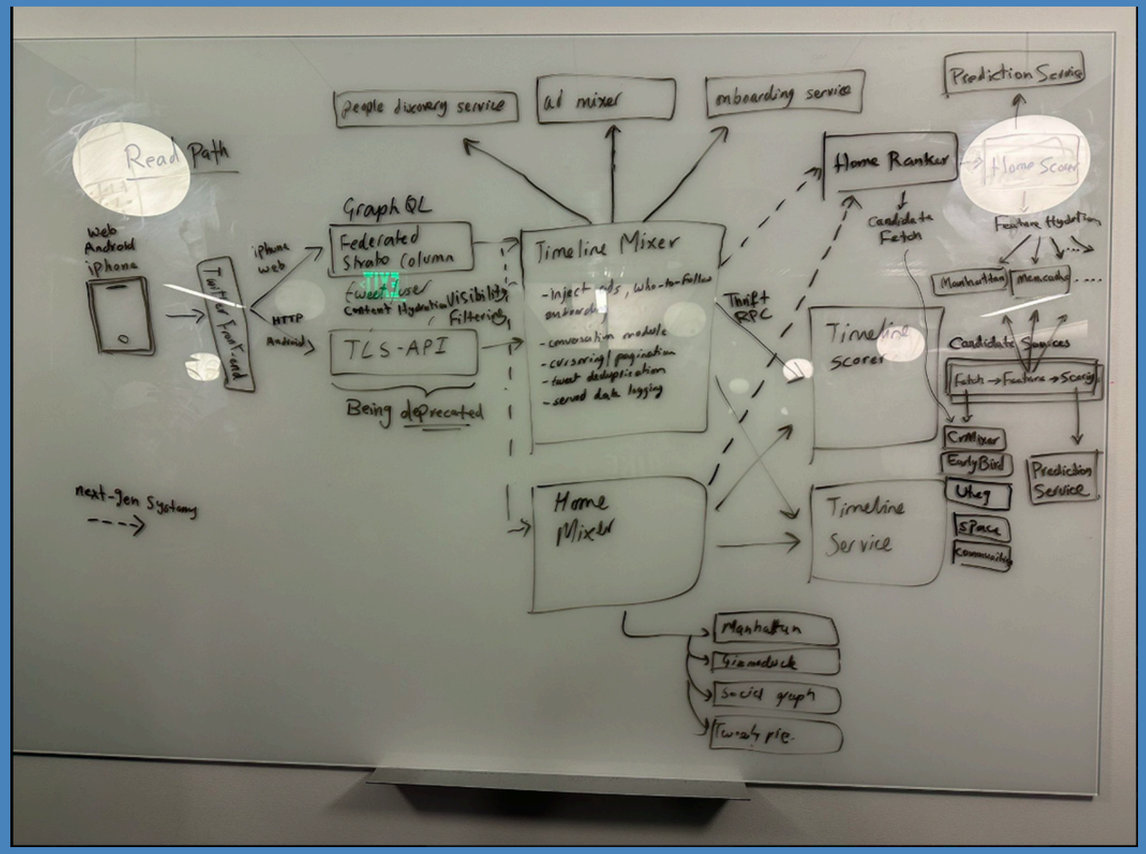
Twitter Recommendation Algorithm: A Computational Perspective

AKSHAY
HIMANSHU
RAGHAVENDRA

The most dramatic moment...



Overview & Agenda

What we'll cover

- System Architecture Overview
- Algorithmic Components & Complexity Analysis
- Graph Algorithms (SALSA, Random Walks)
- Greedy & Heuristic Approaches
- Machine Learning Integration
- Computational Complexity & Scalability
- toy_twitter : notebook

Problem Statement

Challenge: From billions of tweets, select ~1500 most relevant for each user

Constraints:

- Real-time response (< 1.5 seconds)
- Billions of users, millions of tweets per day
- Personalization at scale
- Key Question: How do we efficiently rank and filter?

Twitter (X) Recommendation Pipeline

1. CANDIDATE SOURCING

In-Network Sources
(Accounts You Follow)

Out-of-Network Sources
(Topics, Trends, Similar Users)

RealGraph
(User Interaction Probability)

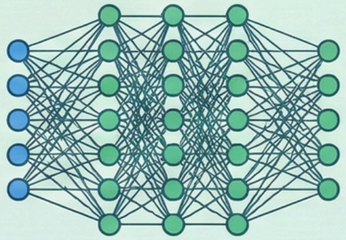
SimClusters
(Community-based Interests)

Candidate Tweet Pool (~1500 Tweets)

2. RANKING

Light Ranker
(Initial Scoring)

Heavy Ranker
(Deep Neural Network, ~48M parameters)



Predictive Models: Probability of Like, Retweet, Reply, Click, Negative Feedback

Scoring based on **Engagement Signals**

Scored Candidates

3. HEURISTICS & FILTERING

Social Proof
(Network Validation)

Author Diversity
(Preventing Dominance)

Content Balance
(In-Network vs. Out-of-Network)

Visibility Filtering
(Safety, Blocked Users, NSFW Content)

Feedback-based Fatigue
(Downranking)

Filtered & Balanced Tweets

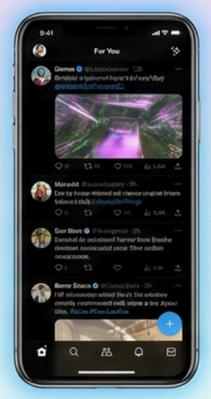
4. MIXING & SERVING

Final Composition

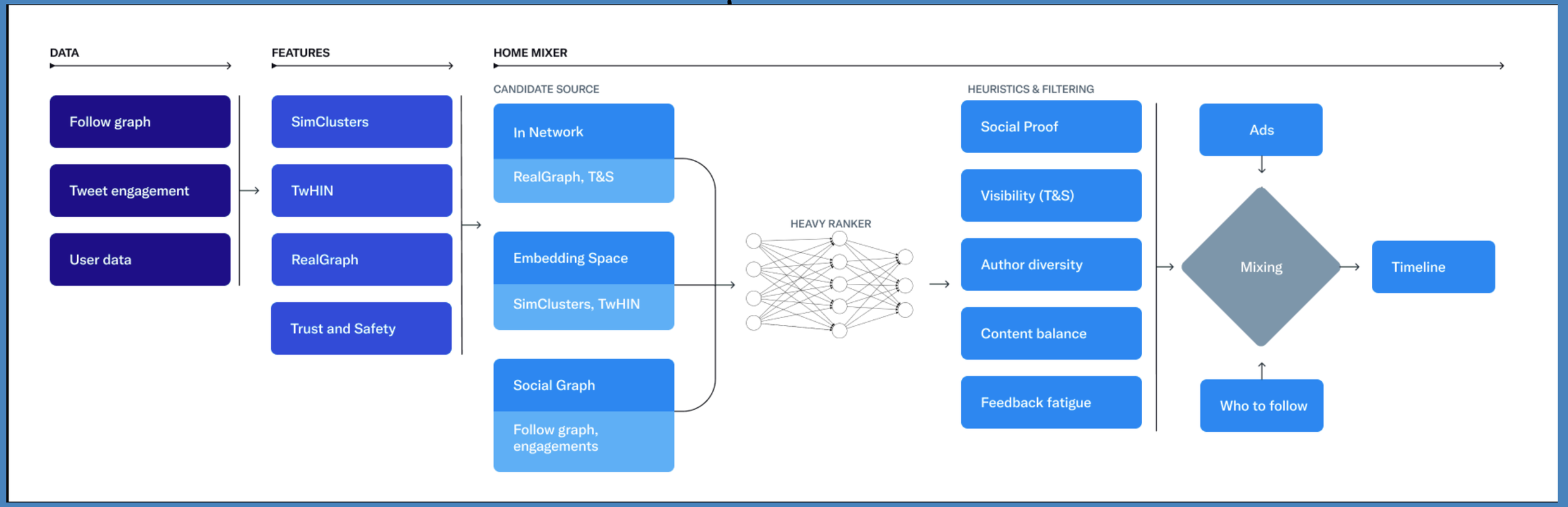
Injecting Ads & Promoted Content

Integrating "Who to Follow" Recommendations

For You Timeline Construction

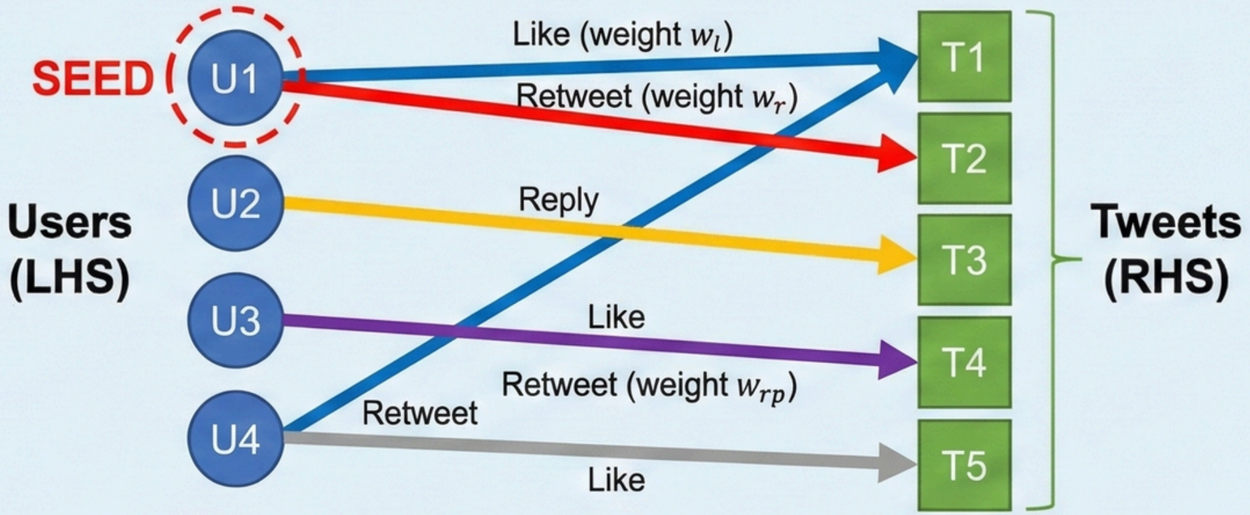


Components used to construct a timeline:

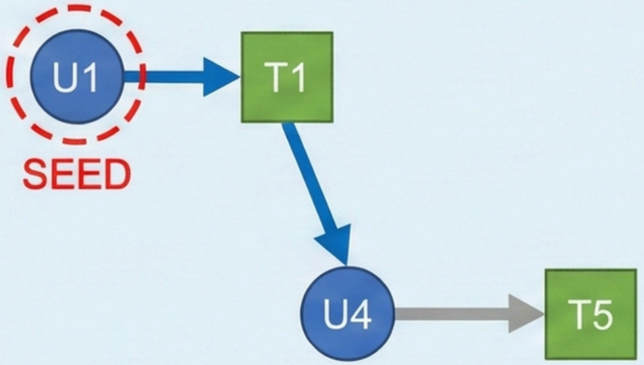


Algorithm	Category	Why	Course Topic	Exact Reason
SALSA	Randomized	Uses random choices for neighbor selection	Randomized Algorithms	Probabilistic exploration, $O(k \times d)$ vs $O(V^2)$
Top-K Heap	Greedy	Always picks highest score first	Greedy Algorithms	Greedy choice property + optimal substructure
Approximate Top-K	Approximate Greedy	Samples subset, estimates threshold	Approximate Greedy	Trade accuracy for speed, probabilistic guarantee
QuickSelect	Randomized	Random pivot selection	Order Statistics	Average $O(n)$, avoids worst-case inputs
ML Training	NP-Hard	Non-convex optimization, subset selection	P vs NP	No polynomial-time solution for global optimum
ML Inference	P (Polynomial)	Fixed architecture, matrix ops	P vs NP	$O(L \times M^2)$ forward pass, deterministic
Feature Selection	NP-Complete	Subset selection problem	NP-Complete	Reduces to subset sum (known NP-complete)
Heuristics	Heuristic	Greedy/evolutionary approximations	Heuristic Algorithms	Practical when exact is intractable
Caching	Amortized	First $O(n)$, then $O(1)$	Amortized Analysis	Average cost over sequence of operations
Cluster Partition	Divide-Conquer	Recursive splitting + merge	Master Theorem	$T(n)=2T(n/2)+O(n) \rightarrow O(n \log n)$

User-Tweet Graph Representation (Bipartite Graph)



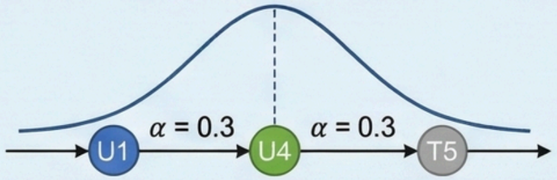
SALSA Algorithm (Stochastic Approach for Link-Structure Analysis)



SALSA Algorithm:

1. Start from seed user (U1).
2. Random walk alternates: User → Tweet → User → Tweet.
3. With probability α, reset to seed.
4. Score tweets by visit frequency (stationary distribution).
5. Convergence to stationary distribution reflects relevance/importance.

Complexity: O(num_walks × max_length × avg_degree)

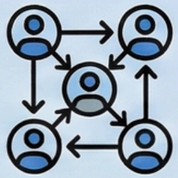


Pseudocode SALSA random walk algorithm

```
def salsa_random_walk(graph, seeds, num_walks, max_length):
    scores = {}
    for seed in seeds:
        for _ in range(num_walks):
            current = seed
            for step in range(max_length):
                if random() < reset_probability:
                    current = seed  # Reset to seed
                else:
                    current = random_neighbor(graph,
current)
            scores[current] = scores.get(current, 0) + 1
    return normalize(scores)
```

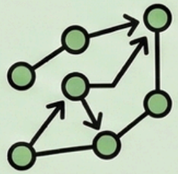
Asymptotic Analysis of Candidate Generation

Multiple Candidate Sources:



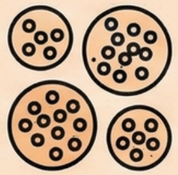
In-Network:

$O(\texttt{following\_count} \times \texttt{tweets\_per\_user})$



Social Graph:

$O(\texttt{num\_walks} \times \texttt{walk\_length})$



SimClusters:

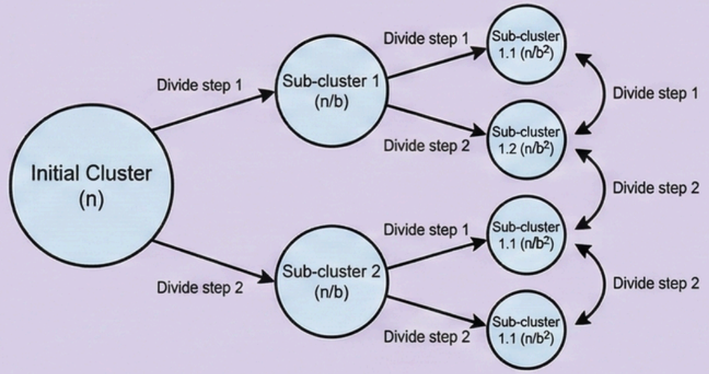
$O(\texttt{user\_clusters} \times \texttt{tweets\_per\_cluster})$

Master Theorem Application:

$$T(n) = aT(n/b) + f(n)$$

If we use **divide-and-conquer** for **cluster-based generation**

Example: Recursive cluster partitioning



Recursive cluster partitioning visualization

SimClusters

NEWS
293M

The Associated Press
@AP

The Guardian
@guardian

CNN
@cnn

The Wall Street Journal
@WSJ

Reuters
@Reuters

NBA
70M

Hélà
@Kyrielrving

James Harden
@JHarden13

Carmelo Anthony
@carmeloanthony

Chris Paul
@CP3

Anthony Davis
@AntDavis23

POP
332M

Selena Gomez
@selenagomez

jlo
@JLo

Rihanna
@rihanna

Lady Gaga
@ladygaga

KATY PERRY
@katyperry

Justin Timberlake
@jtimberlake

BOLLYWOOD
80M

arjun rampal
@rampalarjun

Bipasha Basu
@bipsluvurself

Boman Irani
@bomranirani

TheFarahKhan
@TheFarahKhan

sushmita sen

SOCCER
191M

Manchester City
@ManCity

Sergio Ramos
@SergioRams

Karim Benzema
@benzema

Gareth Bale
@GarethBale

Pseudocode Cluster Candidate generation

```
def generate_candidates_dc(clusters, threshold):
    if len(clusters) <= threshold:
        return brute_force_candidates(clusters)
    mid = len(clusters) // 2
    left = generate_candidates_dc(clusters[:mid],
threshold)
    right = generate_candidates_dc(clusters[mid:],
threshold)
    return merge_and_deduplicate(left, right)
```

Approximate Greedy Algorithms

Challenge: Exact top-K too expensive for real-time

Approximate Solutions:
Sampling-based approximation
Threshold-based filtering
Locality-sensitive hashing for similarity

Approximate Greedy Algorithms

```
def approximate_top_k(candidates, k, sample_size):
    # Sample and estimate threshold
    sample = random.sample(candidates, min(sample_size,
len(candidates)))
    threshold = sorted(sample, key=lambda x: x.score)[-
k].score

    # Filter and return top-k from filtered set
    filtered = [c for c in candidates if c.score >=
threshold]
    return sorted(filtered, key=lambda x: x.score,
reverse=True)[:k]
```

Neural Ranking Model:

Input: Feature vector (user features, tweet features, interaction features**)**
Architecture: Deep neural network
Output: Relevance score
Complexity:
Forward pass: $O(L \times M^2)$ where L = layers, M = hidden units
Training: NP-hard optimization (non-convex)
Inference: Polynomial time
Connection:
P vs NP: Training is hard, inference is in P
Heuristic algorithms: Gradient descent as heuristic for optimization

Order Statistics & Selection

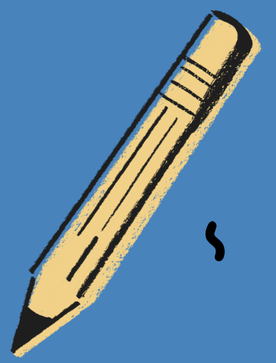
Problem: Finding K-th largest score efficiently

QuickSelect Algorithm:
Randomized selection algorithm
Average case: O(n), Worst case: $O(n^2)$

Order Statistics & Selection

```python
def quickselect(arr, k):
    if len(arr) == 1:
        return arr[0]
    pivot = random.choice(arr)
    left = [x for x in arr if x > pivot]
    right = [x for x in arr if x <= pivot]

    if k <= len(left):
        return quickselect(left, k)
    else:
        return quickselect(right, k - len(left))
```



Scalability & Amortized Analysis

Amortized Cost of Operations:

- Feature extraction: O(1) amortized per candidate
- Batch processing: Amortize ML inference cost
- Caching: Amortize repeated computations

Example:

- First request: O(n) to build cache
- Subsequent requests: O(1) lookup
- Amortized: O(1) per request

NP-Hard Aspects & Heuristics

- NP-Hard Problems in Recommendation:
  a. Optimal feature selection (subset selection)
  b. Optimal model architecture search
  c. Multi-objective optimization (relevance + diversity)
- Heuristic Solutions:
  - Greedy feature selection
  - Evolutionary algorithms for architecture search
  - Weighted sum for multi-objective

Connection: NP-Complete problems, heuristic algorithms, computability

Choosing the absolute "best" 1,500 tweets out of hundreds of millions is technically an NP-Hard problem.

The Impossible Task: To get the mathematically "perfect" ranking, the computer would have to compare every single tweet against every other tweet for every single user combination. This would take too long.

The Solution (Heuristics): Since we can't solve NP-Hard problems perfectly in real-time, engineers use "heuristics" or "greedy algorithms" (like the Greedy approach in previous slide). These don't guarantee the perfect solution, but they give a good enough solution very quickly.

Key Takeaways

Algorithmic Techniques Used:

- Graph algorithms (random walks, SALSA)
- Greedy algorithms (top-K selection)
- Randomized algorithms (sampling, quickselect)
- Heuristic algorithms (ML training, feature selection)
- Amortized analysis (caching, batch processing)

Complexity Trade-offs:

- Exact vs. approximate solutions
- Time vs. space complexity
- Accuracy vs. latency

References

- https://github.com/twitter/the-algorithm
- https://blog.x.com/engineering/en_us/topics/open-source/2023/twitter-recommendation-algorithm