Predictive Precompute with Recurrent Neural Networks

Hanson Wang

Zehui Wang

Yuanyuan Ma

MLSys 2020

FACEBOOK

Defining Precompute

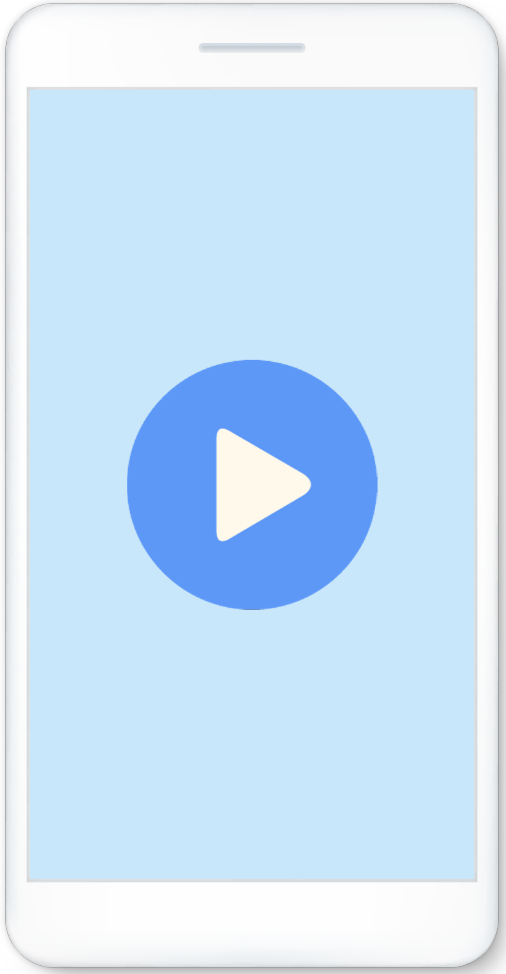
On client: **prefetching**

- Improve the latency of user interactions in the Facebook app by precomputing data queries before the interactions occur

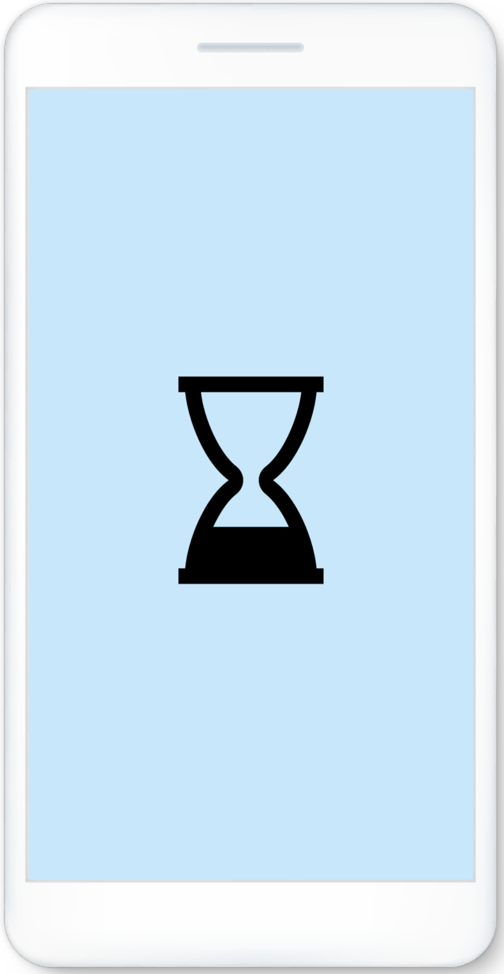
On server: **cache warmup**

- Improve cache hit-rates in Facebook backend services by precomputing cache values hours in advance

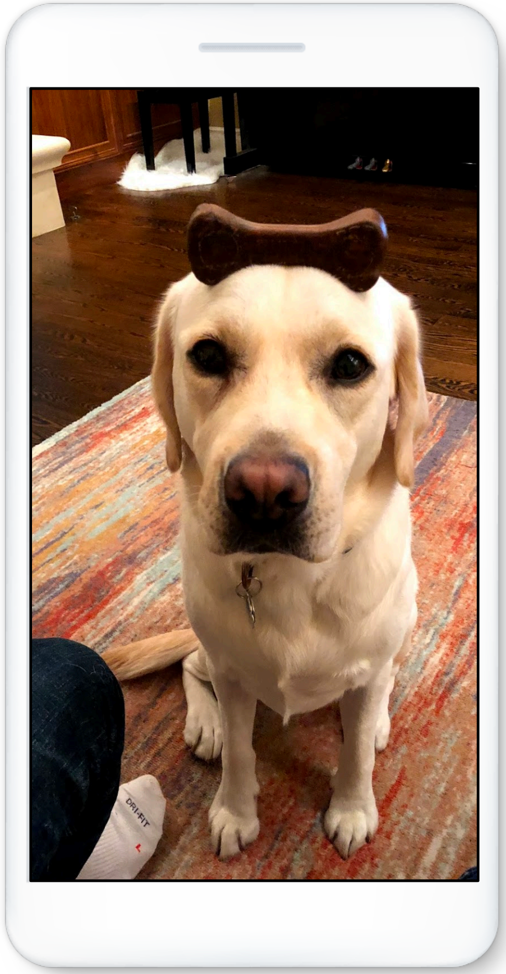
Defining Precompute: Prefetching



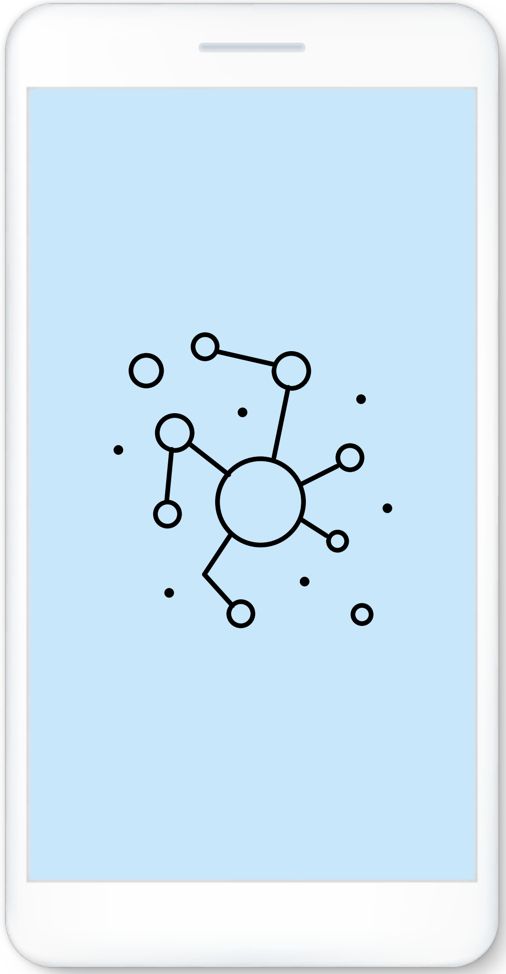
User opens
the tab



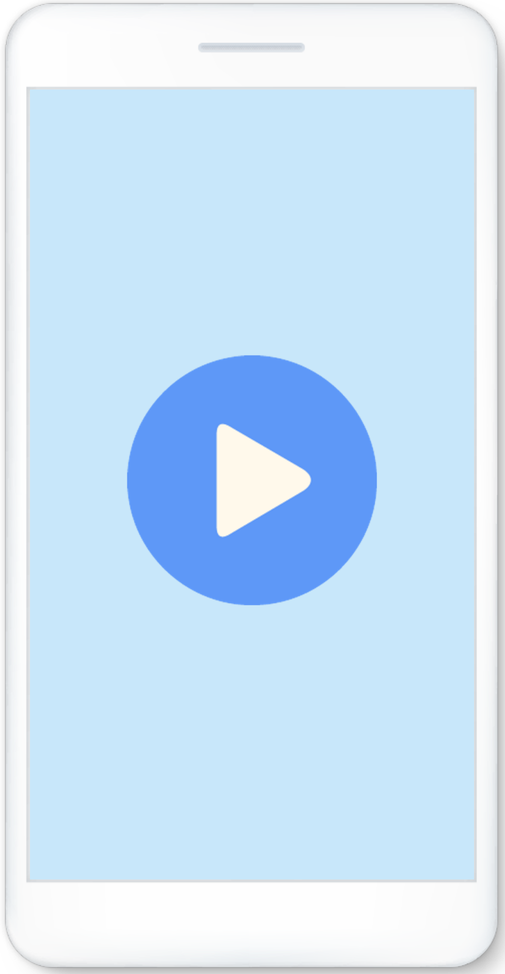
Wait for data
to arrive...



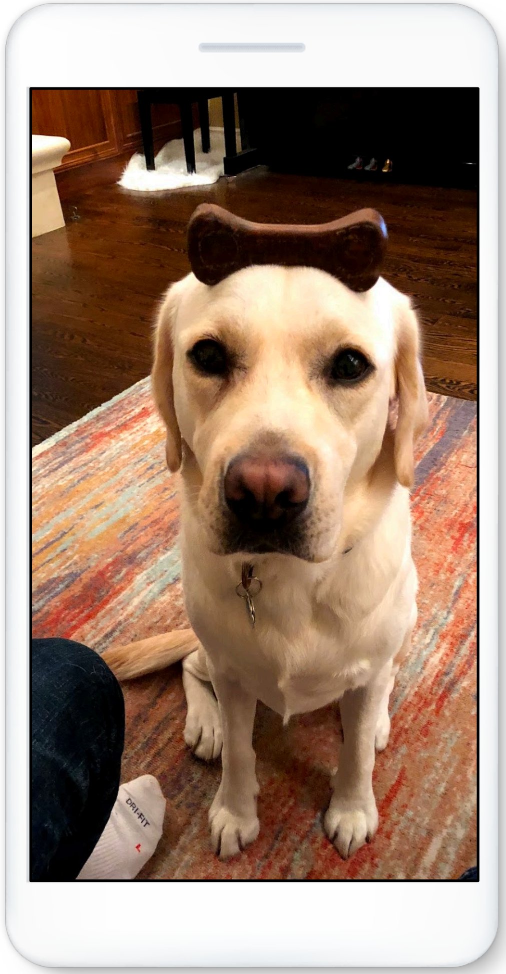
Defining Precompute: Prefetching



Data gets
precomputed
at startup time



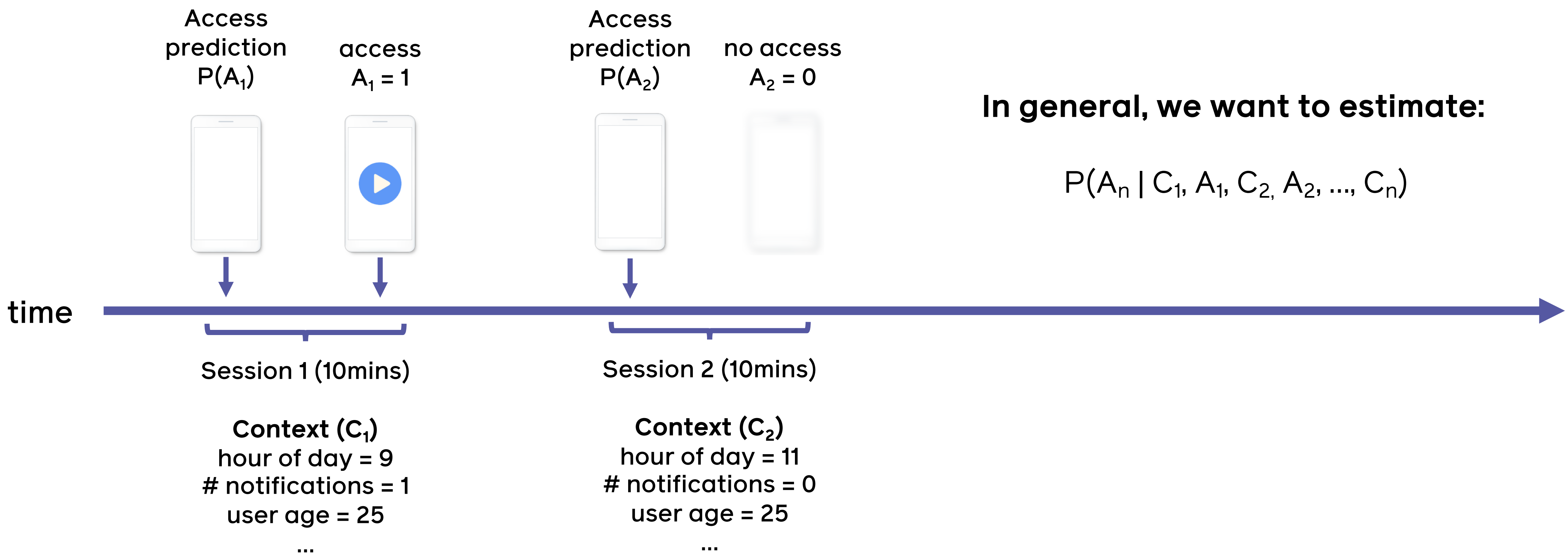
Data is
immediately
available!



Predictive Precompute

- Naïvely precomputing 100% of the time is too expensive
  - Facebook spends non-trivial % of compute on this
- Idea: **Predict user behavior to avoid wasting resources**
- Classification problem: **P(tab access)** at session start
  - Apply threshold on top of probability to make precompute decisions (can be tuned to product constraints)

Formulation as an ML problem



In general, we want to estimate:

$$P(A_n \mid C_1, A_1, C_2, A_2, \ldots, C_n)$$

Formulation as an ML problem

Features

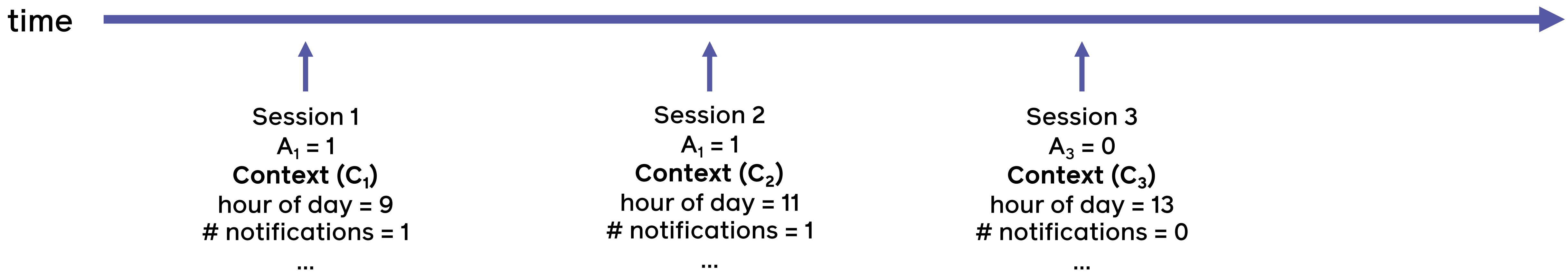
Simple features can be taken from current context (C_i)

- Time-based (hour of day, day of week)
- User-based (age, country)
- Session-based (notification count)
- **How to incorporate previous contexts and accesses?**

Formulation as an ML problem

Historical Features

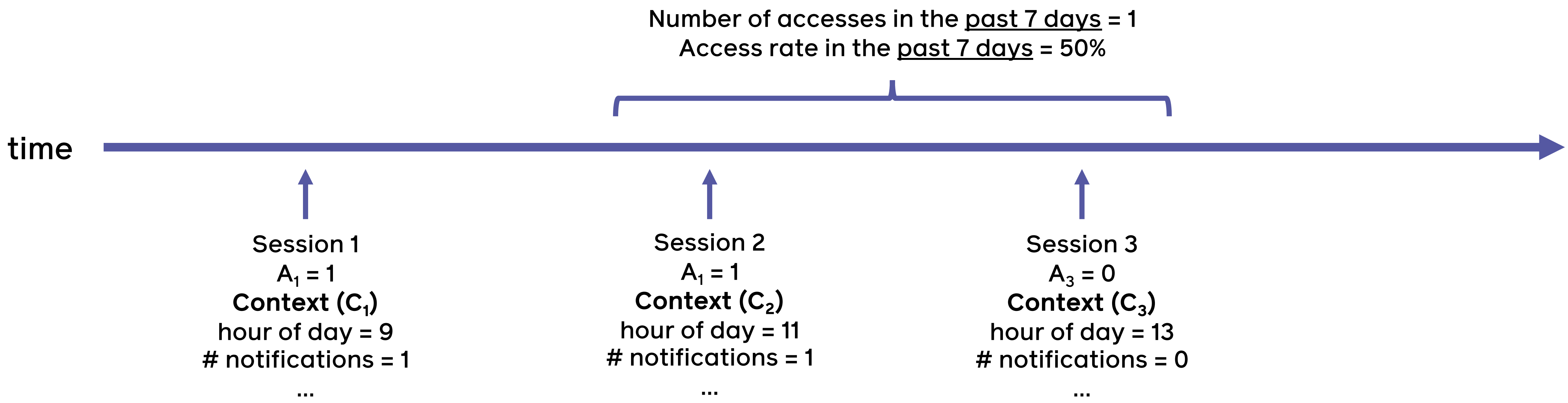
Historical usage features **must be “engineered”** for traditional models



Formulation as an ML problem

Historical Features

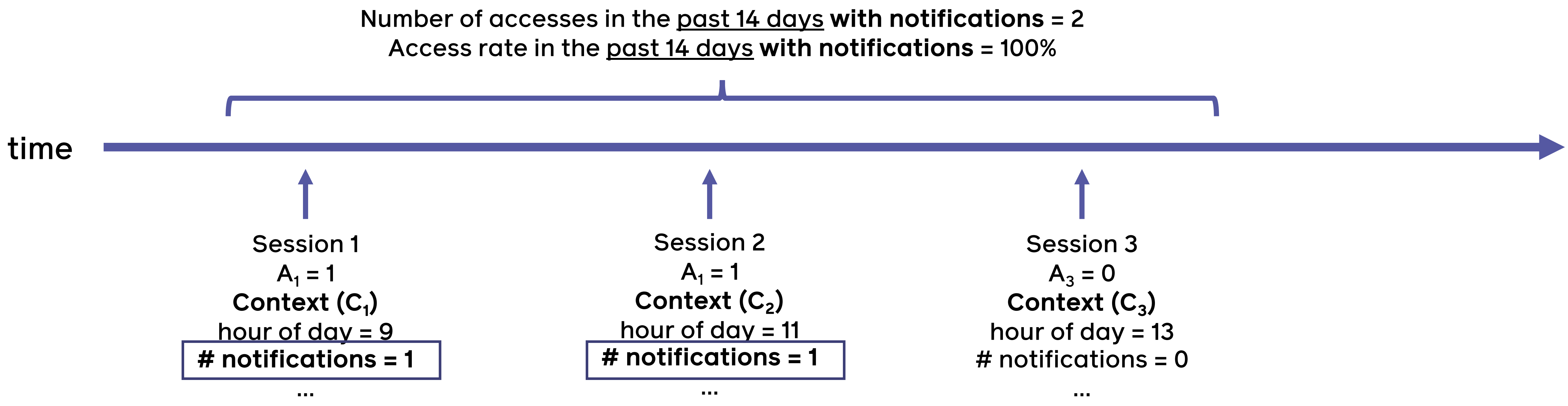
Historical usage features **must be “engineered”** for traditional models



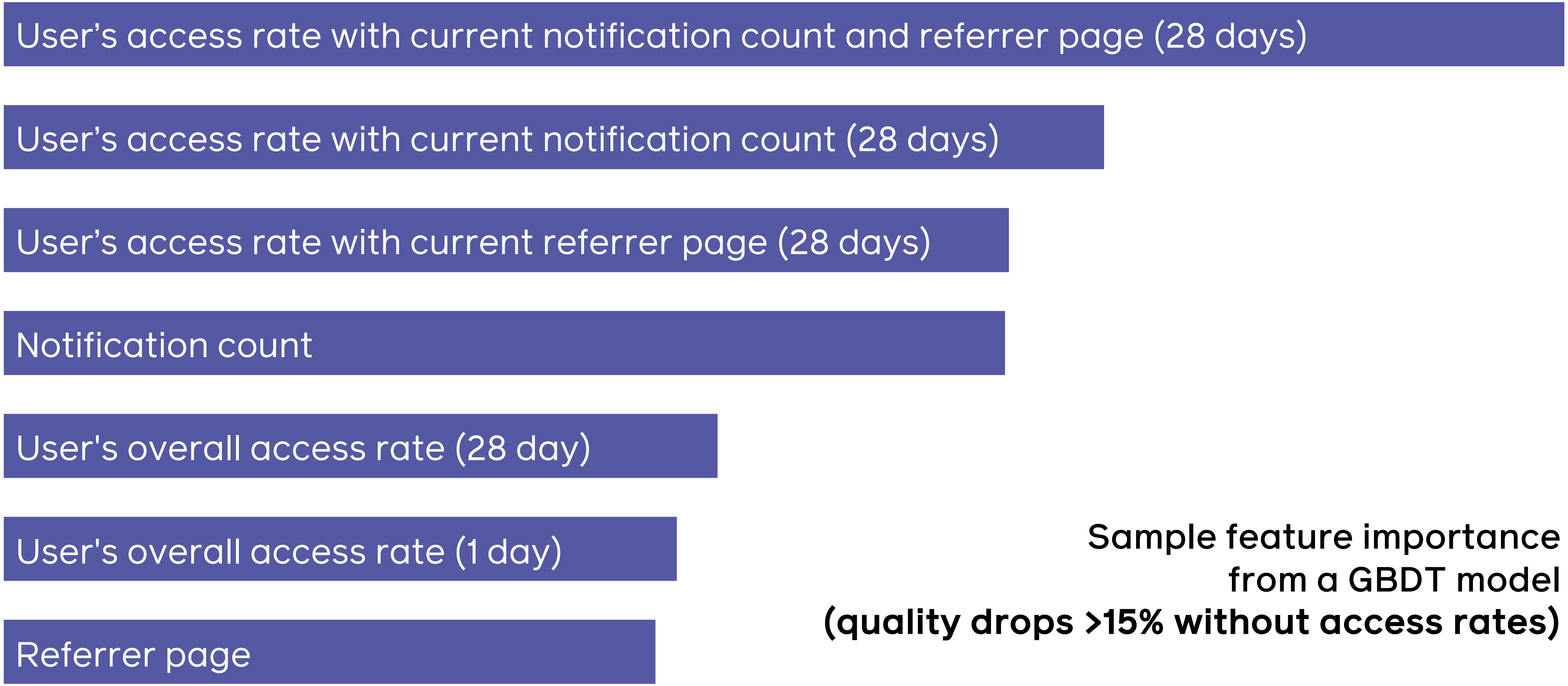
Formulation as an ML problem

Historical Features

Historical usage features **must be "engineered"** for traditional models



Historical features dominate feature importance…



Sample feature importance from a GBDT model (quality drops >15% without access rates)

Formulation as an ML problem

Features

“Recipe” for historical features:

- Select an **aggregation type** (count, access rate, time elapsed…)
- Select a **time range** (1 day, 7 days, 28 days…)
- (Optional) **Filter** on a subset of context attributes
  (with / without notifications, at the current hour of the day, …)

💥 **Combinatorial explosion of features!**

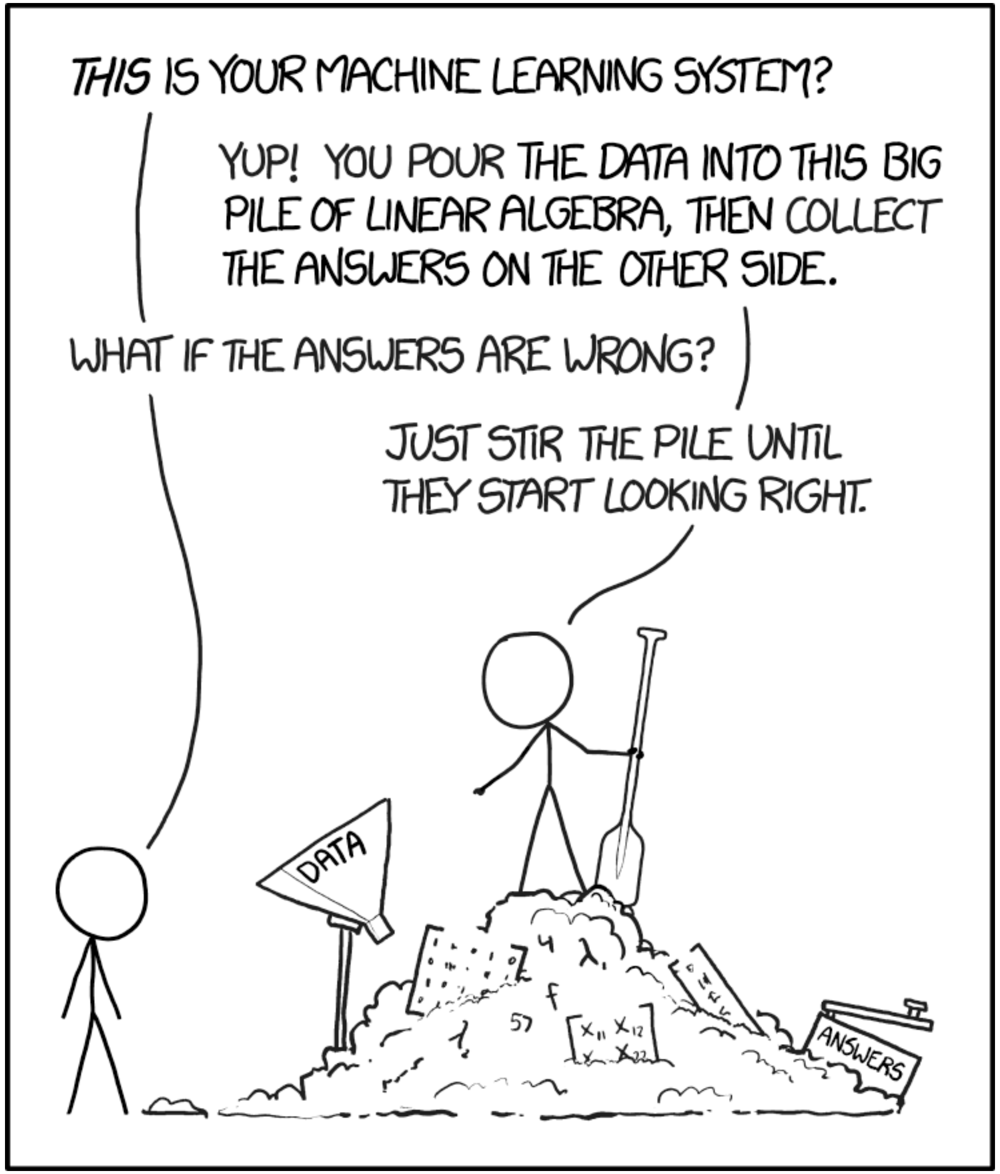
💰 Aggregation features make inference expensive!

Formulation as an ML problem

Models

Traditional models

- Simple baseline: output the lifetime access rate for each user
  - Most basic historical feature, surprisingly effective
- Logistic Regression, Gradient-boosted Decision Trees
  - Consumes concatenated vector of engineered features



Alt-text: The pile gets soaked with data and starts to get mushy over time, so it's technically recurrent.

— xkcd #1838

Neural networks to the rescue

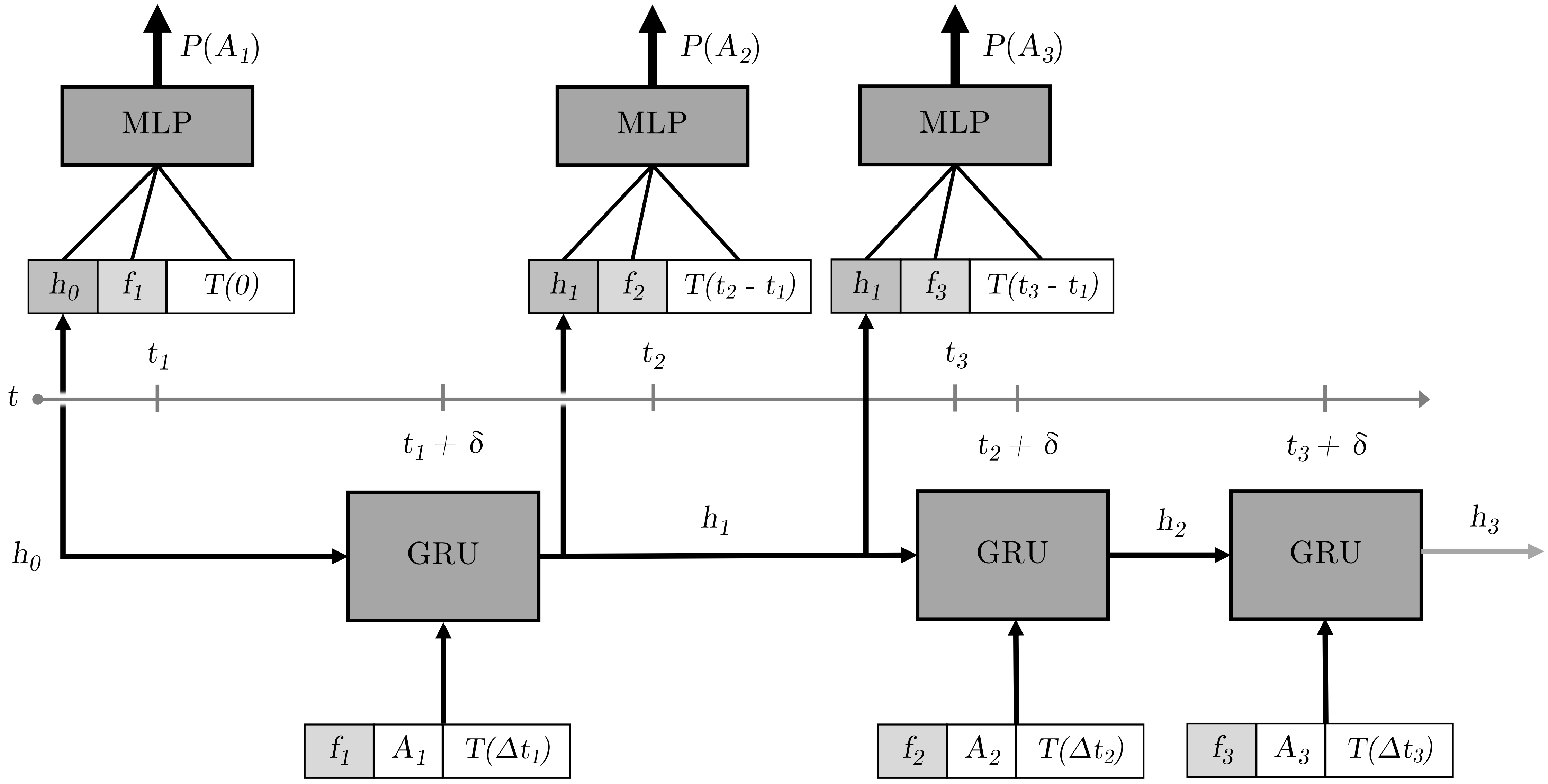
Recurrent neural networks address problems with historical features:

Complex, non-linear interactions between features can be captured through a hidden state "memory" for each user.

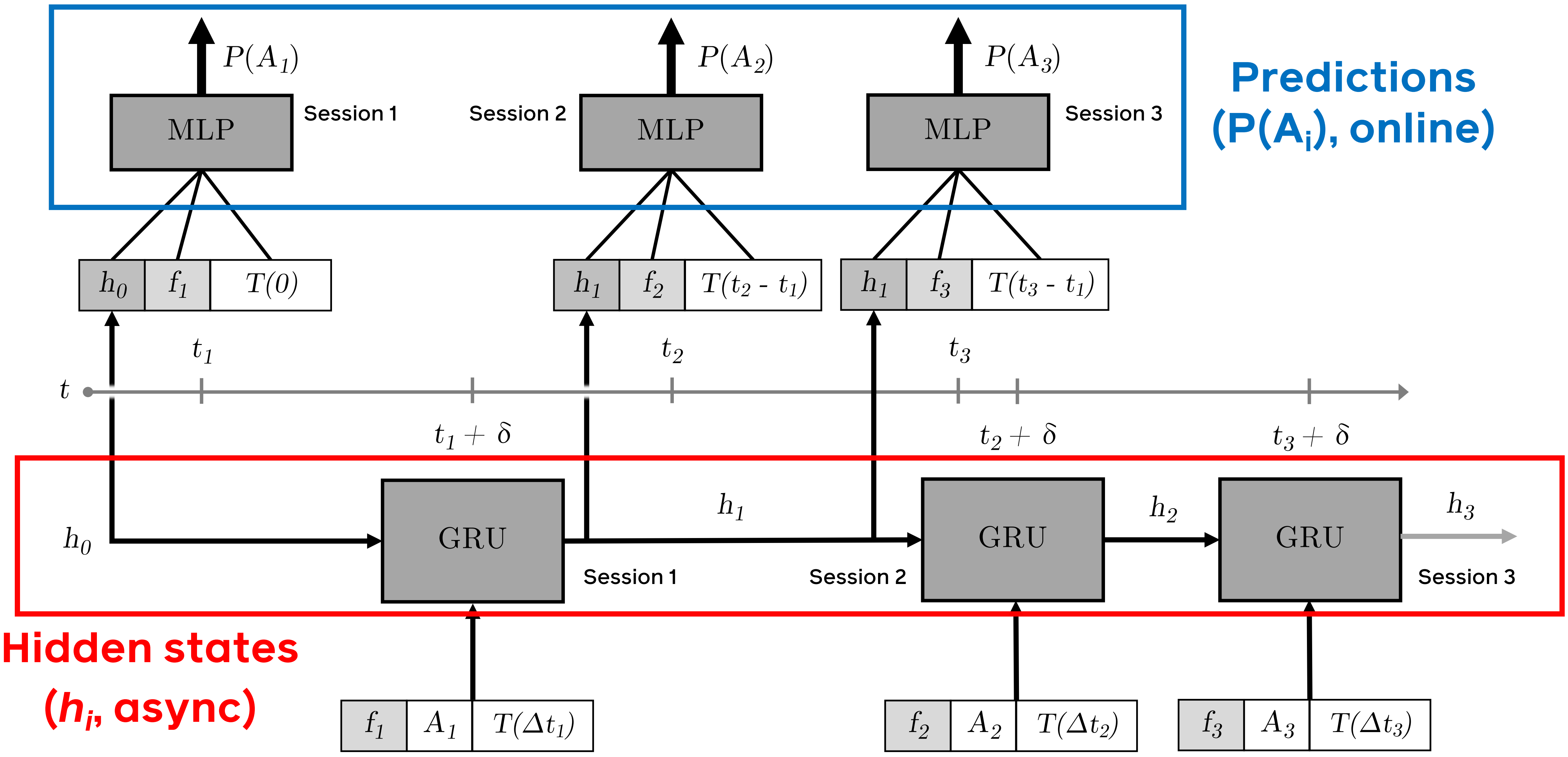
Hidden state updates are incremental in nature.
Storage consumption is bounded by the number of dimensions.

Model each user's session history as a sequential prediction task.

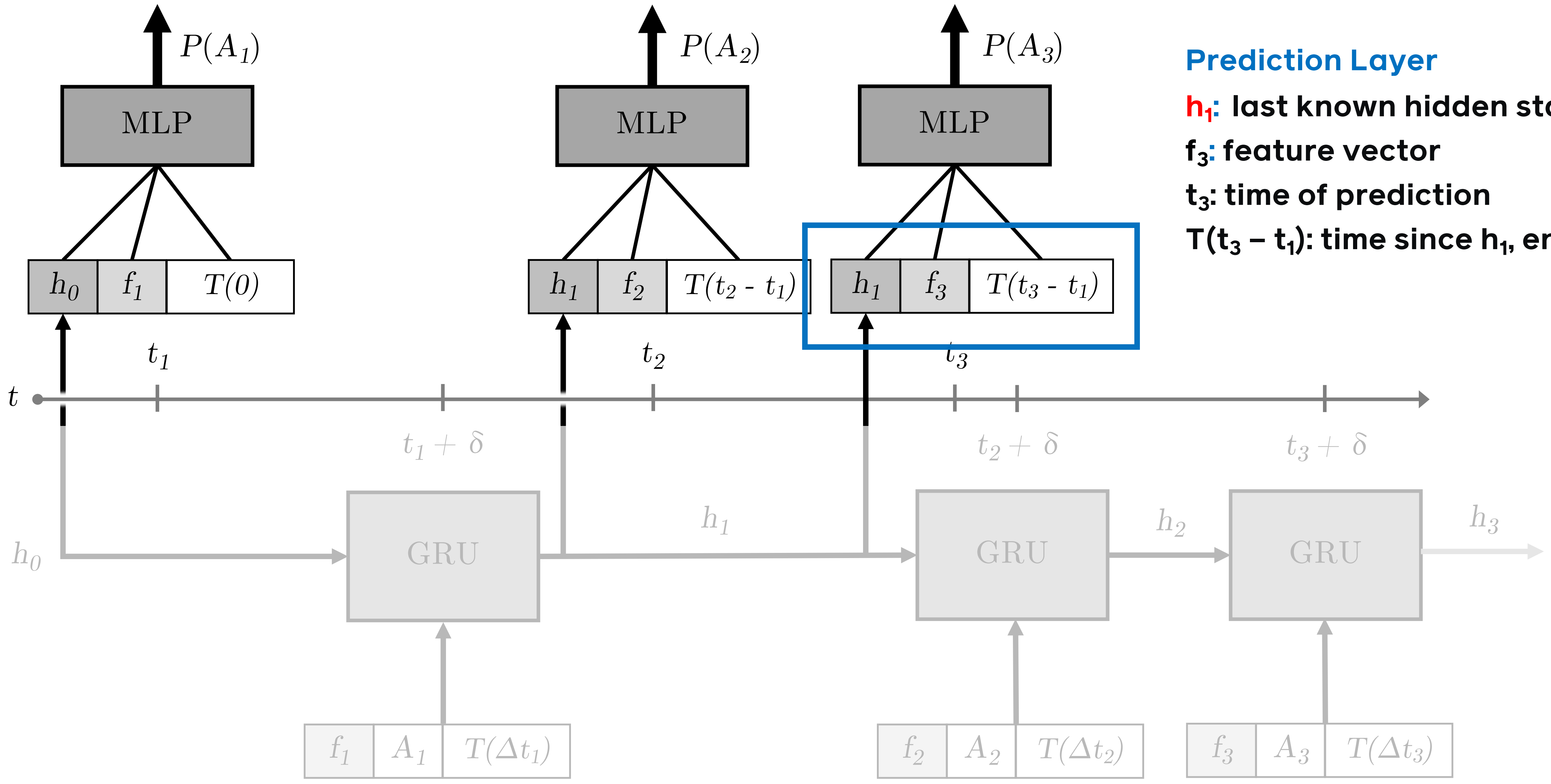
Recurrent Network Architecture



Recurrent Network Architecture



Recurrent Network Architecture



Prediction Layer

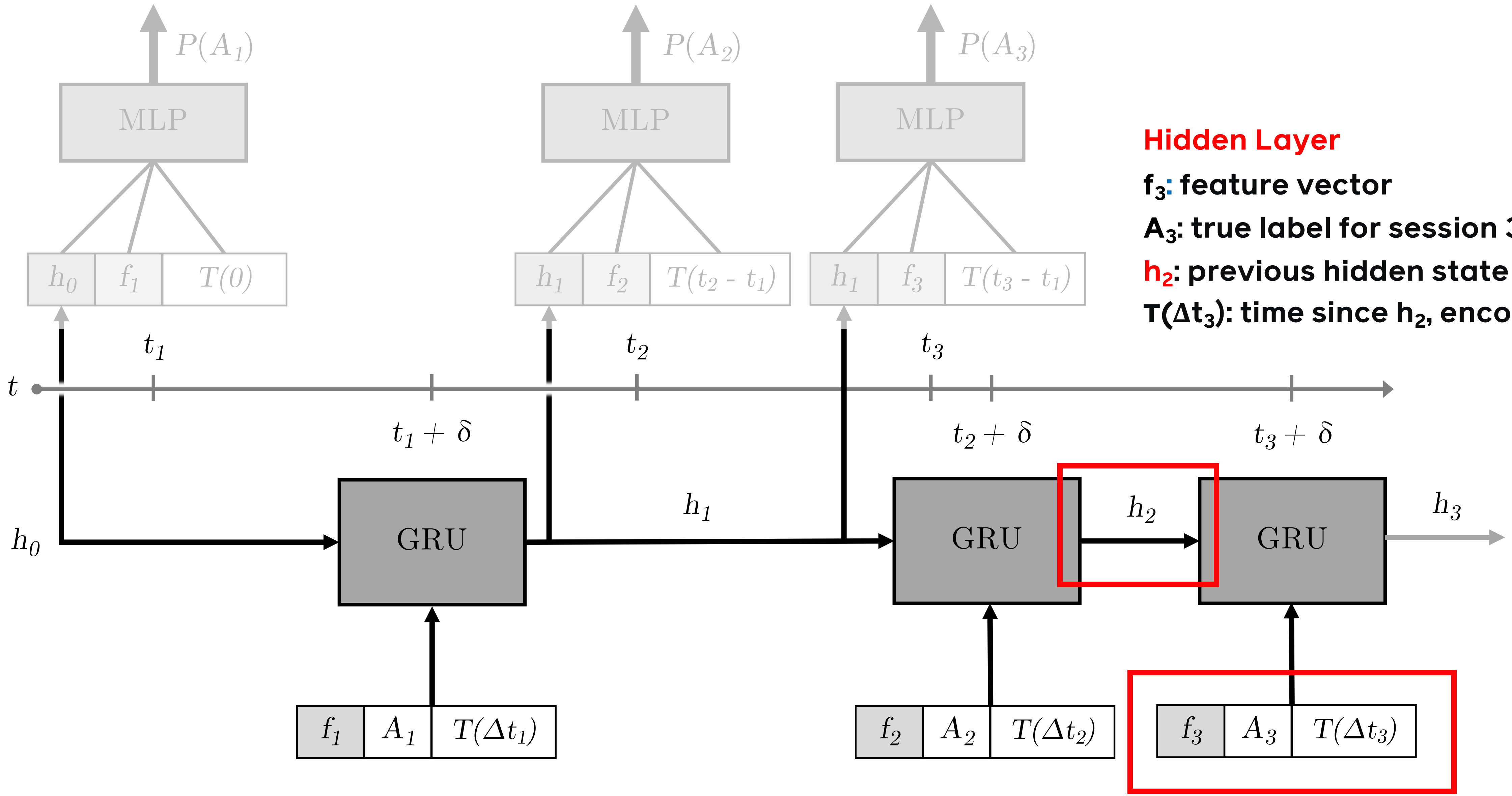
h_1: last known hidden state

f_3: feature vector

t_3: time of prediction

$T(t_3 - t_1)$: time since h_1, encoded

Recurrent Network Architecture



Hidden Layer

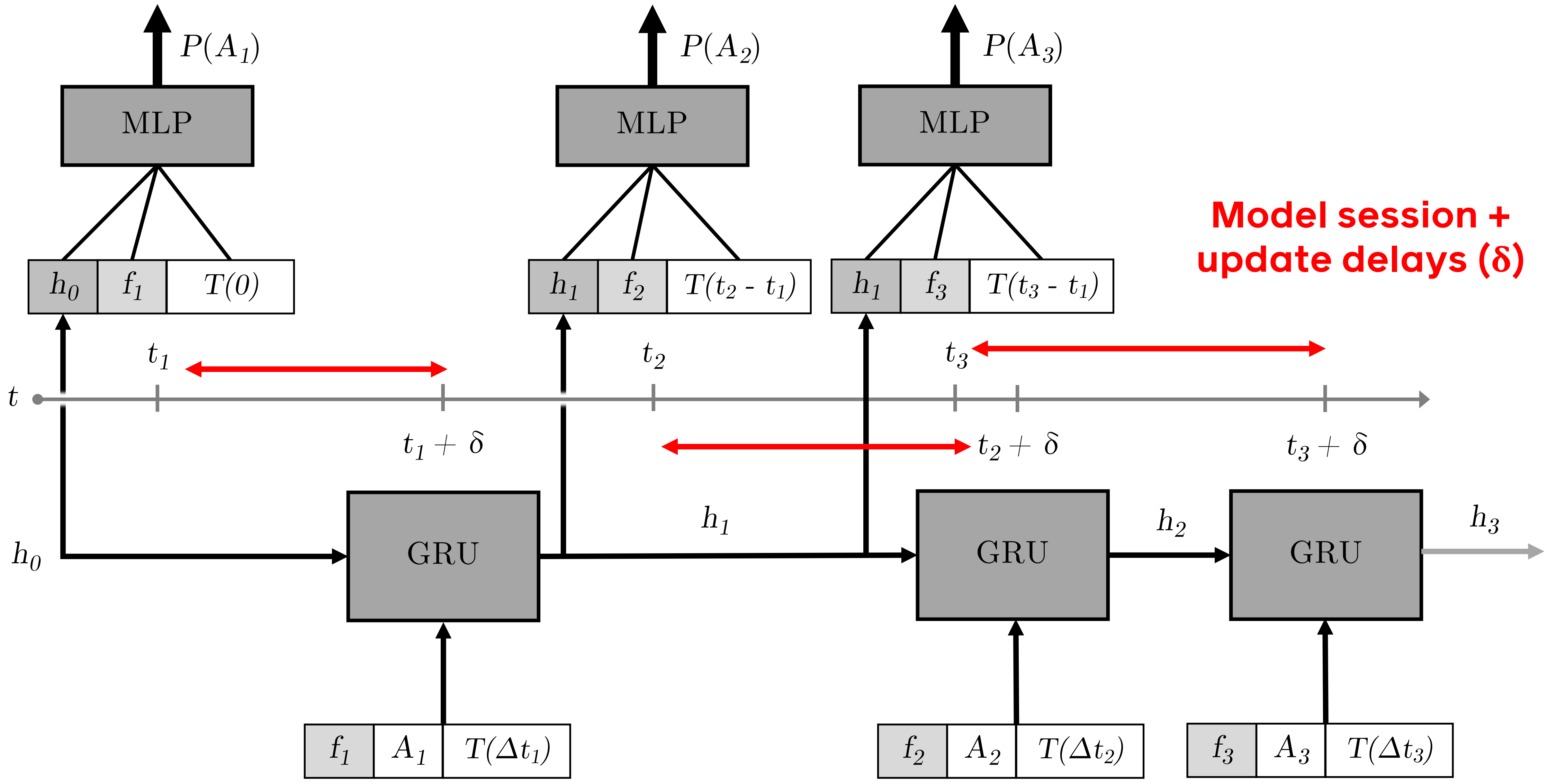
f_3: feature vector

A_3: true label for session 3

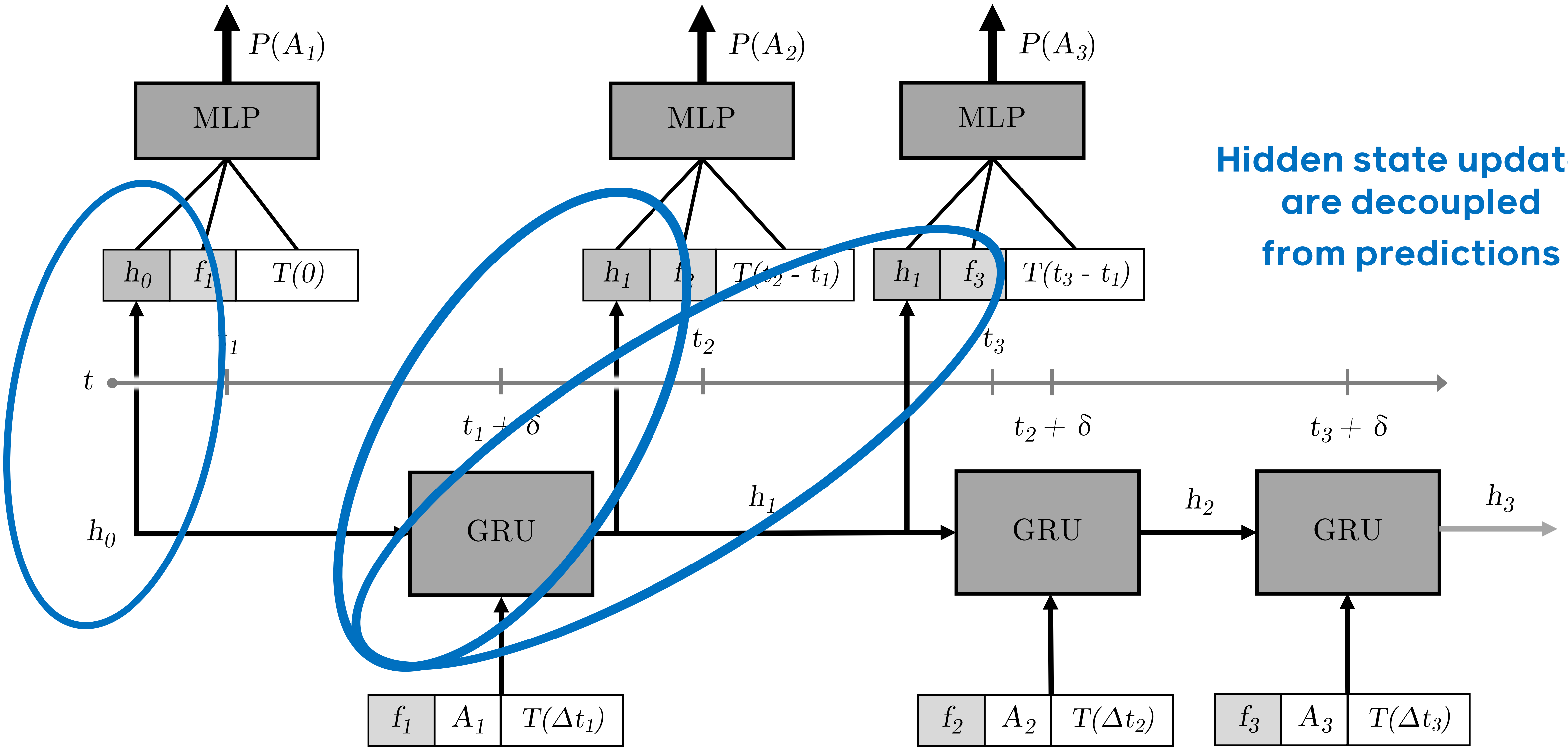
h_2: previous hidden state

$T(\Delta t_3)$: time since h_2, encoded

Recurrent Network Architecture

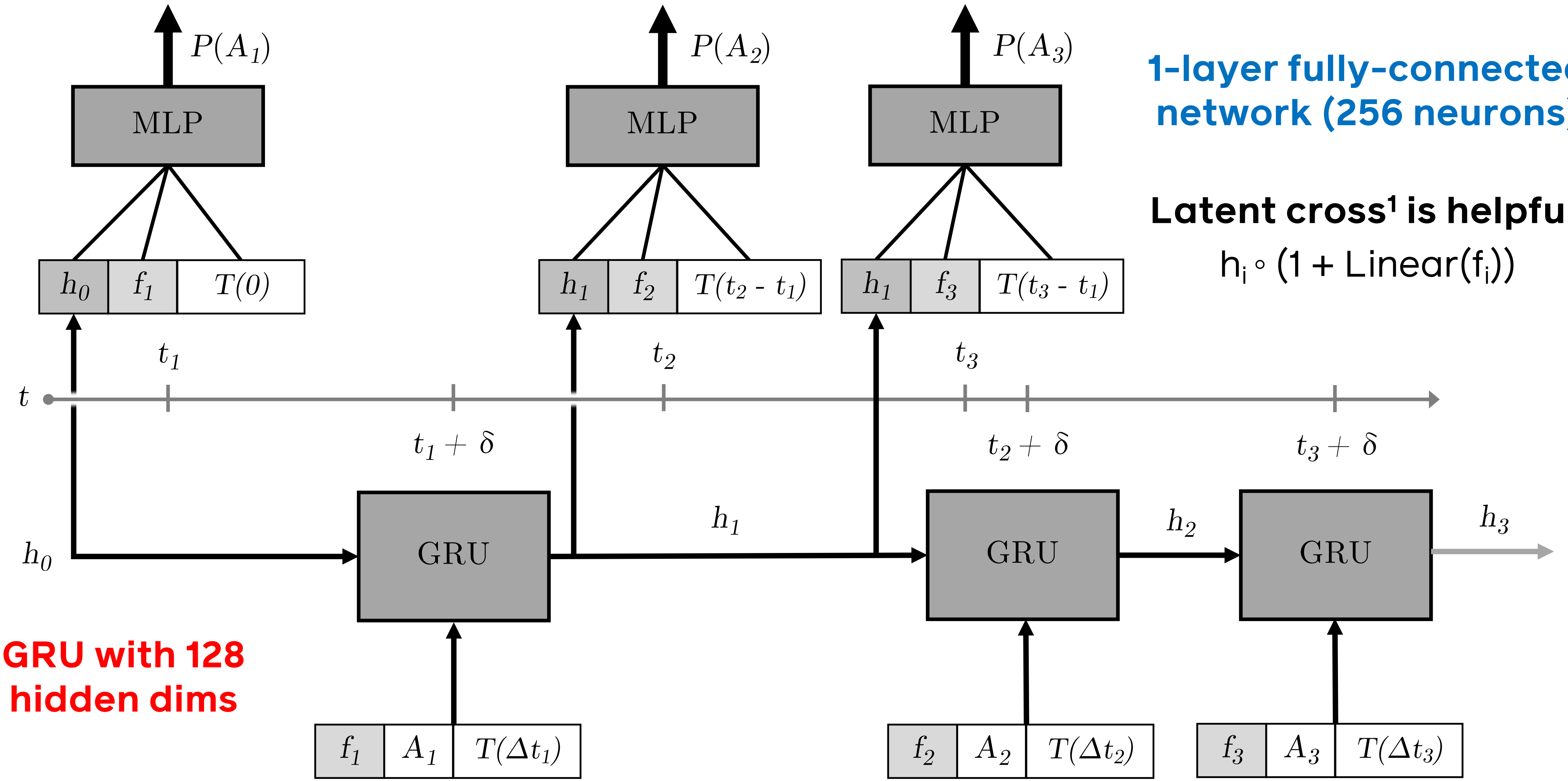


Recurrent Network Architecture



Hidden state updates are decoupled from predictions

Recurrent Network Architecture



1-layer fully-connected network (256 neurons)

Latent cross[1] is helpful:

$h_i \circ (1 + \mathrm{Linear}(f_i))$

GRU with 128 hidden dims

[1] Beutel, A., Covington, P., Jain, S., Xu, C., Li, J., Gatto, V., and Chi, E. H (2018). *Latent cross: Making use of context in recurrent recommender systems.*

Training details

- **1M user histories** over a **30 day** period
  - **~60 sessions per user on average, ~10% positive rate**
- Only compute loss on **last 21 days**
  - All evaluation metrics use **last 7 days**
- Training takes about **~8 hours on GPU** (PyTorch)
  - **Faster with BPPSA?**

Results

Precision and Recall for Precompute

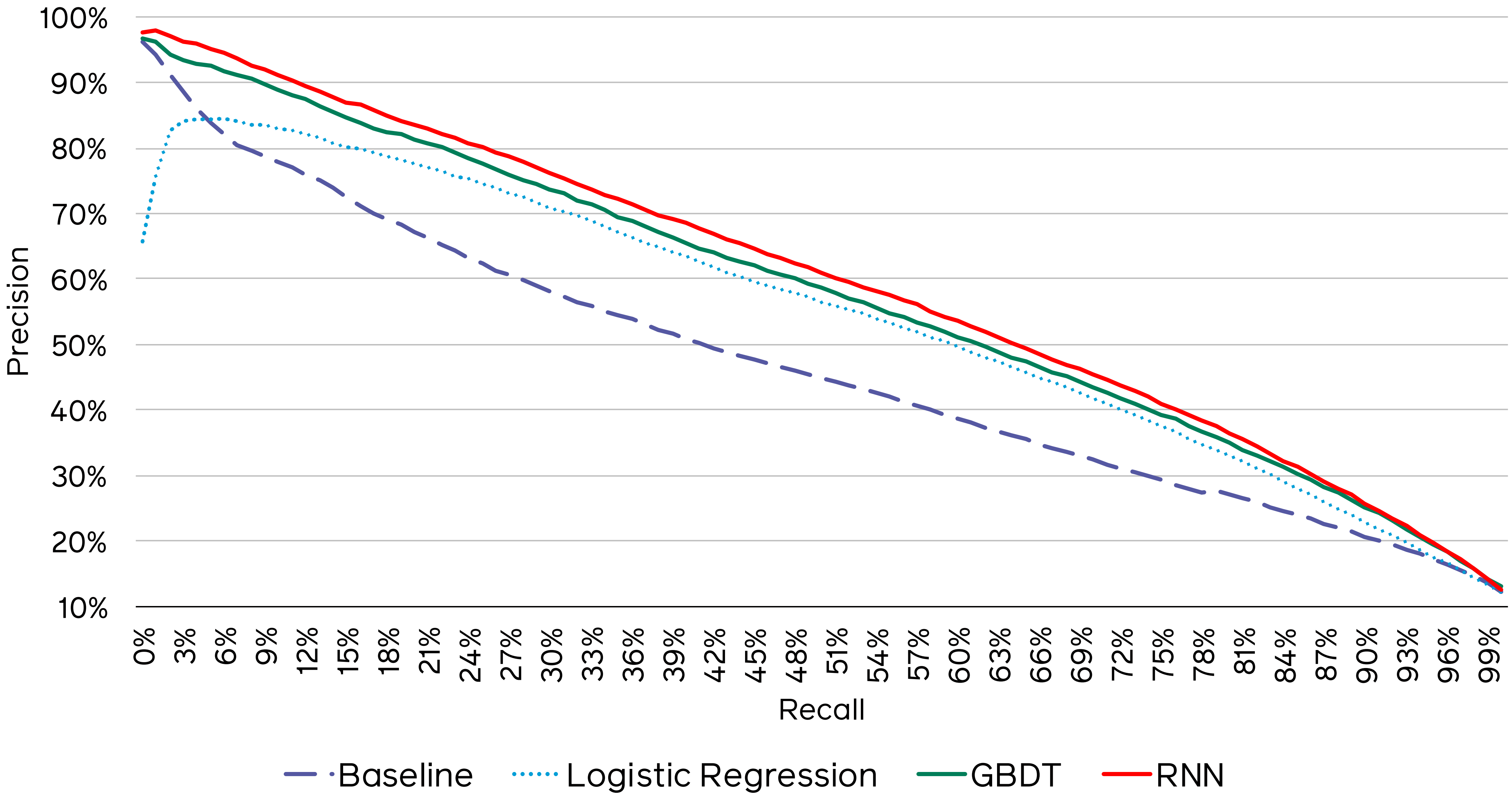
Precision: (true positives) / (predicted positives)

- What percentage of precomputed results are accessed?
- Inversely correlated to additional compute cost.

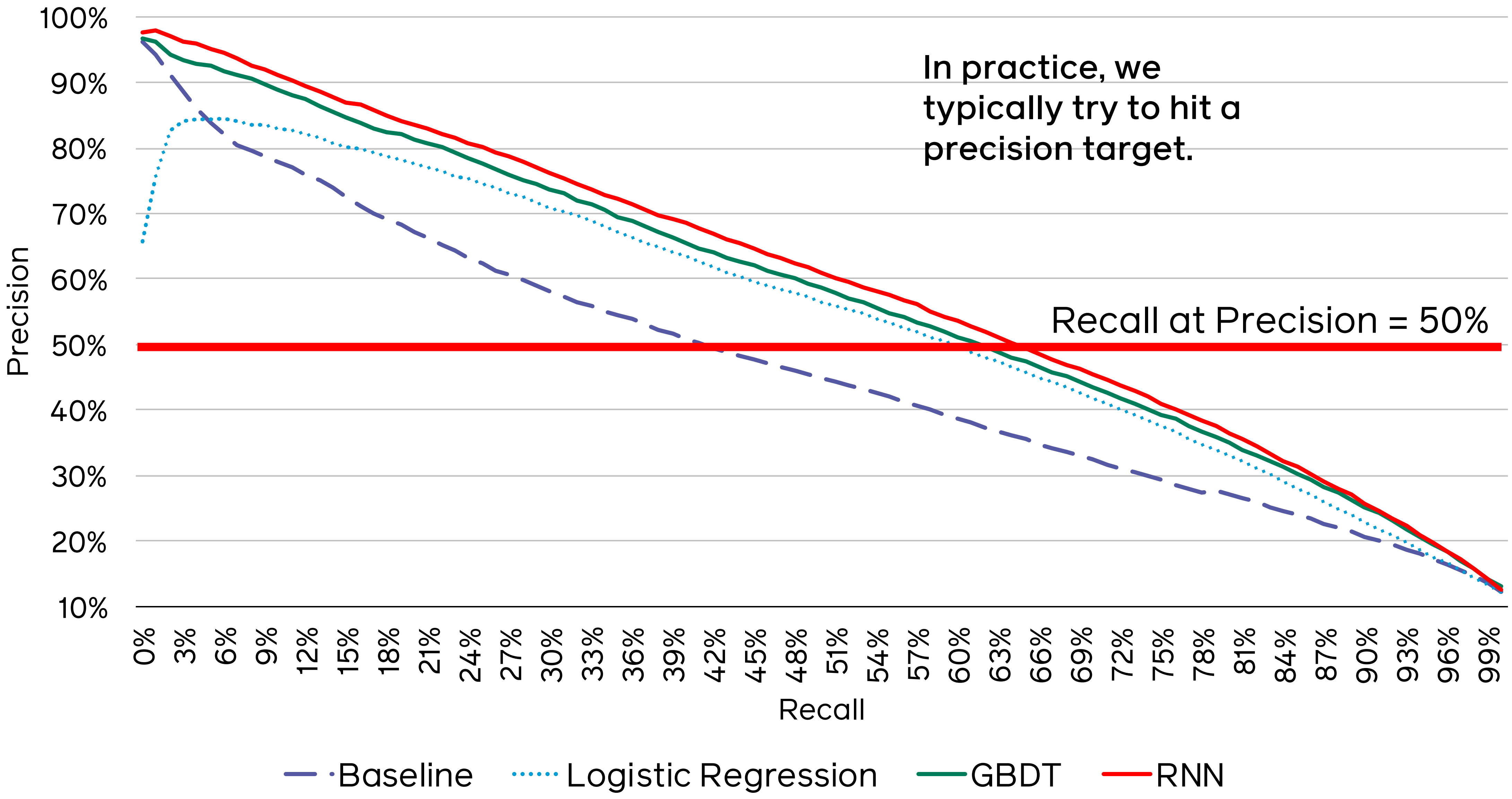
Recall: (true positives) / (total positives)

- What percentage of accesses used precomputed results?
- Directly correlated to product latency improvements.

Precision-Recall Curves: FB Mobile Tab



Precision-Recall Curves: FB Mobile Tab



Numerical comparison: FB Mobile Tab

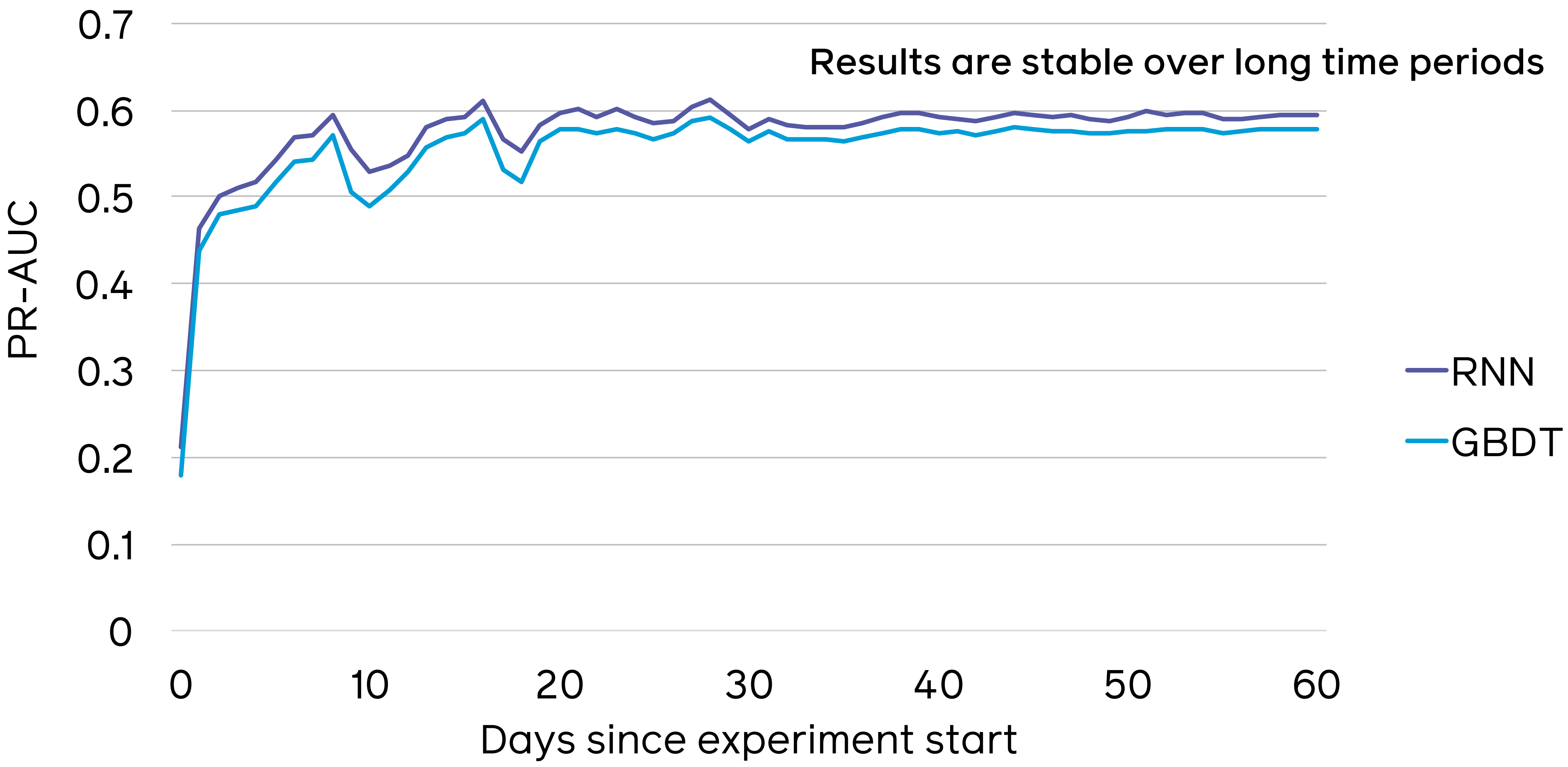
Model Type	PR-AUC	R@50%
Baseline	0.470	0.413
Logistic Regression	0.546	0.596
GBDT	0.578	0.616
Recurrent Neural Network	**0.596**	**0.642**
Improvement	3.11%	4.22%

Numerical comparison: Mobile Phone Use[2]

Public benchmark from Pielot, M., Cardoso, B., Katevas, K., Serra, J., Matic, A., and Oliver, N (2017).
Beyond interruptibility: Predicting opportune moments to engage mobile phone users.

Model Type	PR-AUC	R@50%
Baseline	0.591	0.811
Logistic Regression	0.683	0.906
GBDT	0.686	0.917
Recurrent Neural Network	**0.767**	**0.977**
Improvement	**11.8%**	**6.54%**

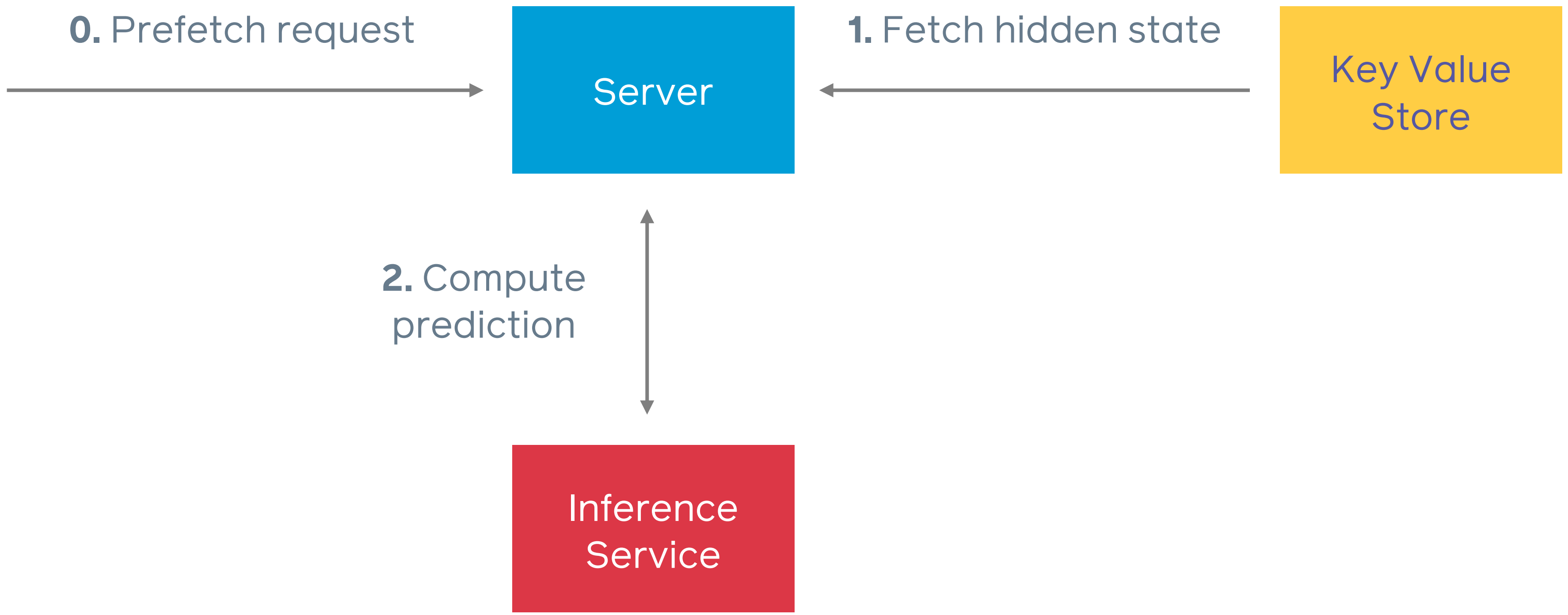
Online Testing



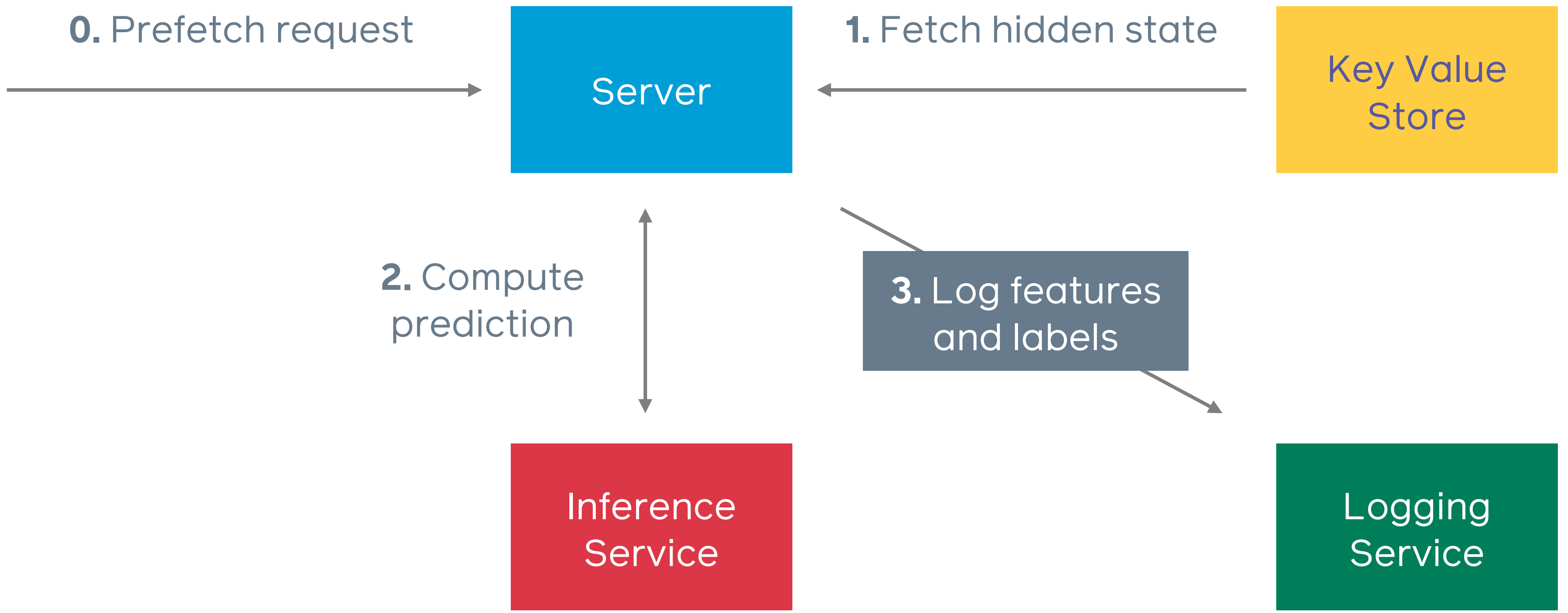
System Architecture

System Architecture

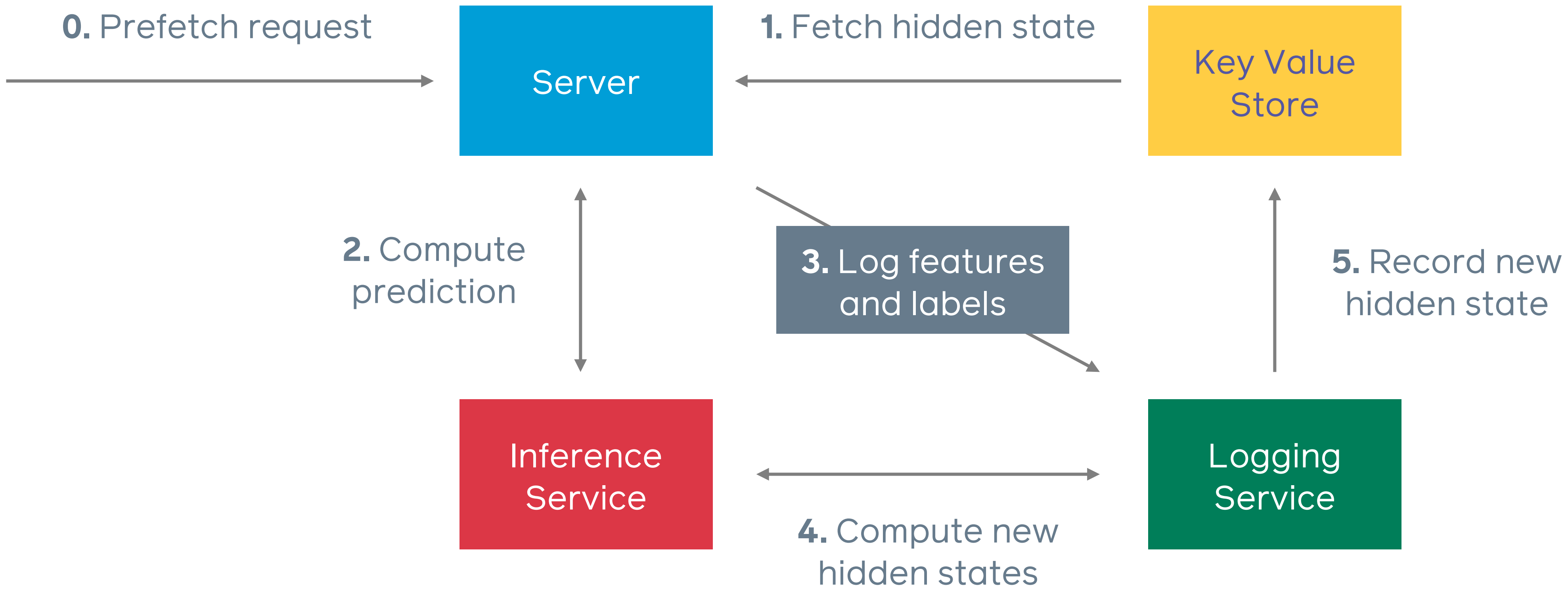
System Architecture



System Architecture



System Architecture



Traditional Methods

- Manually engineered features
- 10-100s of aggregation feature lookups per prediction
- Multiple KBs of storage required per user
- **~0.1ms model latency**

RNN Method

- **Minimal feature engineering**
- **1 key-value lookup per prediction**
- **Tunable (128 dim ~= 0.5KB) small storage cost per user**
- ~1ms model latency

10x overall reduction in compute costs

Summary

Precompute tasks, like application prefetching and cache warmup, can be modeled well through ML

Recurrent neural networks achieve superior modeling performance while reducing feature engineering time

RNNs also have surprisingly favorable characteristics when used in large-scale systems

Thank you

FACEBOOK