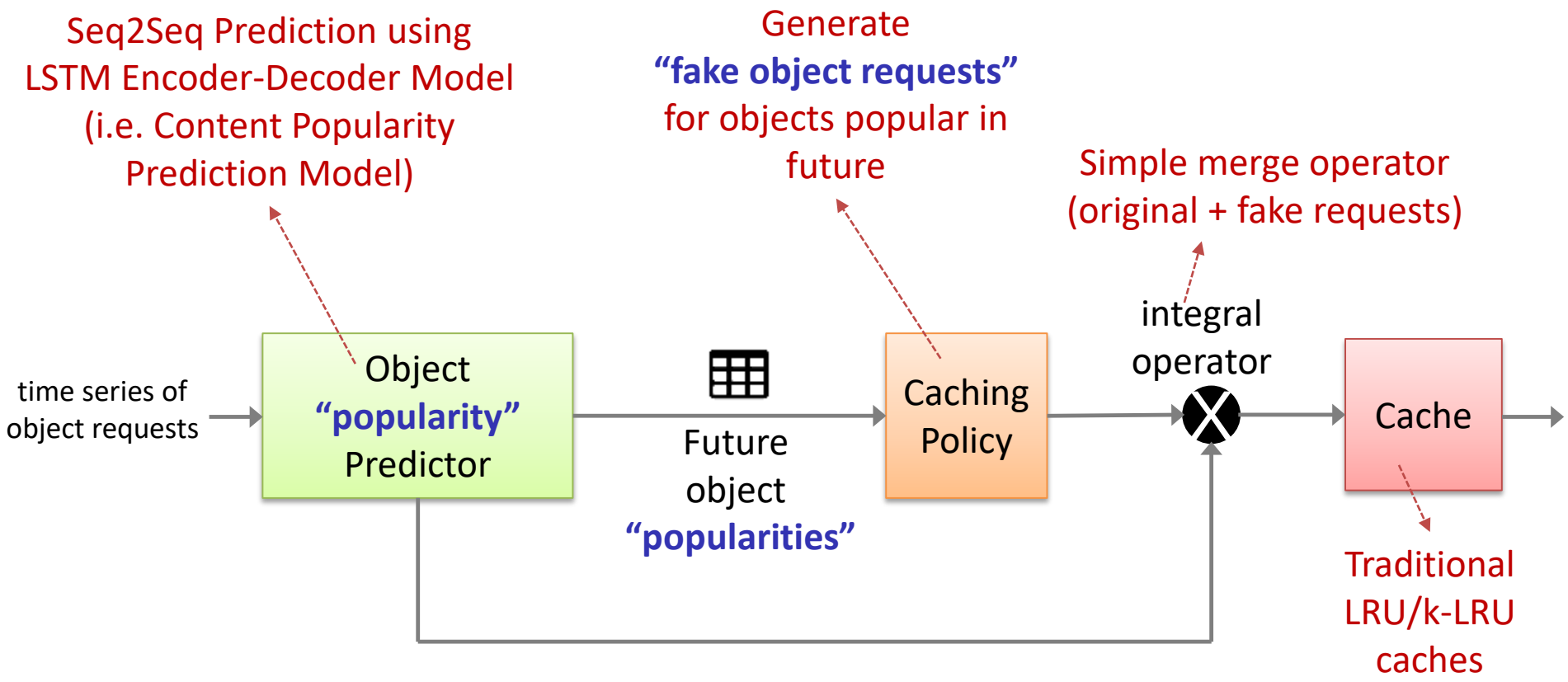
DEEPCACHE

- A Framework that leverages state-of-the-art ML algorithms to improve cache efficiency

- Predict object characteristics to develop new (or enhance existing) caching strategies
  → paving way to develop ***"smart"*** caching policies



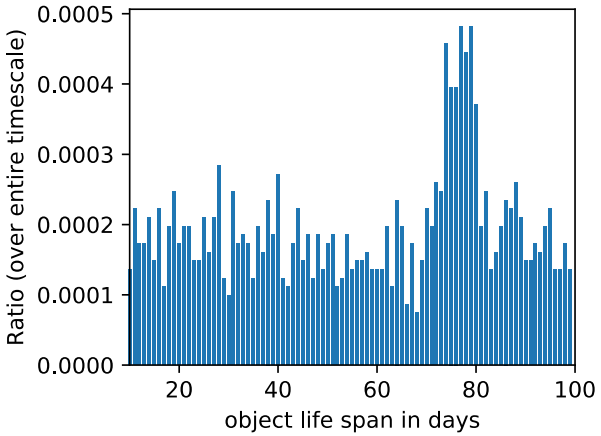
A Case for DEEPCACHE



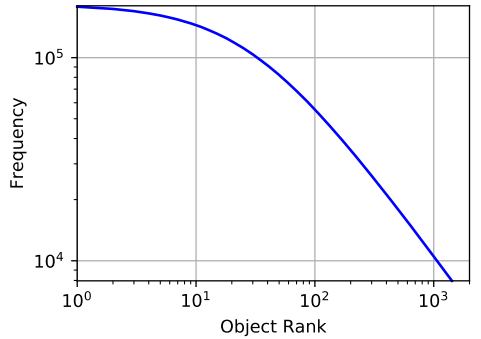
Synthetic Workload Generation & Evaluation Settings

Dataset 1 (D1):

- 50 objects, 80K requests
- object popularities follow (varying) Zipf distributions, 6 intervals
- Cache size = 5



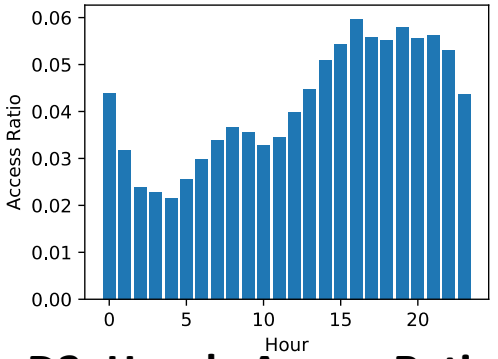
D2: Histogram of Object Life Span



D2: Object Popularity

Dataset 2 (D2) *realistic workload*:

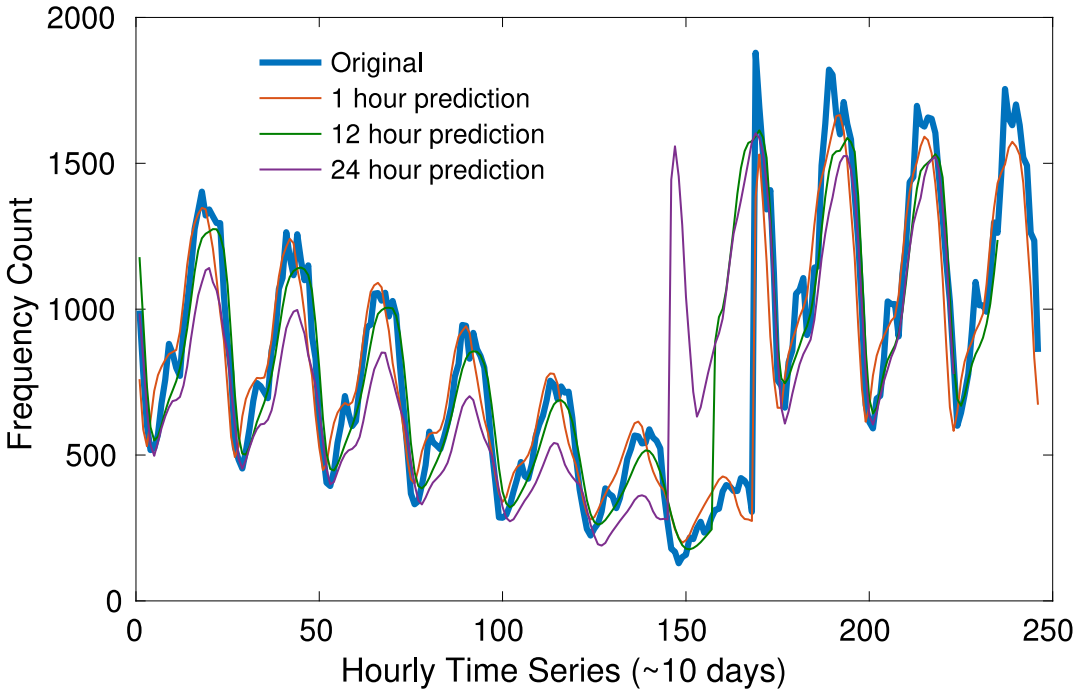
- 1425 objects, 2M requests
- Generalized Zipf distribution to generate object popularities
- Varying object life span (# days object seen)
- Each object's request arrival process within a day follows a diurnal pattern
- Daily access rate of each object follows linear/non-linear function, # requests for obj i diminishes each day
- Cache size = 150



D2: Hourly Access Ratio

Performance of Content Popularity Prediction

LSTM performs quite well in tracking the original time series over the predicted time-series at multiple future time steps



Prediction Accuracy

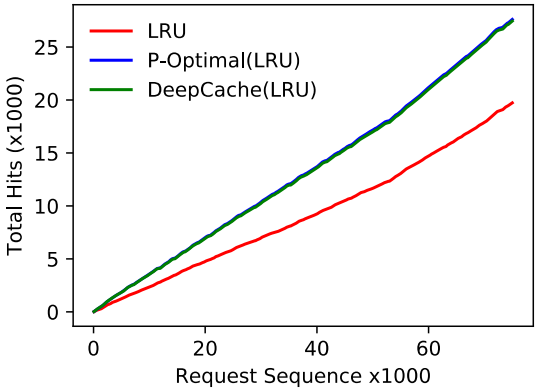
	MSE	MAE
Dataset 1	1.2×10^{-6}	4.8×10^{-3}
Dataset 2	3.8×10^{-6}	8.3×10^{-3}

Mean-squared-error (MSE) and mean-absolute-error (MAE) for both datasets are low, i.e. strong performance of our LSTM model.

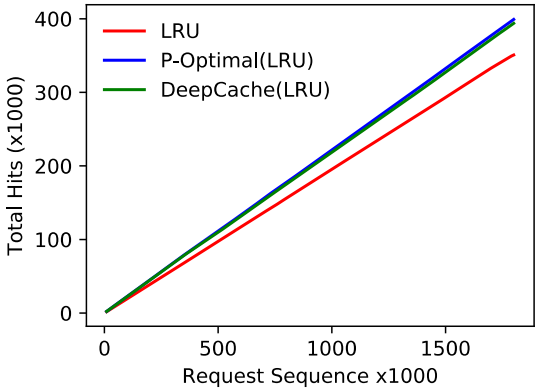
LSTM performs well for predicting <1, 12, 24> hour(s) ahead of time

Figure shows prediction in comparison with original values over a time series of ~ 10 days

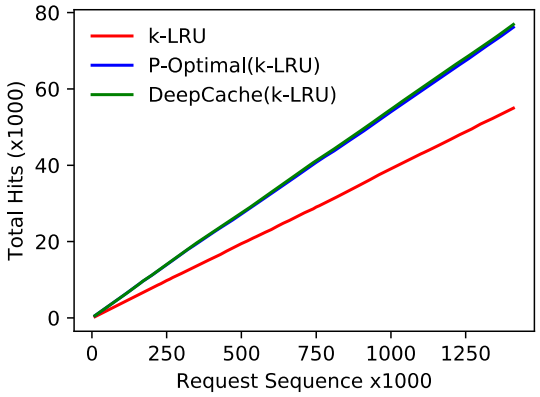
Cache Hit Efficiency in DEEPCACHE



D1: LRU



D2: LRU



D2: k-LRU

For both datasets D1 and D2:

DEEPCACHE-enabled-LRU outperforms Traditional LRU. Similar observation for k-LRU.

P-Optimal shows performance with 100% content popularity prediction accuracy.

DEEPCACHE is able to attain the optimal performance!

Conclusion

- *DEEPCACHE Framework*, a framework that applies state-of-the-art machine learning tools to the problem of content caching
- Cache prediction problem using seq2seq modeling
- Simple yet effective cache policy that generates *"fake requests"* to interoperate with traditional caching policies
- Evaluation on synthetic datasets (that emulates realistic workloads) show DEEPCACHE enabled caches significantly outperforms caches without DEEPCACHE

Thank you!

Questions?