Lecture 11: Production Realities in Bandits

ID 6002W: Online and Reinforcement Learning

Web-enabled M.Tech. program, IIT Madras

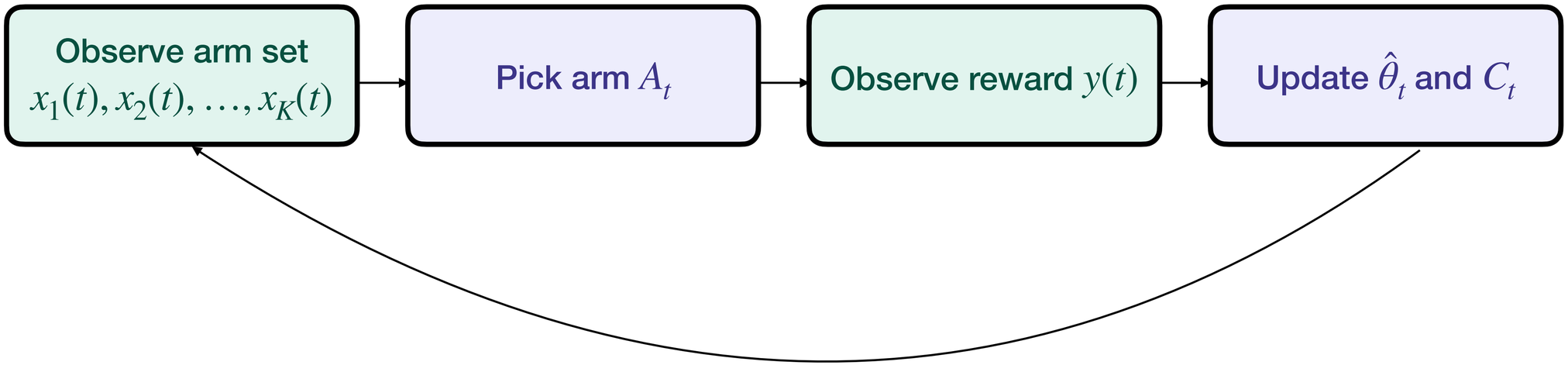
Suryanarayana Sankagiri

16th June, 2026

Homework and Exam Logistics

- HW 2 due on Wednesday, 17 June, 11:59 PM
- HW 3 will be released on Thursday, 18 June
  - It will cover linear bandits: Lectures 7 - 9
  - HW 3 due on Tuesday, 30 June, 11:59 PM
  - Please complete it as early as possible—useful for exam!
- Exam on Sunday, 28 June
  - Exam largely covers Lectures 1 - 9, very light on the rest.

Recap of Linear Bandits



Estimate θ with uncertainty quantified $\rightarrow$ act optimistically to balance exploration and exploitation

Some Critical Failure Points

In practical systems

- The LinUCB/LinTS model and algorithm is clean
  - But real-world systems are not!

- $A_t = \arg\max_{i \in [K]} \langle \hat{\theta}_{t-1}, x_i \rangle + \sqrt{\beta} \|x\|_{V_{t-1}^{-1}}$
  - computation scales linearly with number of arms

Some Critical Failure Points

In practical systems

- The LinUCB/LinTS model and algorithm is clean
  - But real-world systems are not!

- $A_t = \arg\max_{i \in [K]} \langle \hat{\theta}_{t-1}, x_i \rangle + \sqrt{\beta} \|x\|_{V_{t-1}^{-1}}$
  - computation scales linearly with number of arms
    - Feasible with millions of items, in real-world recommender systems?
      - Product/ad-display must be quick!

Some Critical Failure Points

In practical systems

- The LinUCB/LinTS model and algorithm is clean
  - But real-world systems are not!

- We minimise cumulative regret, equivalently maximise cumulative reward $\sum X_t$
  - But do we really observe what we want to maximise?

Some Critical Failure Points

In practical systems

- The LinUCB/LinTS model and algorithm is clean
  - But real-world systems are not!

- We minimise cumulative regret, equivalently maximise cumulative reward $\sum X_t$
  - But do we really observe what we want to maximise?
    - Maximising click-through rate will only promote click-bait content!
      - We need to think about reward design

Some Critical Failure Points

In practical systems

- The LinUCB/LinTS model and algorithm is clean
  - But real-world systems are not!
- We assume reward $y(t) = \langle x(t), \theta^* \rangle + \eta_t$ is immediately available

Some Critical Failure Points

In practical systems

- The LinUCB/LinTS model and algorithm is clean
  - But real-world systems are not!

- We assume reward $y(t) = \langle x(t), \theta^* \rangle + \eta_t$ is immediately available
  - But what we care about may not be immediately observable
    - E.g., long-term engagement on a podcast platform, patient recovery to a drug
      - How can bandits work with delayed feedback?

Some Critical Failure Points

In practical systems

- The LinUCB/LinTS model and algorithm is clean
  - But real-world systems are not!

- We assume reward $y(t) = \langle x(t), \theta^* \rangle + \eta_t$, where θ^* is unknown but fixed

Some Critical Failure Points

In practical systems

- The LinUCB/LinTS model and algorithm is clean
  - But real-world systems are not!

- We assume reward $y(t) = \langle x(t), \theta^* \rangle + \eta_t$, where θ^* is unknown but fixed
  - I.e., we assume a stationary world/environment
    - Real world data is not necessarily stationary
      - How can we adapt to time-varying environments?

Some Critical Failure Points

In practical systems

- The LinUCB/LinTS model and algorithm is clean
  - But real-world systems are not!
- Today, we will look at some principled solutions to these four problems
- Not an exhaustive list!
- Can you think of anything more?

Some Critical Failure Points

In practical systems

- The LinUCB/LinTS model and algorithm is clean
  - But real-world systems are not!
- Today, we will look at some principled solutions to these four problems
- Not an exhaustive list!
- Can you think of anything more?

As ML engineers, these are the kinds of challenges that will surface in your work

Handling Large Item Sets

Latency Requirements

100 ms latency in real-world systems

A practical requirement for smooth user experience

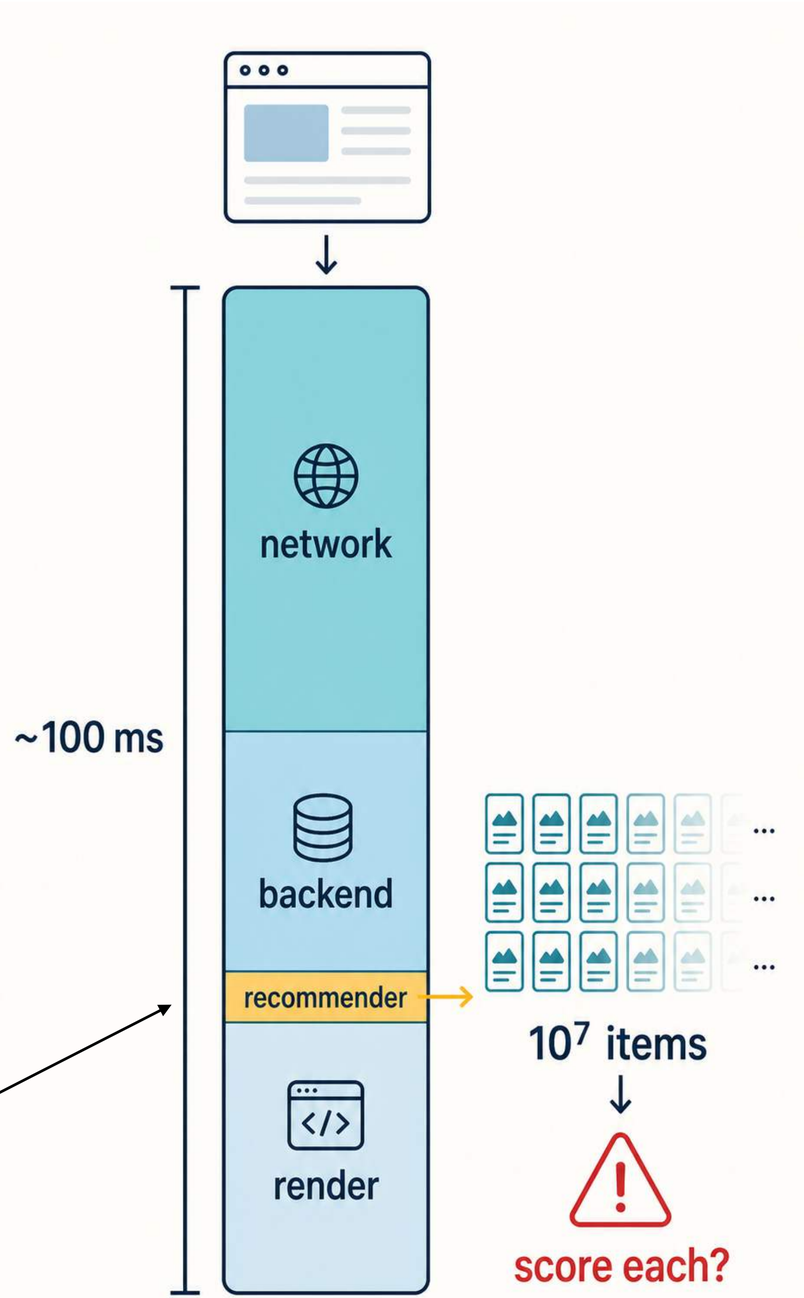
Computations for each arm:

- One matrix-vector multiplication + two dot products:

$$x_i^\top V_t^{-1} x_i + \langle \hat{\theta}_t, x_i \rangle$$

- Scales as d^2

Cannot do this for 10^7 arms in a few milliseconds



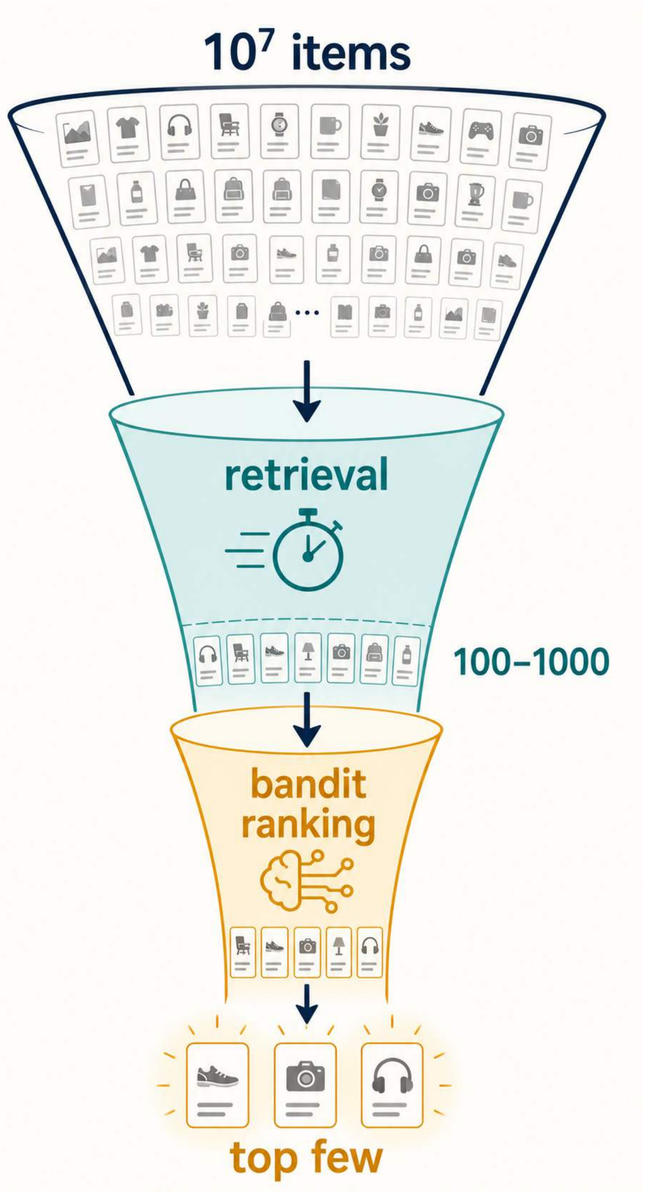
How Large Scale Systems Work

Stage 1: Retrieval (search) on full catalog

- Filters large catalog into moderate size
- $10^7 - 10^8$ items $\rightarrow$ $10^2 - 10^3$ items
- Based on search query or user feature

Stage 2: Linear bandit algo on smaller set

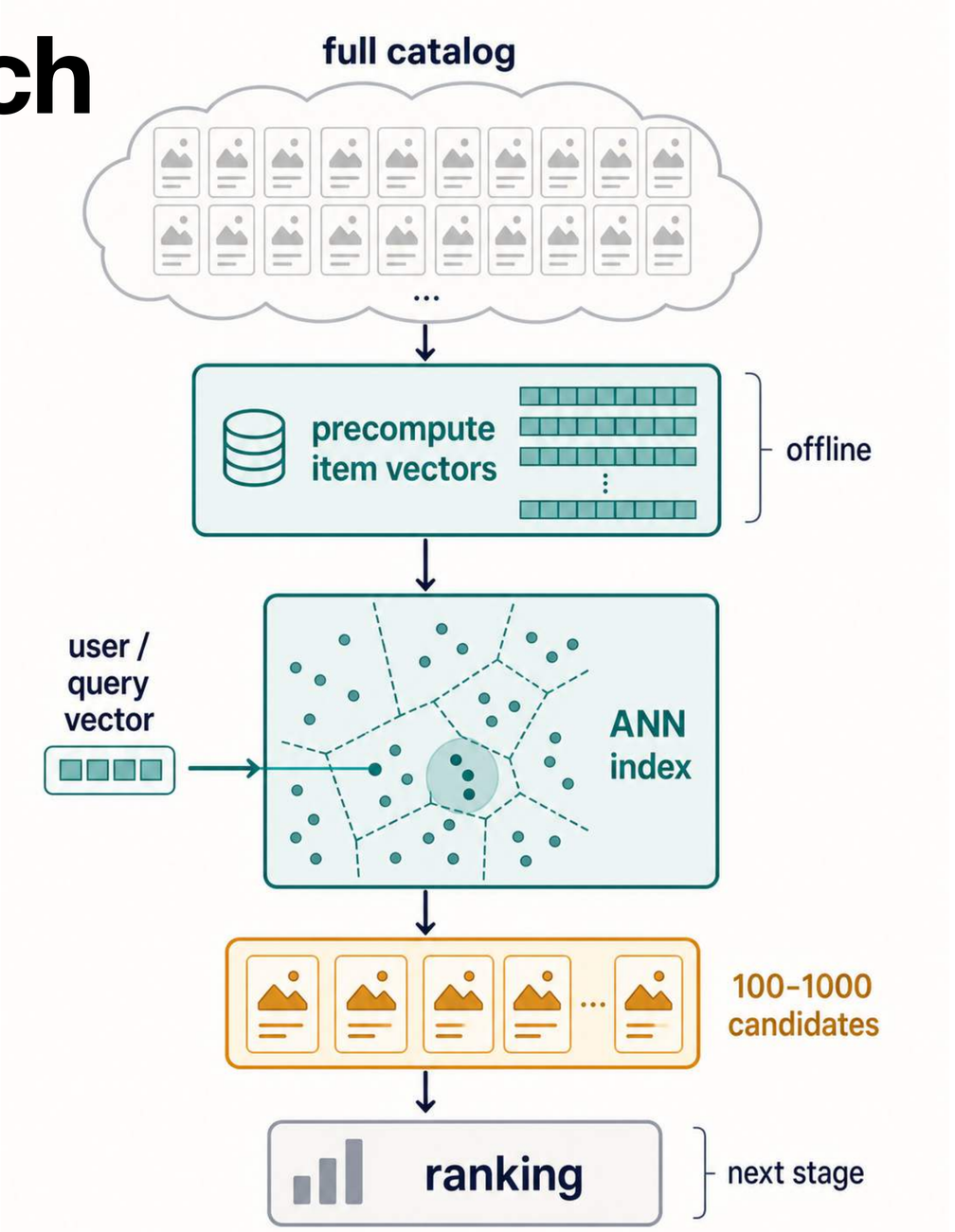
- Reduces computational load
- Bandit ranks the shortlist, not the catalog



Nearest Neighbour Search

Retrieval on Full Catalog

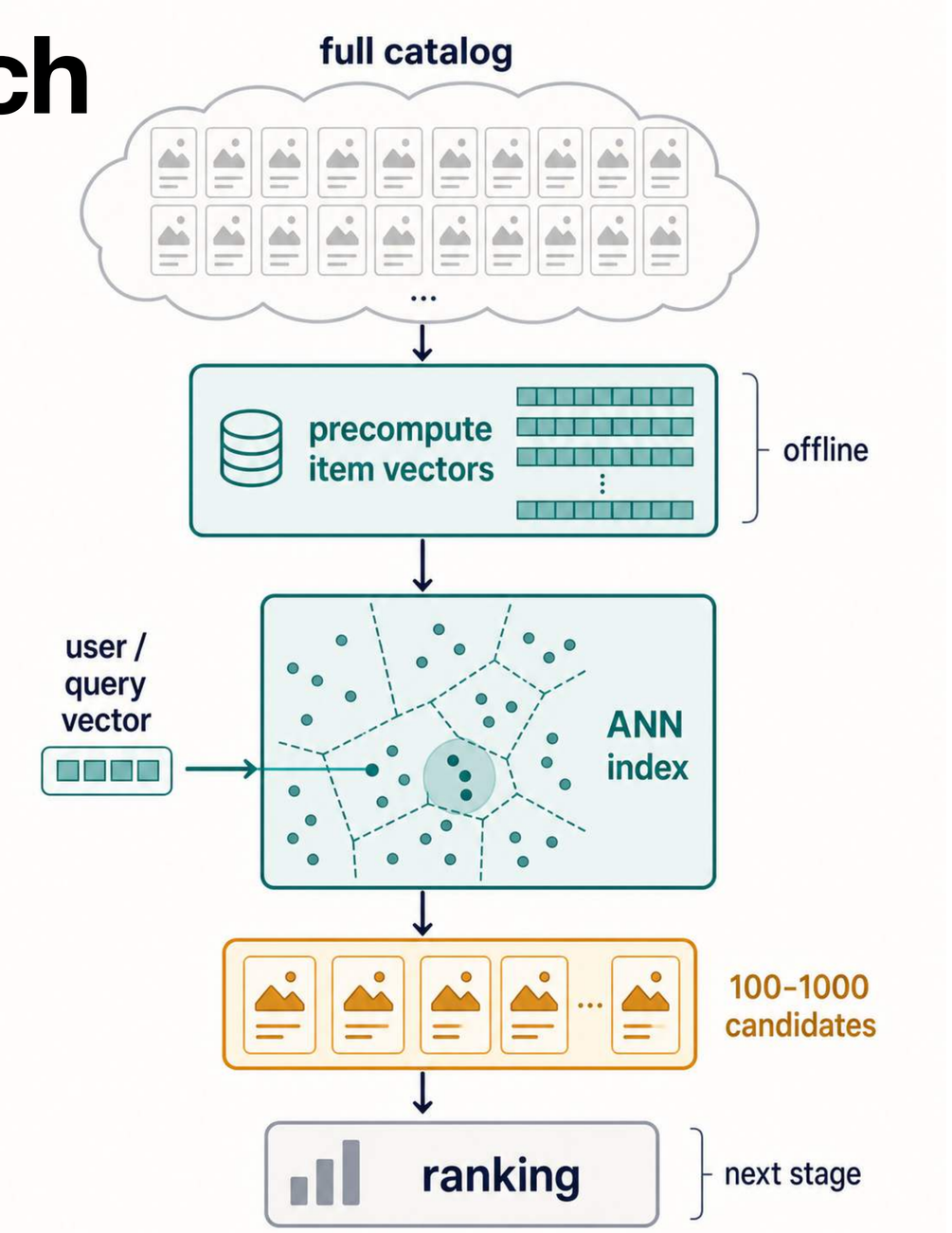
- Have pre-computed item embeddings
- Compute query/user embeddings online



Nearest Neighbour Search

Retrieval on Full Catalog

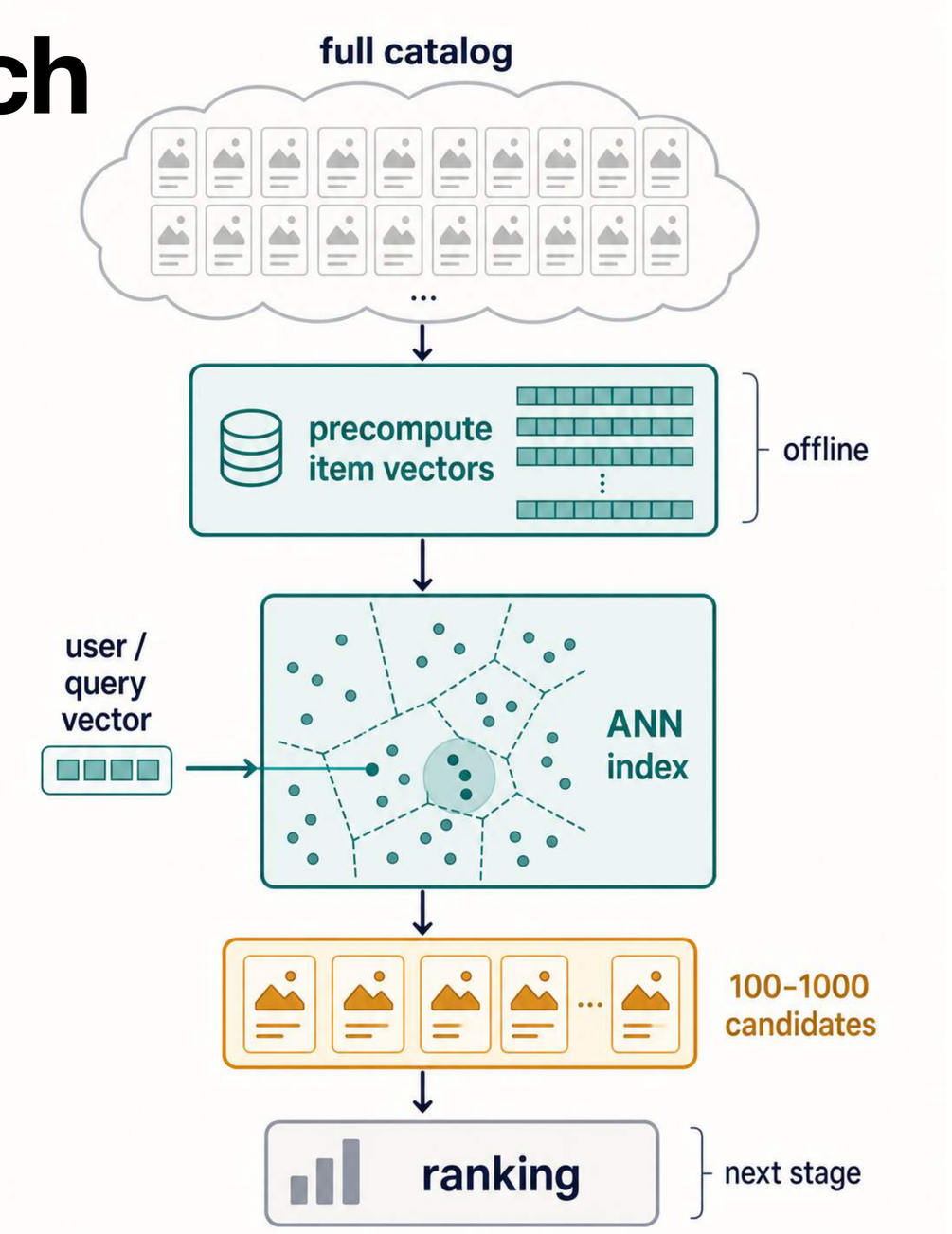
- Have pre-computed item embeddings
- Compute query/user embeddings online
- Compute *approximate nearest neighbours* to ‘target’ in embedding space
  - Avoid scoring every item
  - Exact ranking of items NOT important



Nearest Neighbour Search

Retrieval on Full Catalog

- Have pre-computed item embeddings
- Compute query/user embeddings online
- Compute *approximate nearest neighbours* to ‘target’ in embedding space
  - Avoid scoring every item
  - Exact ranking of items NOT important
- Returns $10^2 - 10^3$ items

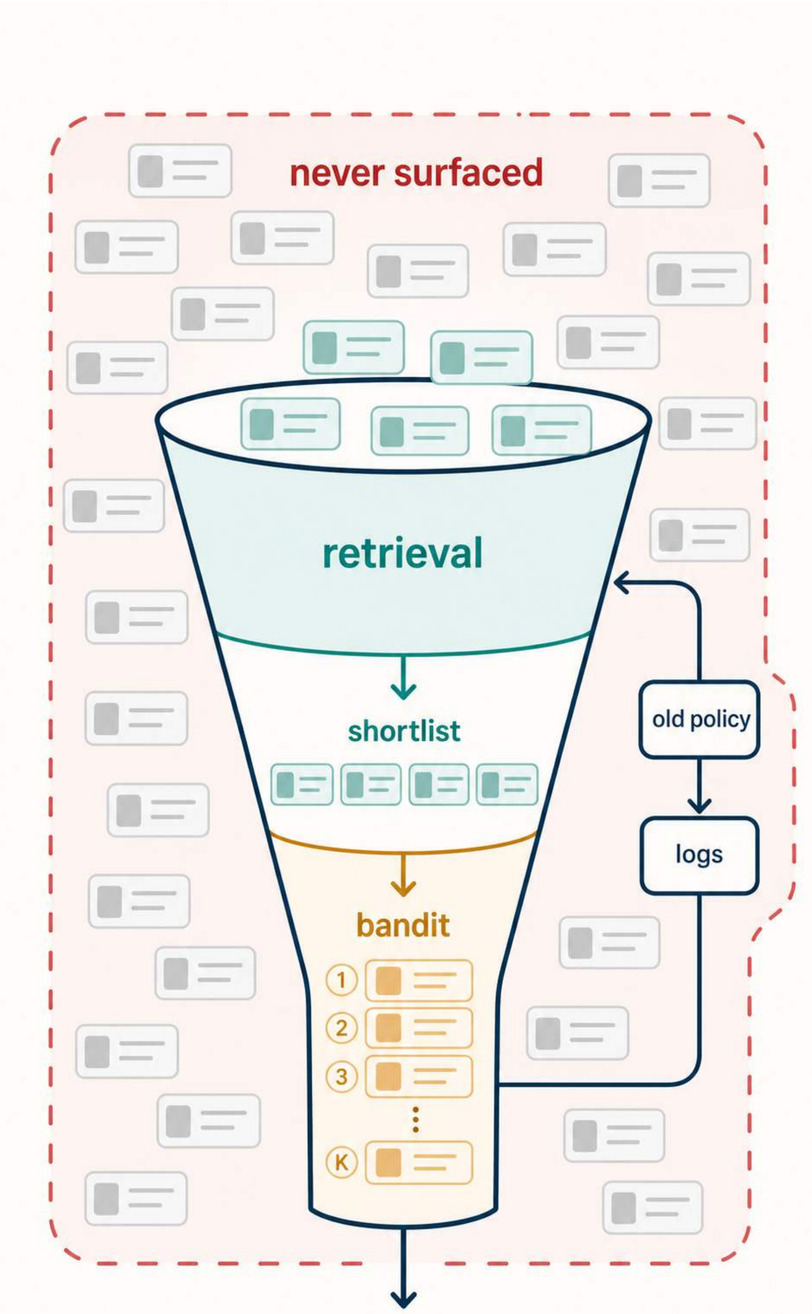


The Blind Spot

Exploration is limited by retrieval

For the bandit algorithm, the set of arms is whatever the retrieval gives us for this request

- If retrieval never surfaces an item, bandit never sees it ⇒ UCB/TS cannot help!

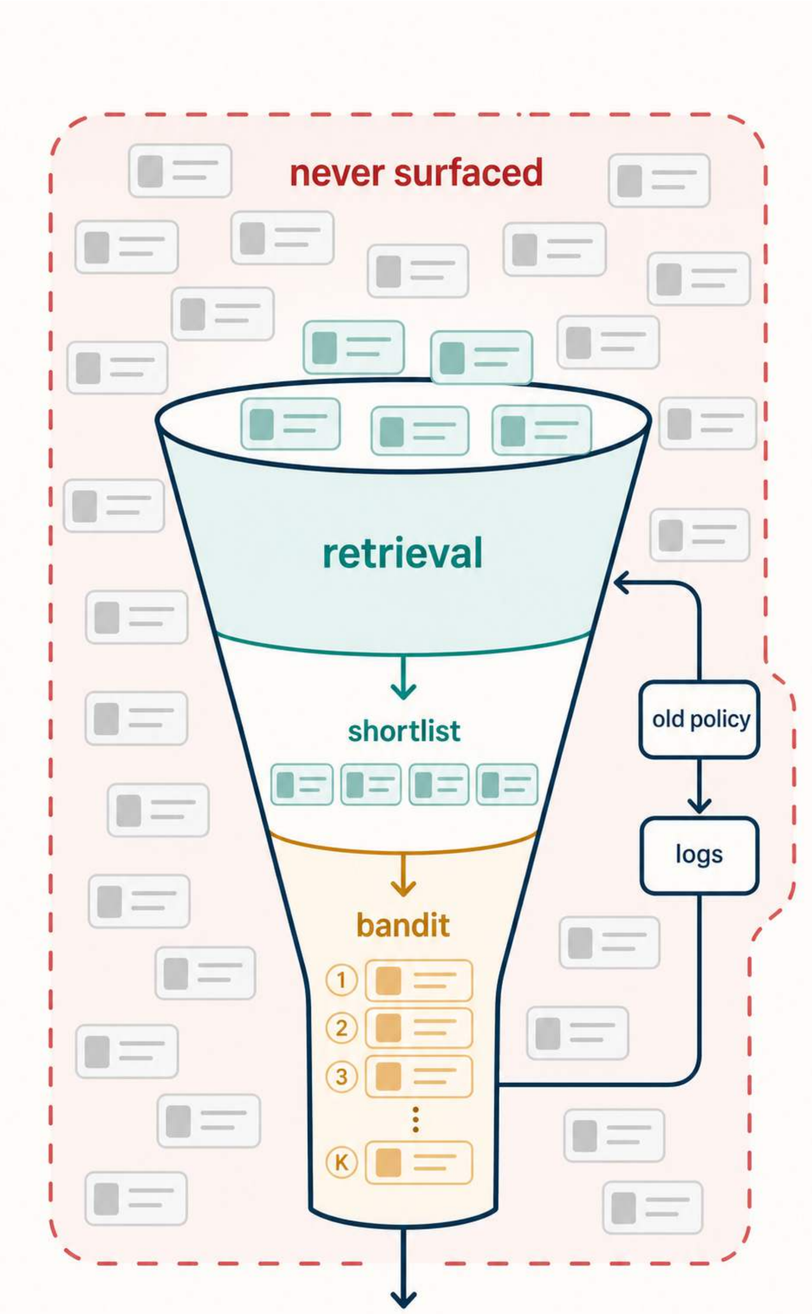


The Blind Spot

Exploration is limited by retrieval

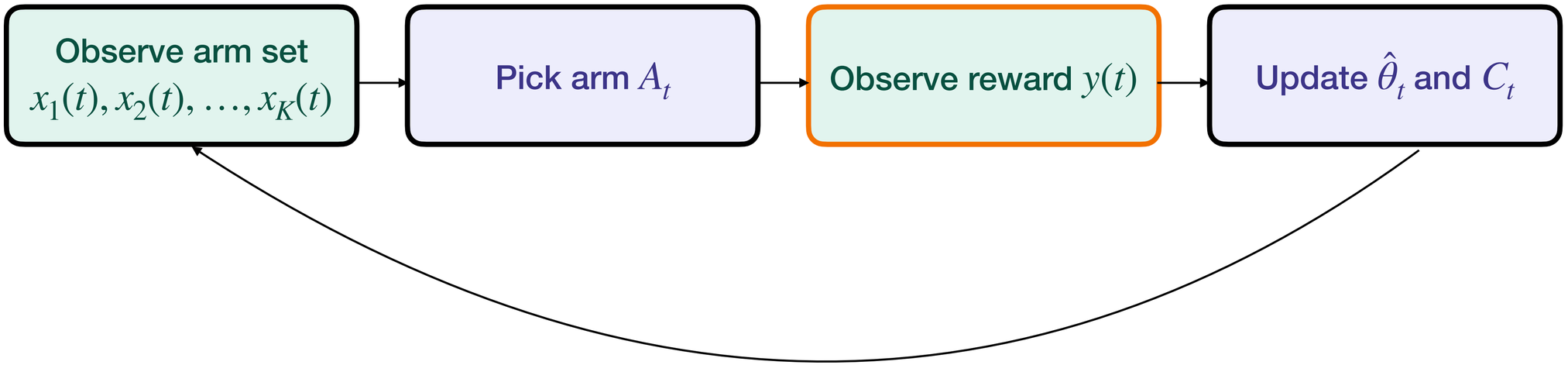
For the bandit algorithm, the set of arms is whatever the retrieval gives us for this request

- If retrieval never surfaces an item, bandit never sees it ⇒ UCB/TS cannot help!
- Retrieval system uses user/item embeddings → learned from past data
- Must ensure retrieval system explores enough



Reward Design

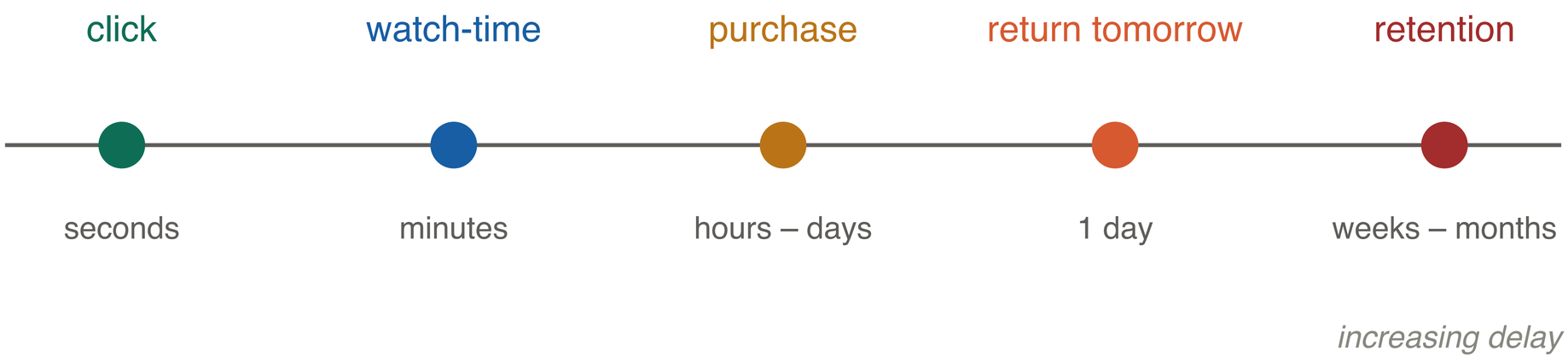
What Is The Reward, Actually?



- In a real-world system, we, the engineers, must define the reward
- Click? Watch-time? Purchase? Recurring Subscription? Not an easy answer!

Proxy Versus Value

Different timelines for different delays



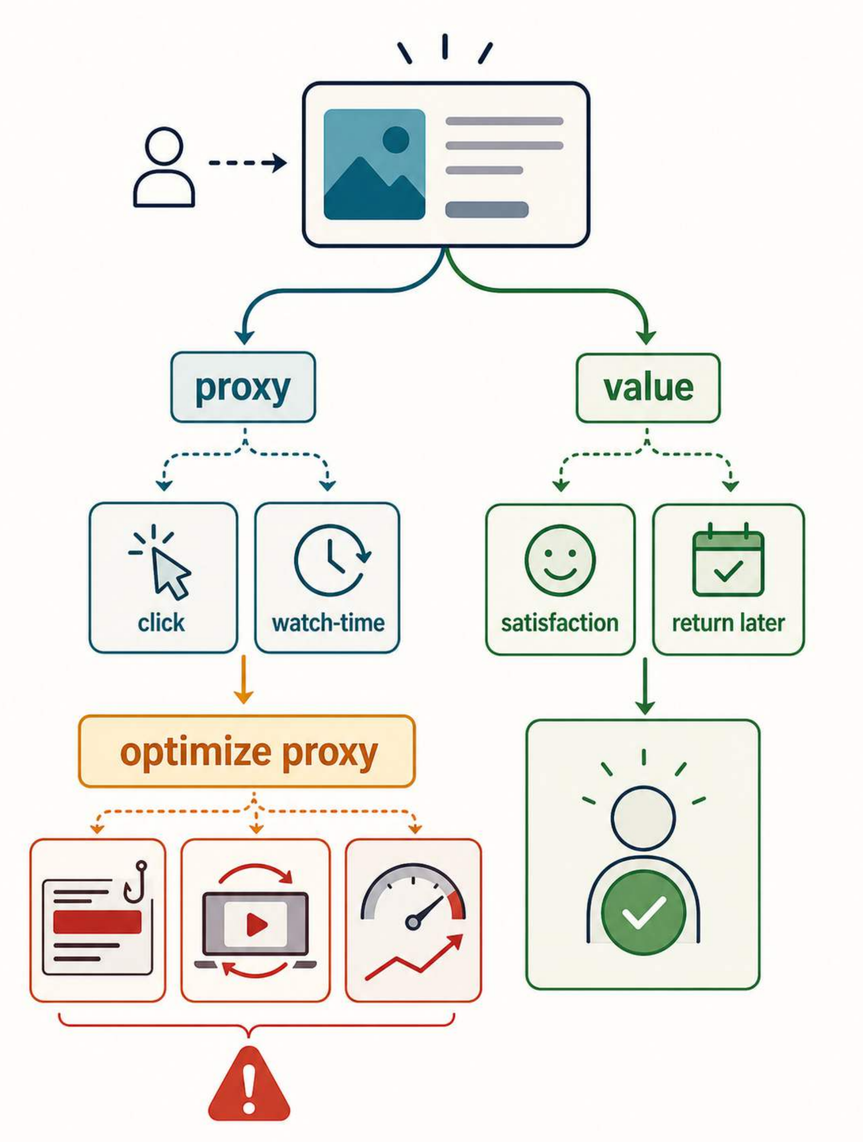
- Clicks, watch-time, completion are easy to observe
- But they are only proxies for user value

Proxy Rewards Can Be Dangerous

- The bandit optimises what we reward, not what we value

- If we optimise the proxy too hard, the system may learn the wrong behaviour

Goodhart's Law: When a measure becomes a target, it ceases to be a good measure

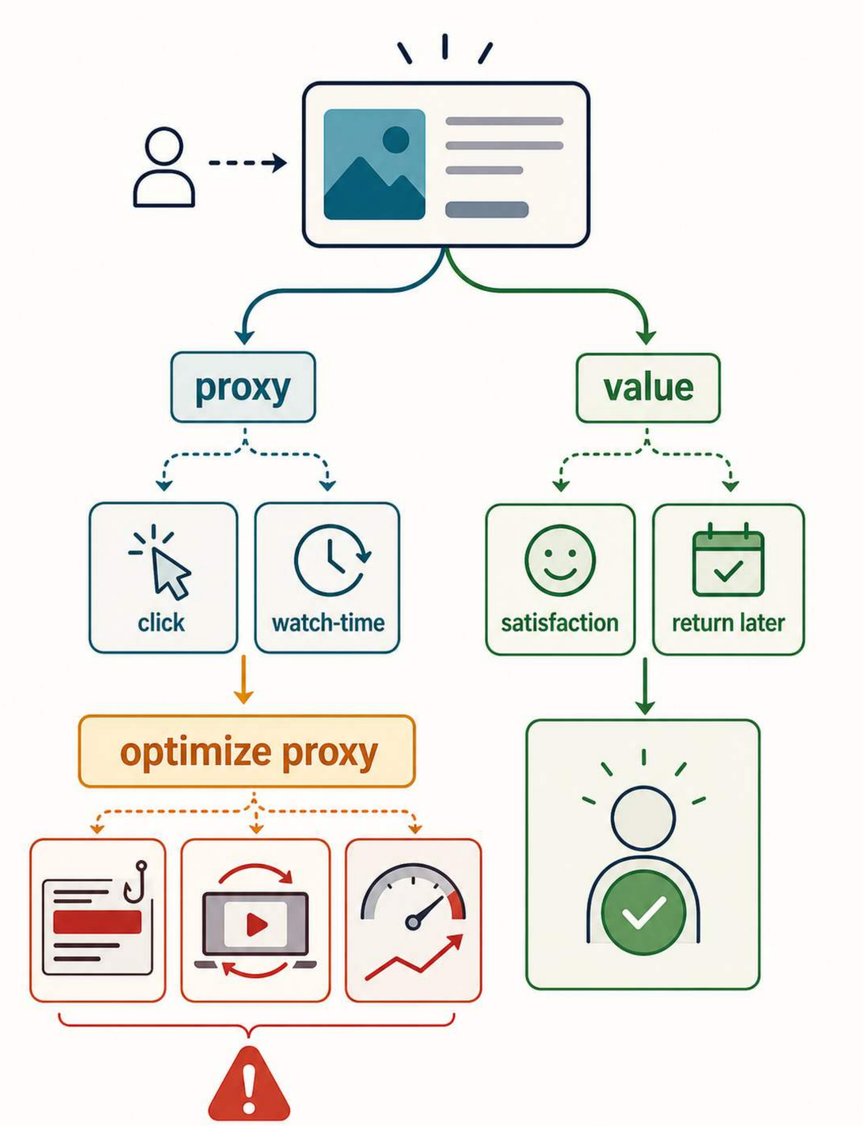


The Production Dilemma

- Fast proxy: available now, but misaligned
- True reward: better aligned, but delayed

- Waiting slows learning

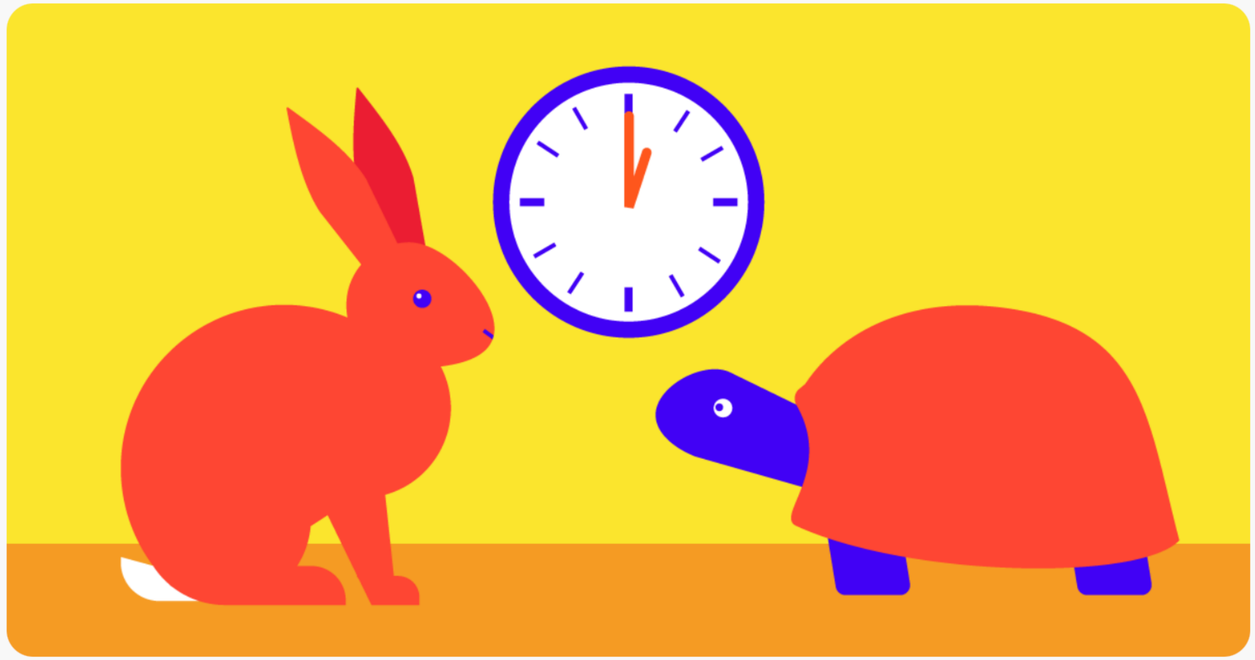
- Using only proxies can learn the wrong thing



A Case-Study From Spotify

R&D Blog Publications Research Areas Podcasts Jobs About Us

Optimizing for the Long-Term Without Delay



Jul 24, 2023

Published by Thomas M. McDonald, Lucas Maystre, Mounia Lalmas, Daniel Russo, Kamil Ciosek

ARTIFICIAL INTELLIGENCE

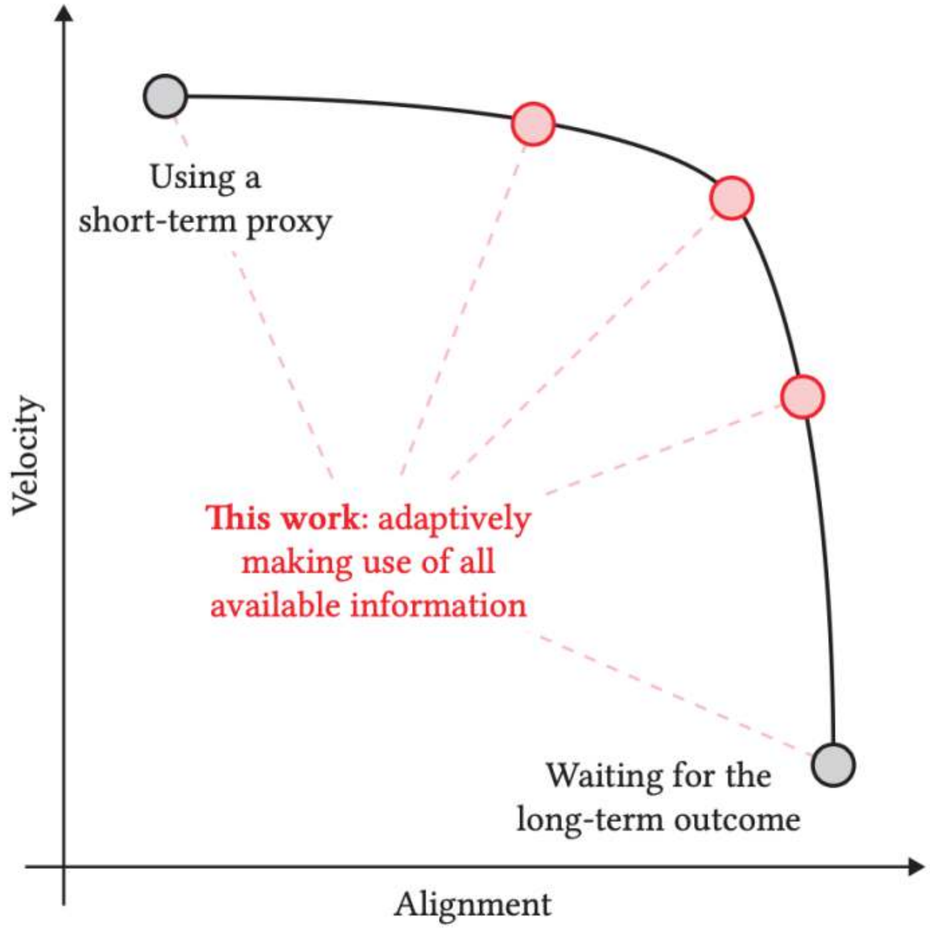
SEARCH & RECOMMENDATIONS

USER MODELING

SHARE THIS ARTICLE

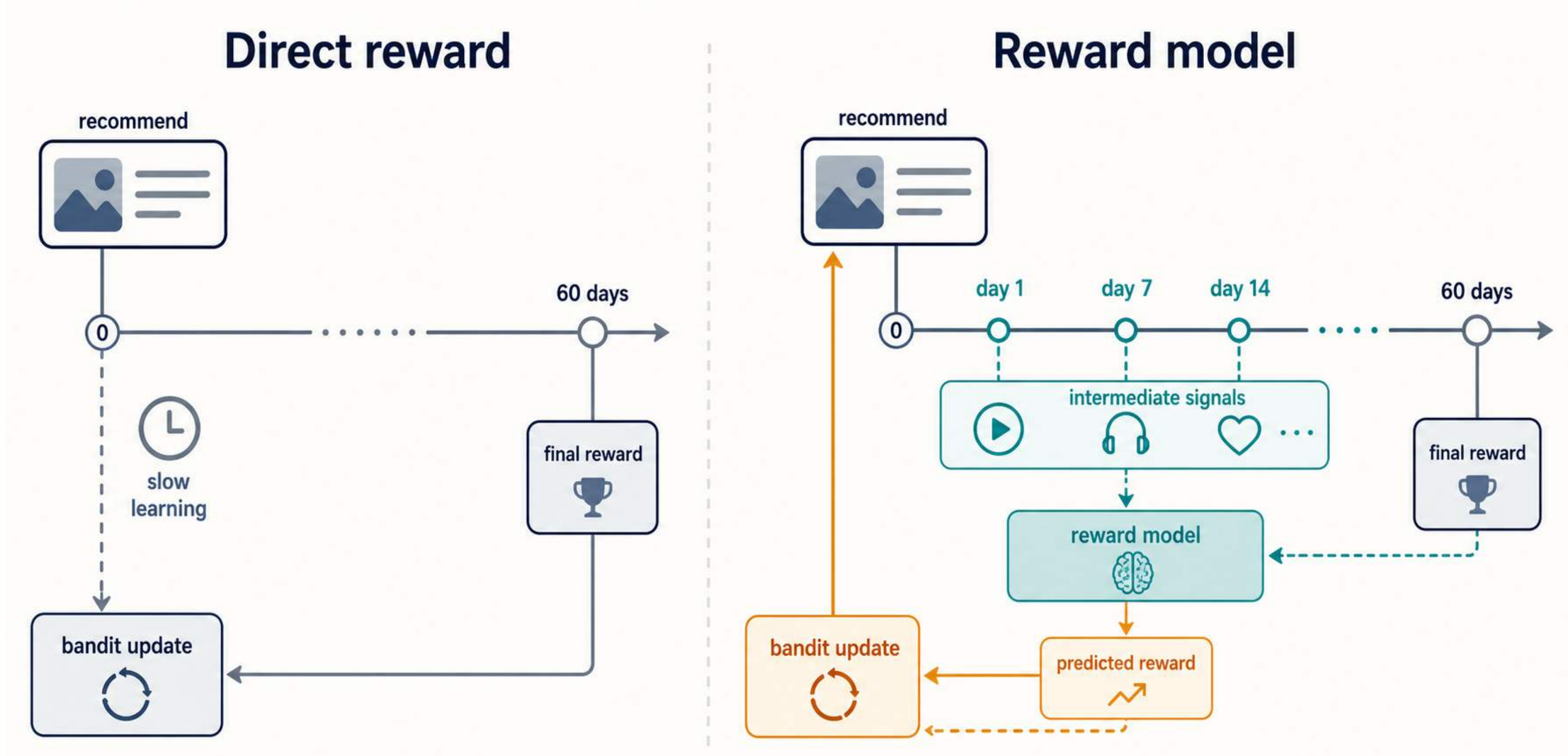
A Case-Study From Spotify

- Spotify wants to optimise for long-term user satisfaction → drives subscriptions
  - e.g., how engaging is a podcast over a *two-month period*
  - Not merely how many users have *started listening* to the podcast
- Can’t afford to wait for two months to observe reward!
  - Must use intermediate signals creatively



Using Reward Models

Estimating long-term reward from short-term signals



The New Bottleneck

- Bandit algorithm design challenge: exploration-exploitation tradeoff

  ✓This is now well understood (hopefully!)

The New Bottleneck

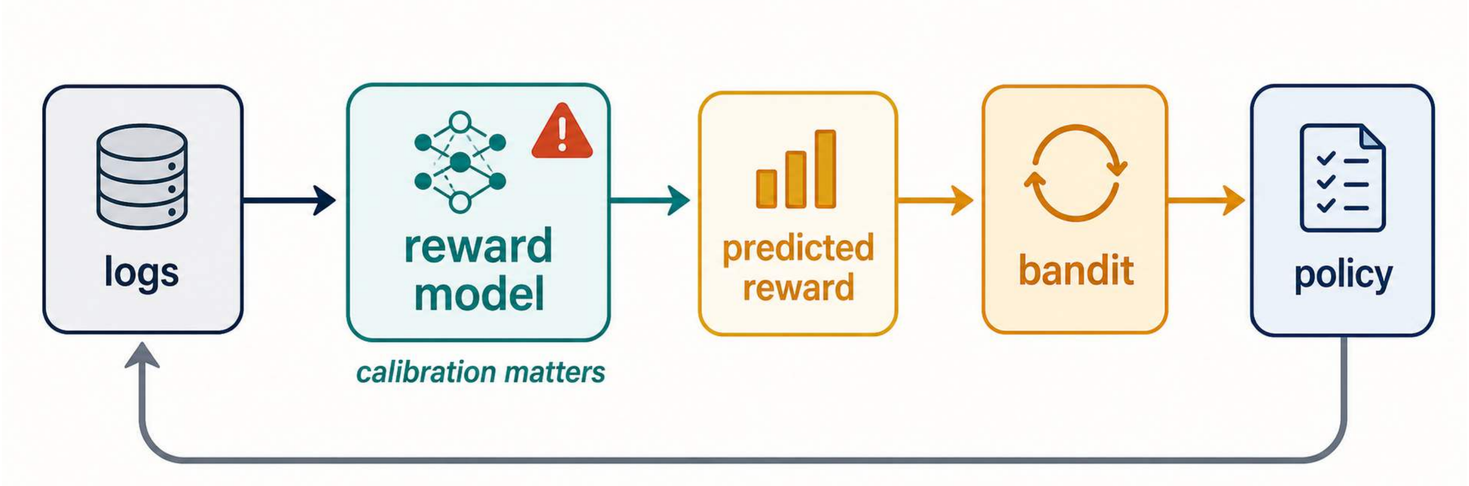
- Bandit algorithm design challenge: exploration-exploitation tradeoff

  ✓This is now well understood (hopefully!)

- Reward model design challenge: using available signals to predict true reward

  ✴Opportunity: can use multiple signals to estimate a rich reward signal

  ◉Risk: if the reward model learns the wrong thing, we create a biased policy



Non-Stationarity

The World Is Not Stationary

User interests drift

Old data may become misleading

We can change what counts as data

The World Is Not Stationary

User interests drift

Old data may become misleading

We can change what counts as data

- Discounted update: down-weight older observations
- Linear bandit version:

$$V_t = \lambda I + \sum_s \gamma^{t-s} x(s) x(s)^\top, \; b_t = \sum_s \gamma^{t-s} y(s) x(s)$$

The World Is Not Stationary

User interests drift

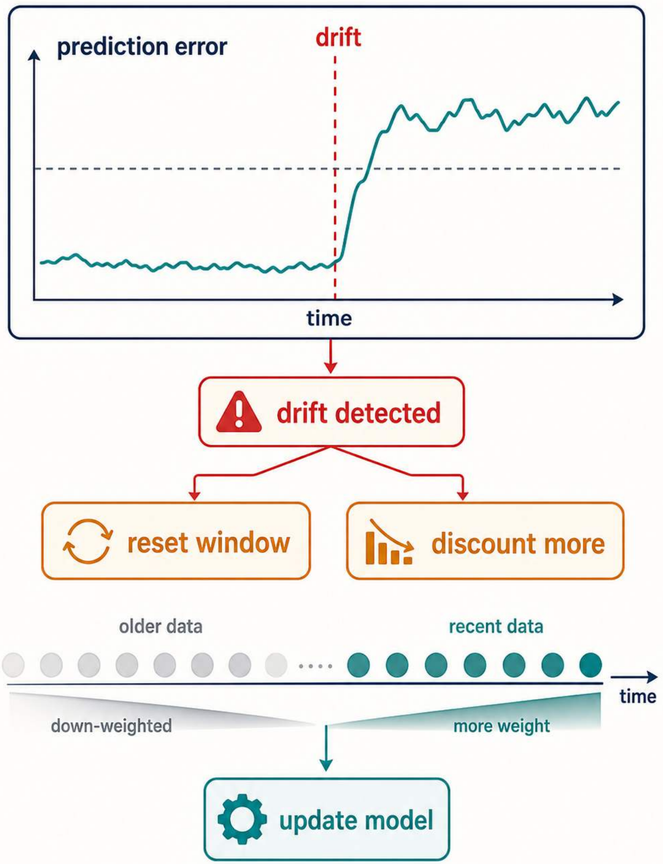
Old data may become misleading

We can change what counts as data

- Discounted update: down-weight older observations
- Linear bandit version:

$$V_t = \lambda I + \sum_s \gamma^{t-s} x(s)x(s)^\top,\ b_t = \sum_s \gamma^{t-s} y(s)x(s)$$

- Algorithm remains the same as before



Production reality: need to detect when model is drifting

Conclusion

We discussed the following challenges to applying (linear) bandits in practice:

- Large set of actions/arms, e.g., items in a recommender systems, makes it *computationally infeasible* to apply LinUCB directly
  - Solution: a two-stage system: a fast retrieval algorithm followed by the actual bandit algorithm
- The true quantity that we want to optimise, is often impossible to measure (user satisfaction), or is available after a long delay (user return)
  - Solution: design a reward model (ML model) that predicts true reward from available signals