Learning on Streaming Graphs

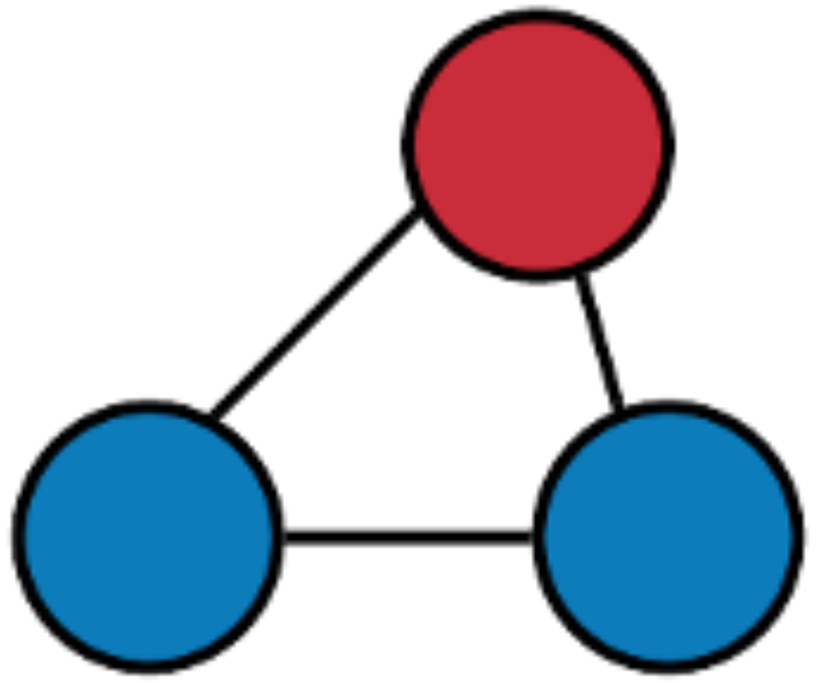
M. Perini, G. Ramponi, P. Carbone, V. Kalavri

massimo.perini@ed.ac.uk

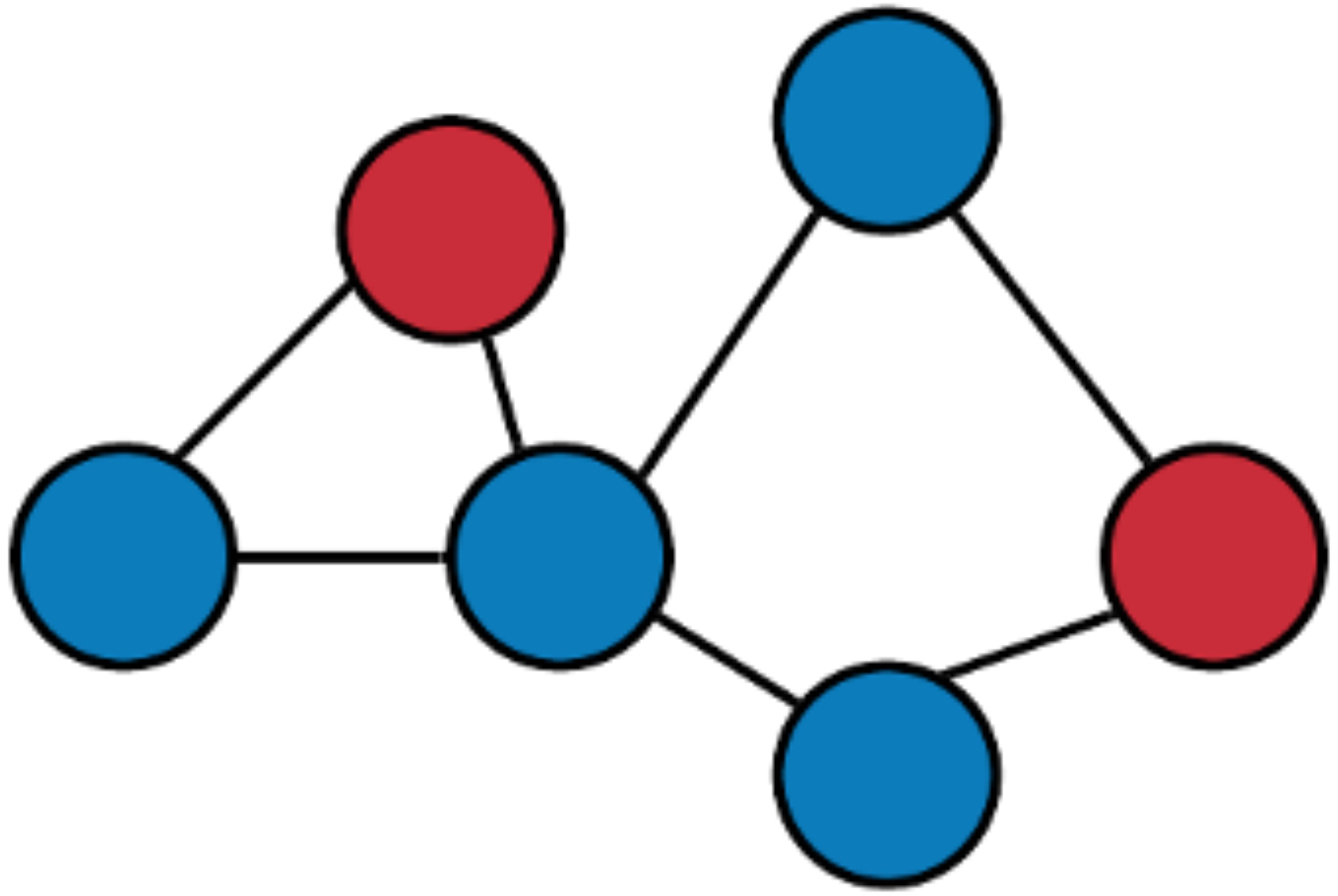
Example

- Anti-money laundering prevents criminals from moving illicit funds through the financial system.
- Machine Learning to detect suspicious transactions.
- Challenges:
  - Massive graph
  - New interactions
  - Timely access to updated predictions.

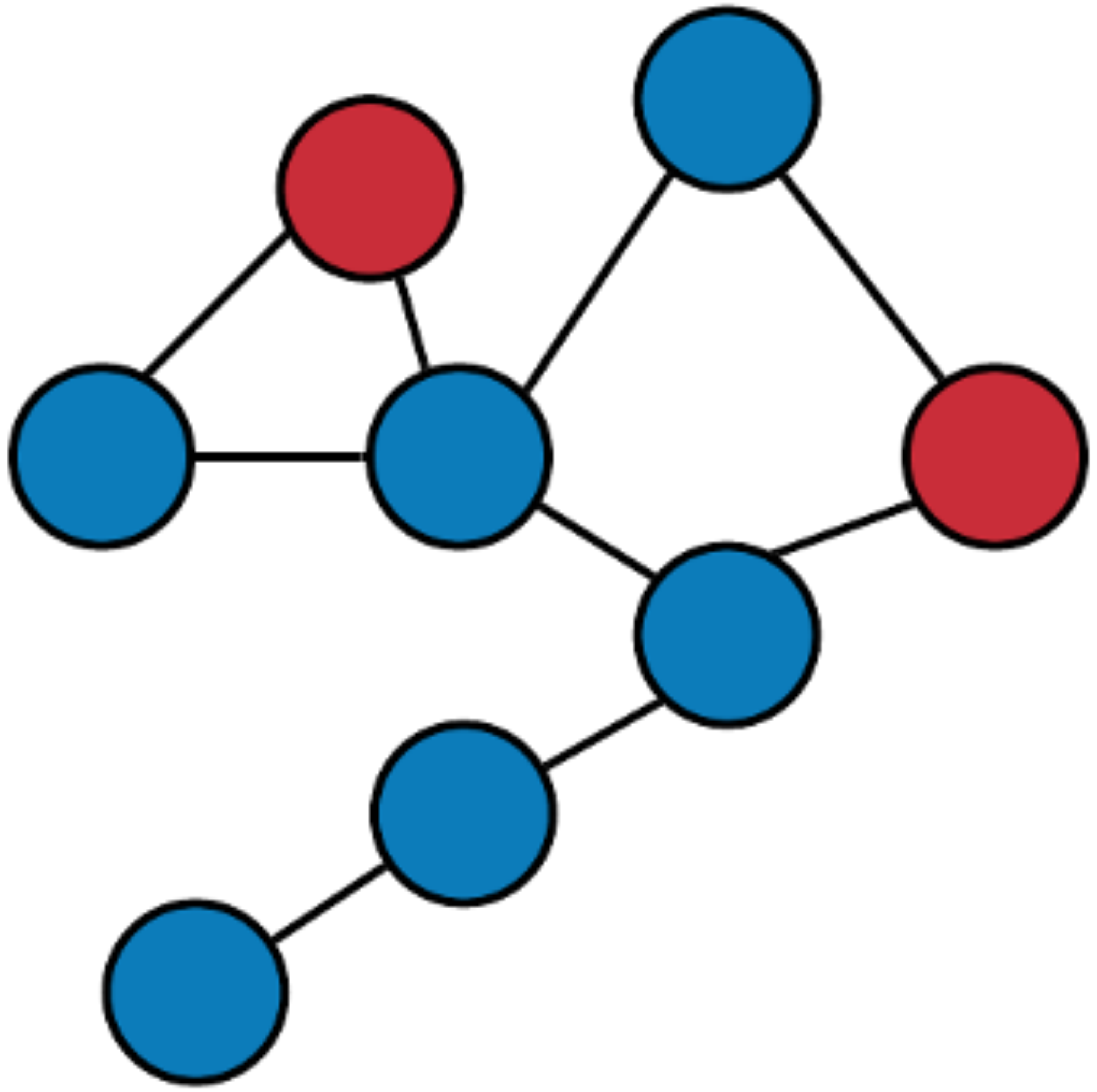
Example



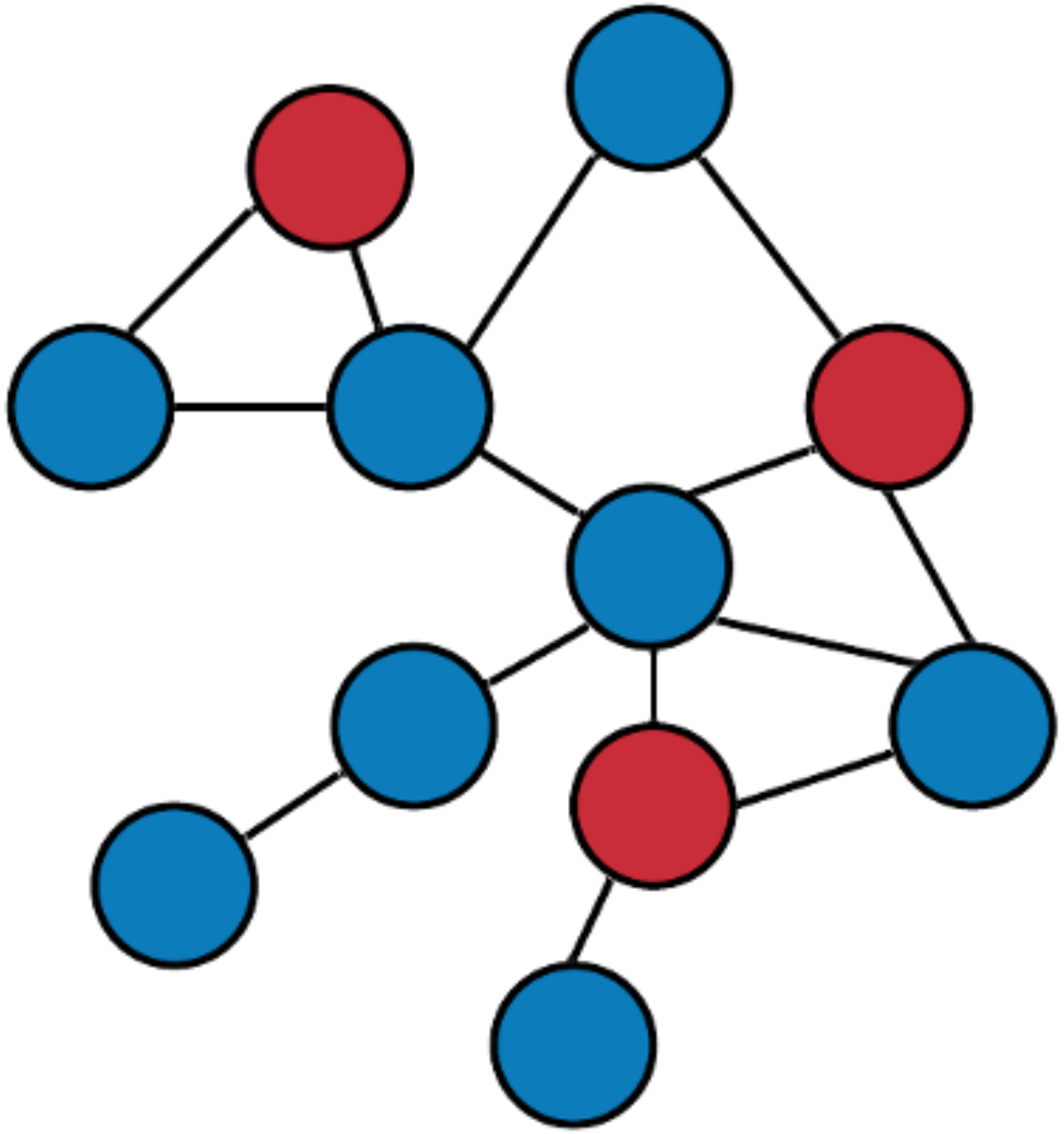
Example



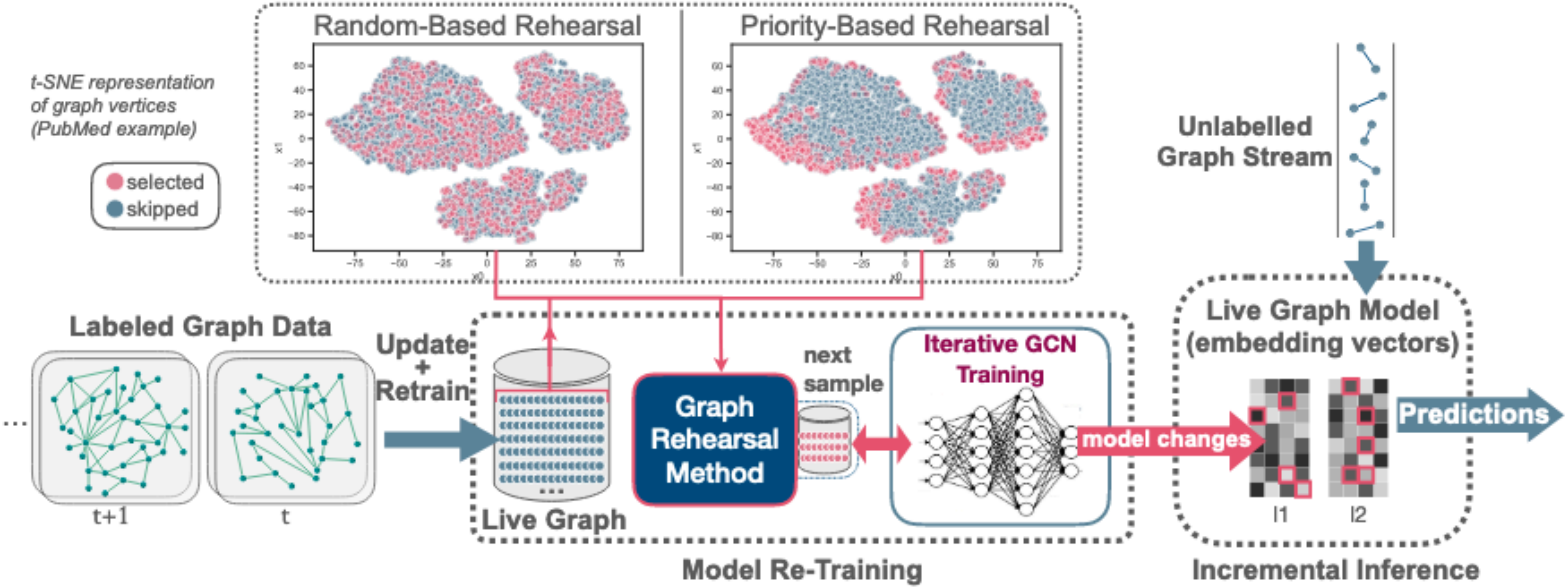
Example



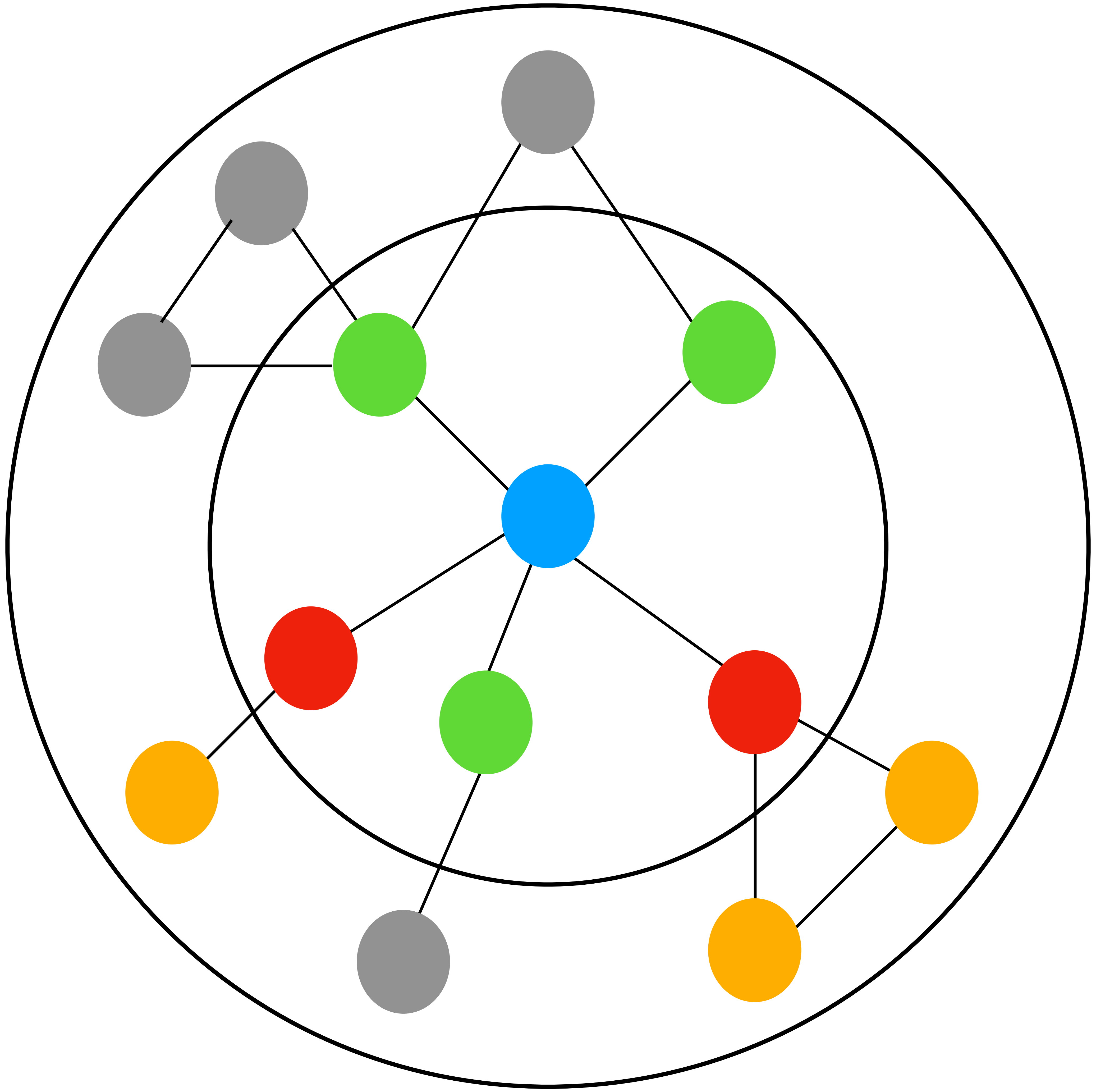
Example



Overall system



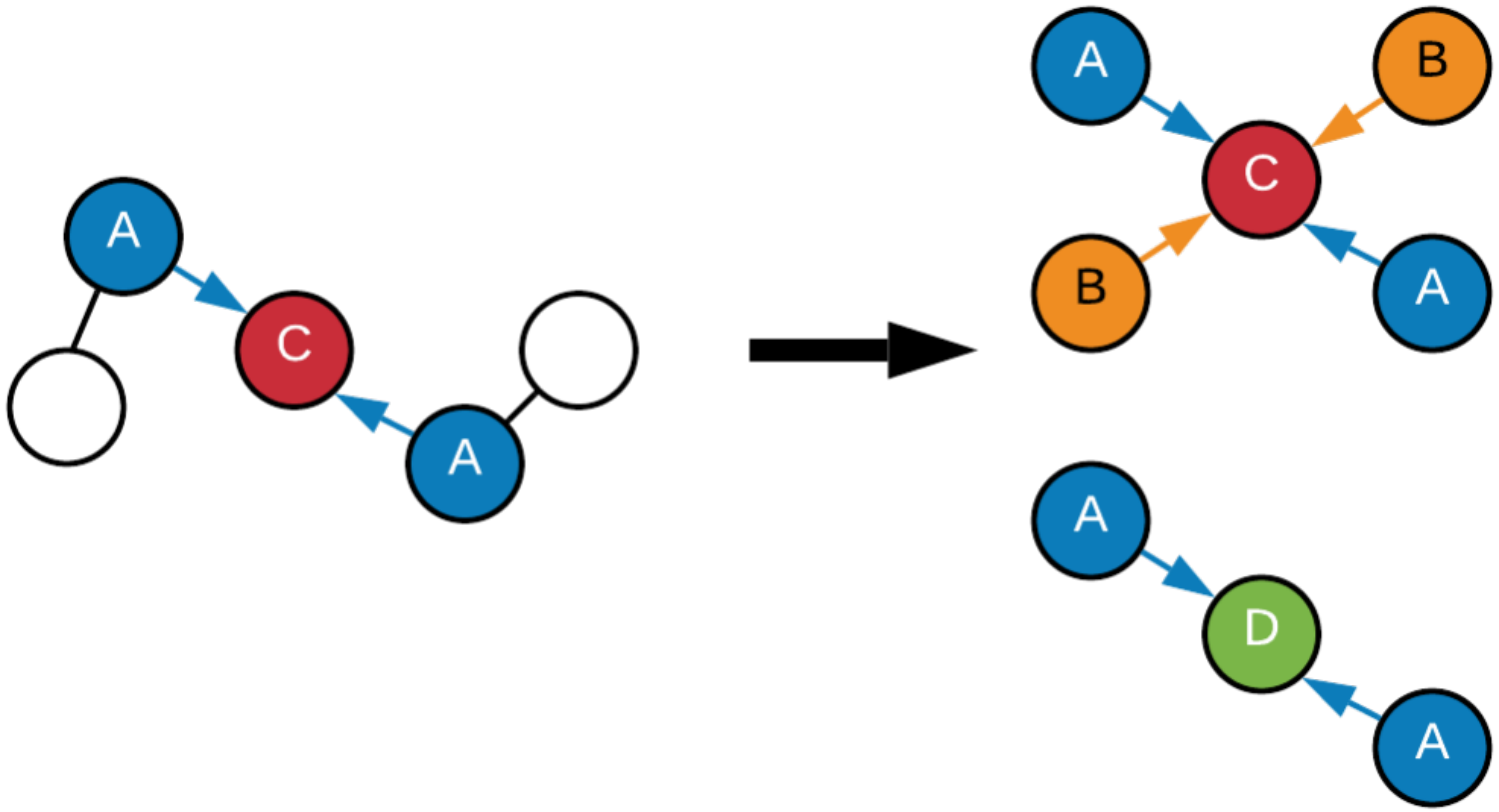
Graph Neural Network



- Inductive graph neural networks maps vertices to labels
- Each node is represented by the aggregation of its neighborhood.

Concept drift

- The function learned might evolve with the evolution of the graph.



No rehearsal

- New data interferes with the knowledge gained previously
- Training methods assume samples are i.i.d.
- An online training algorithm (SGD) might converge towards a bad solution

Continual learning

- Catastrophic forgetting: new data interferes with the knowledge gained previously.
- Previous online learning approaches involve specific neural network architecture or regularization
- They perform worse than a baseline that trains on few random samples.

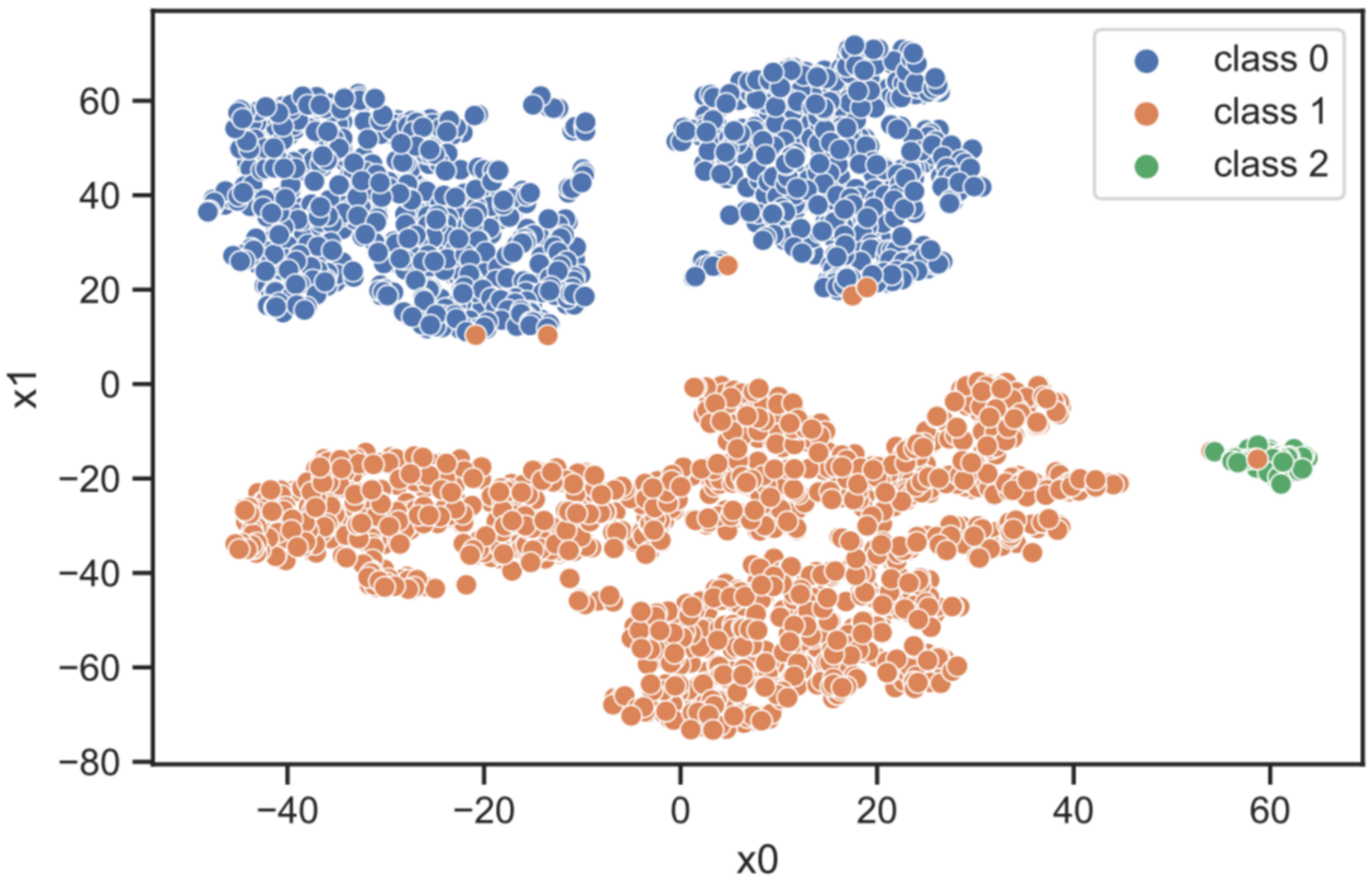
RBR: Random-based Rehearsal

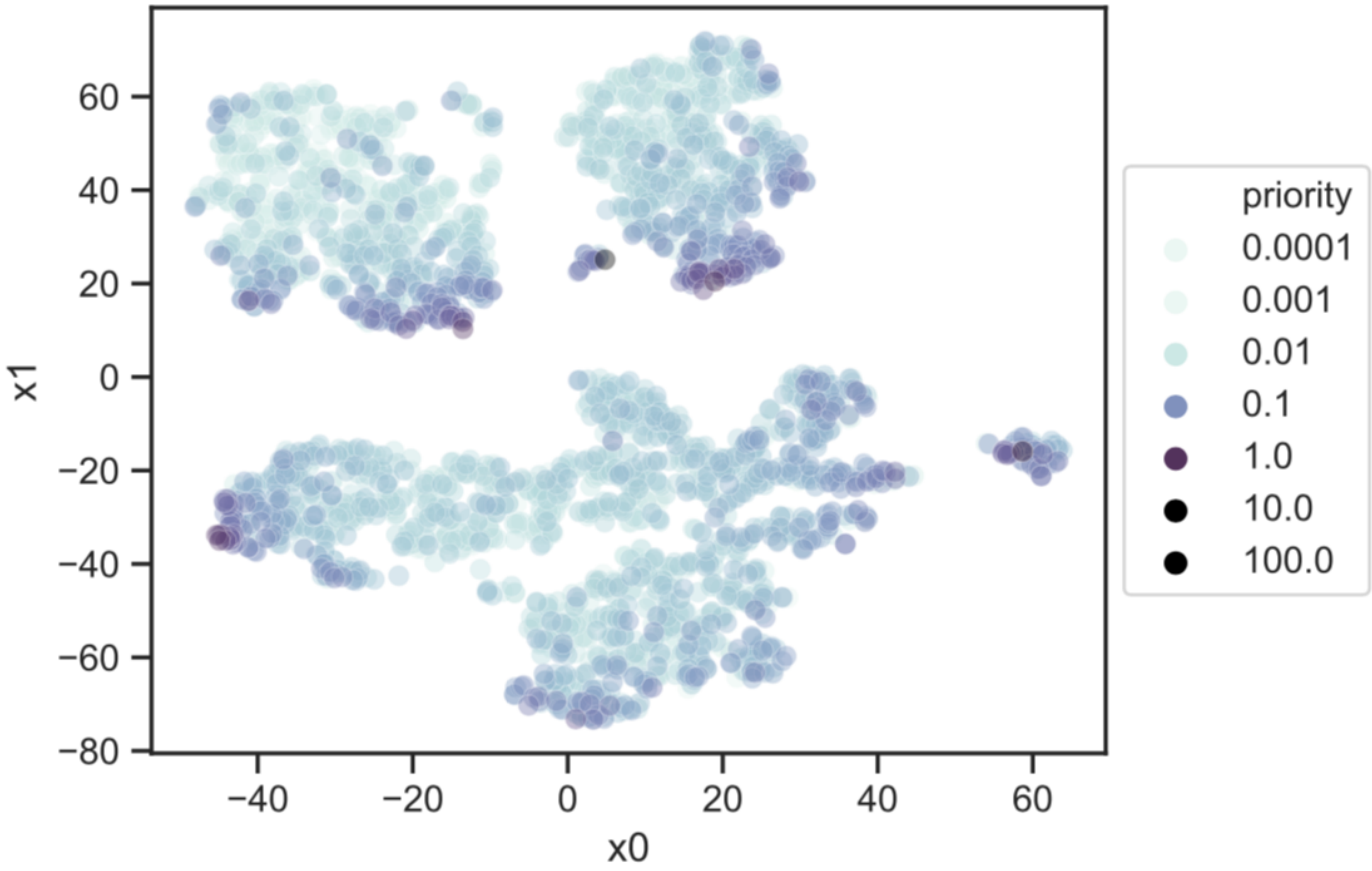
- Reuse past data to update the model in the new snapshot.
- Works if sampling at random selects a representative set of the vertices
- Trains more old samples

PBR: Priority-based Rehearsal

- Idea: learn more from some samples (“support vectors”) than from others
- We can measure how unexpected is a vertex
    - How much the model can’t predict it
- “Important” ones are drawn more frequently
- New vertices get maximum priority
- Updates error during gradient updates (log(|V|) with segment tree)

PBR: Priority-based Rehearsal





Experiments

Evaluation

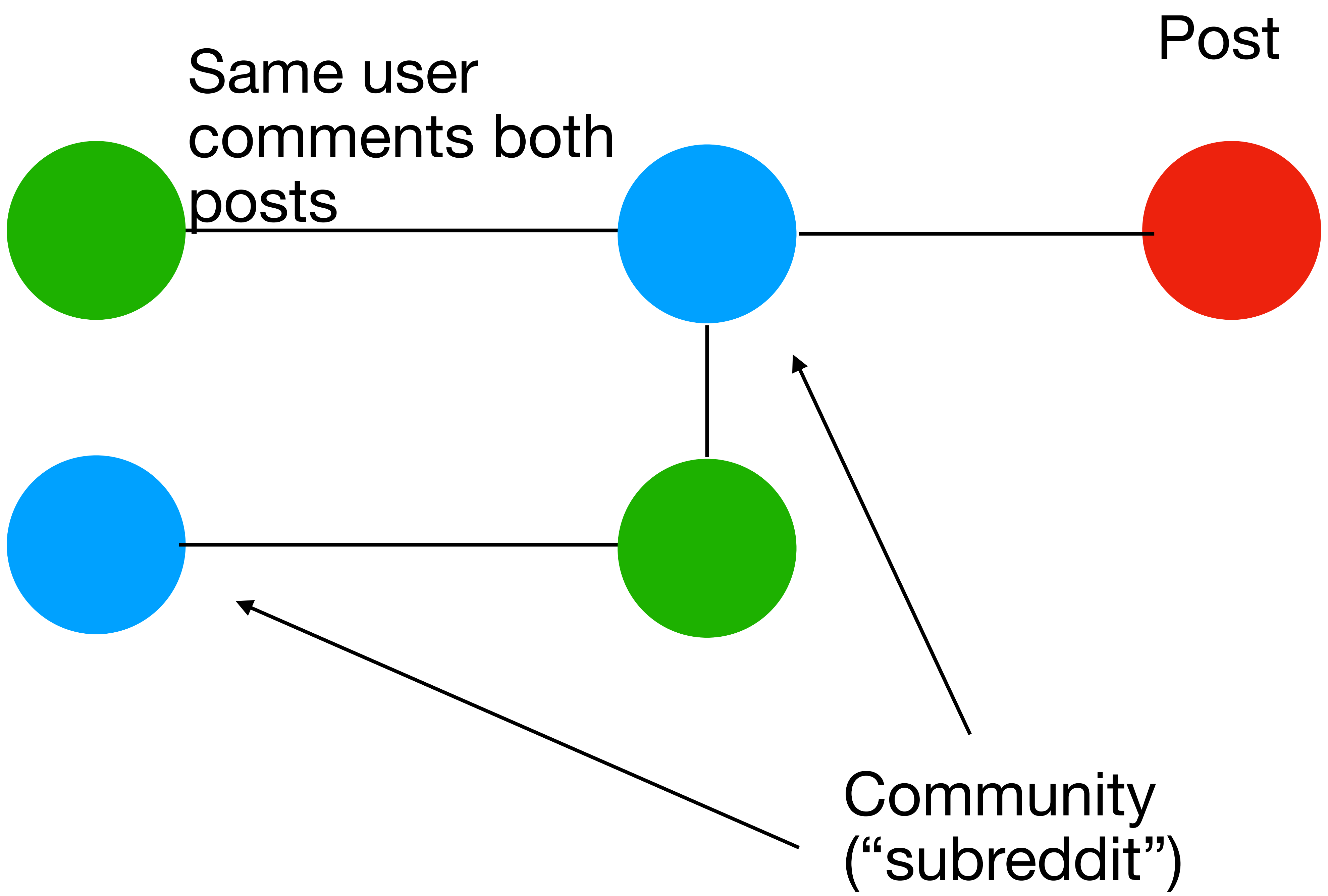
- Sequence of temporal snapshots
- Default strategy: new vertex added to test or train set at random.
- Temporal strategy: test set using vertices of next snapshots

Baselines

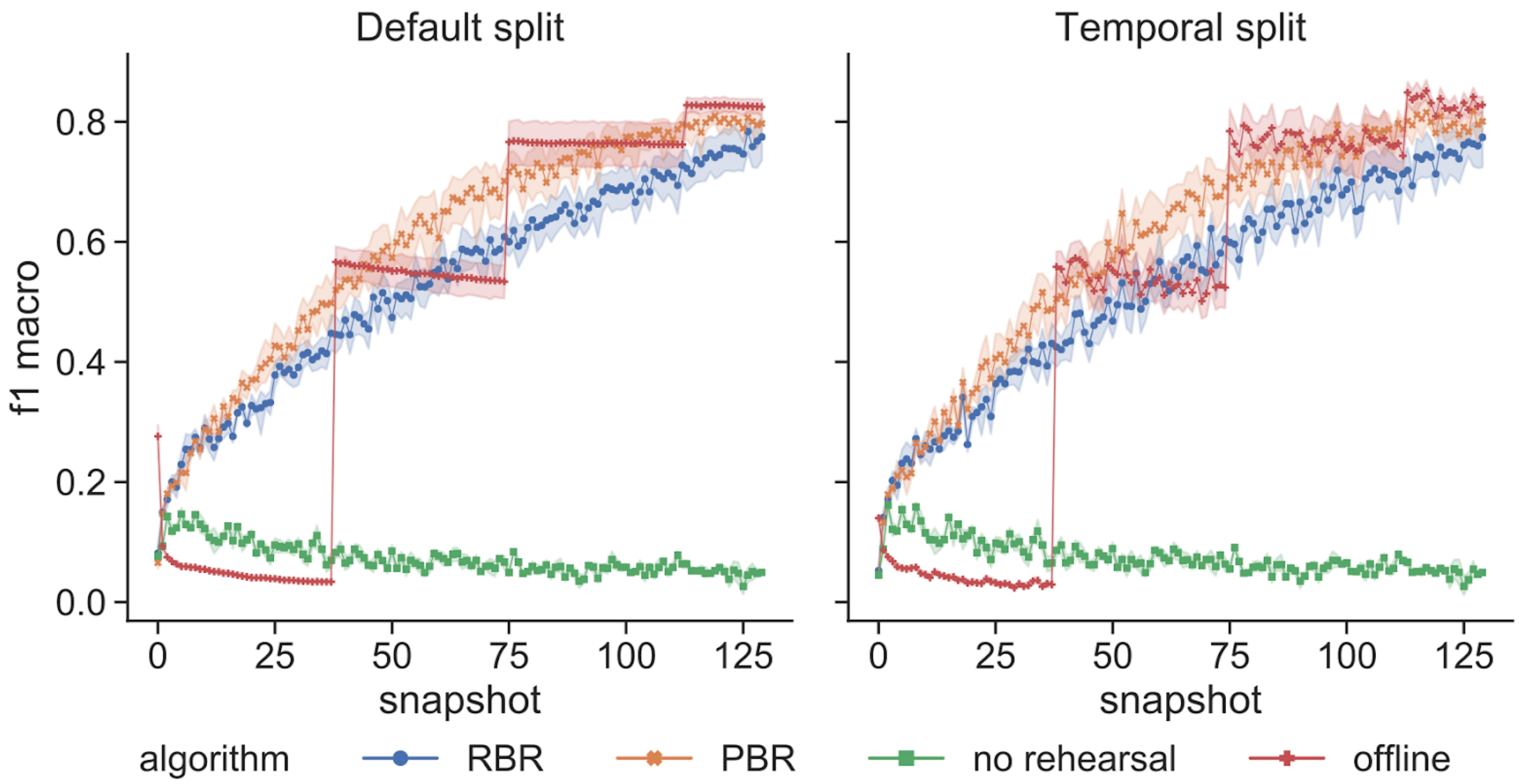
- Offline: Train over the full graph with multiple epochs
- No-Rehearsal: Train over the new vertices.
- ContinualGNN: replay-based method

Reddit

- Dynamic edge addition, social network



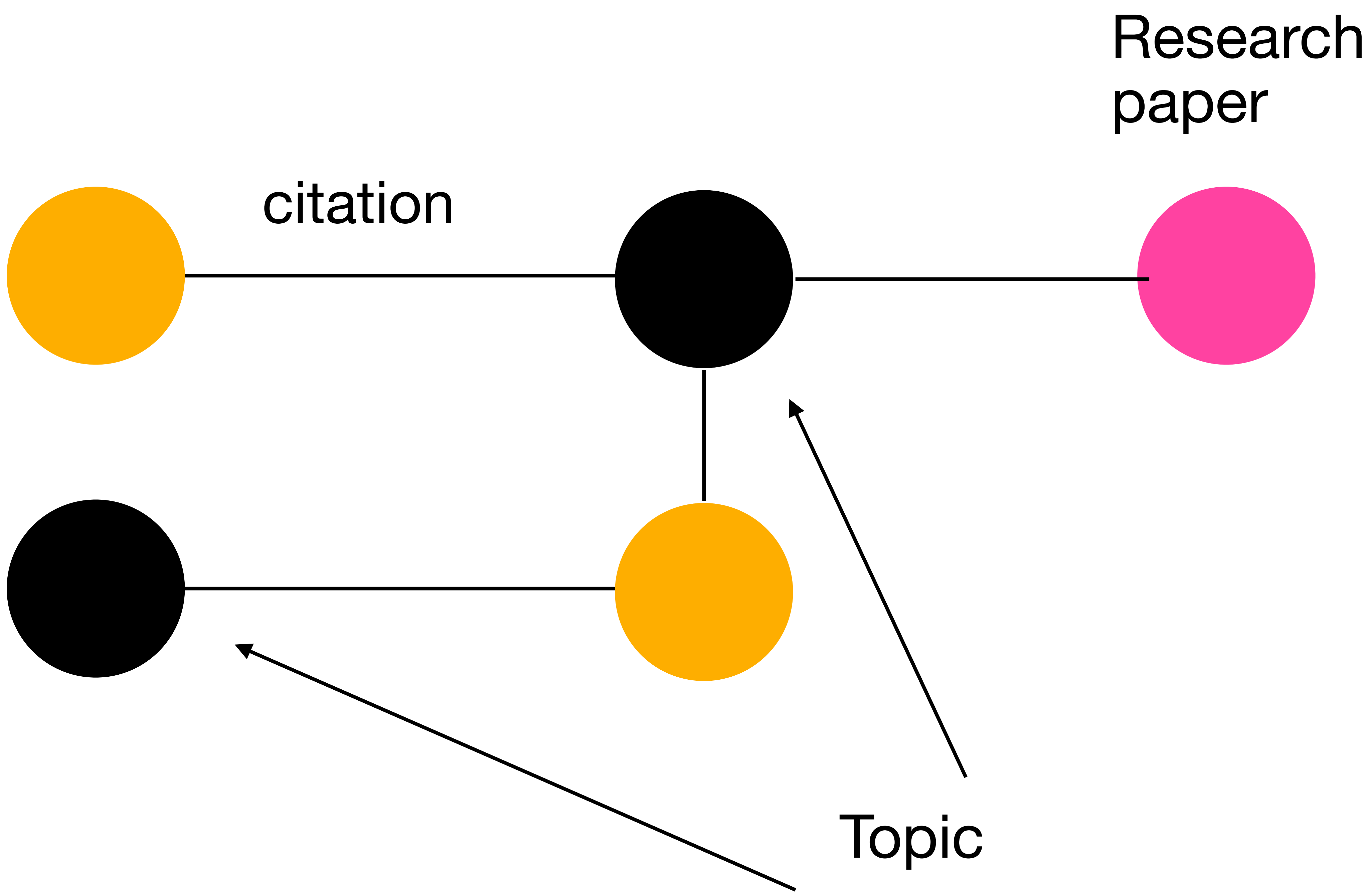
Reddit



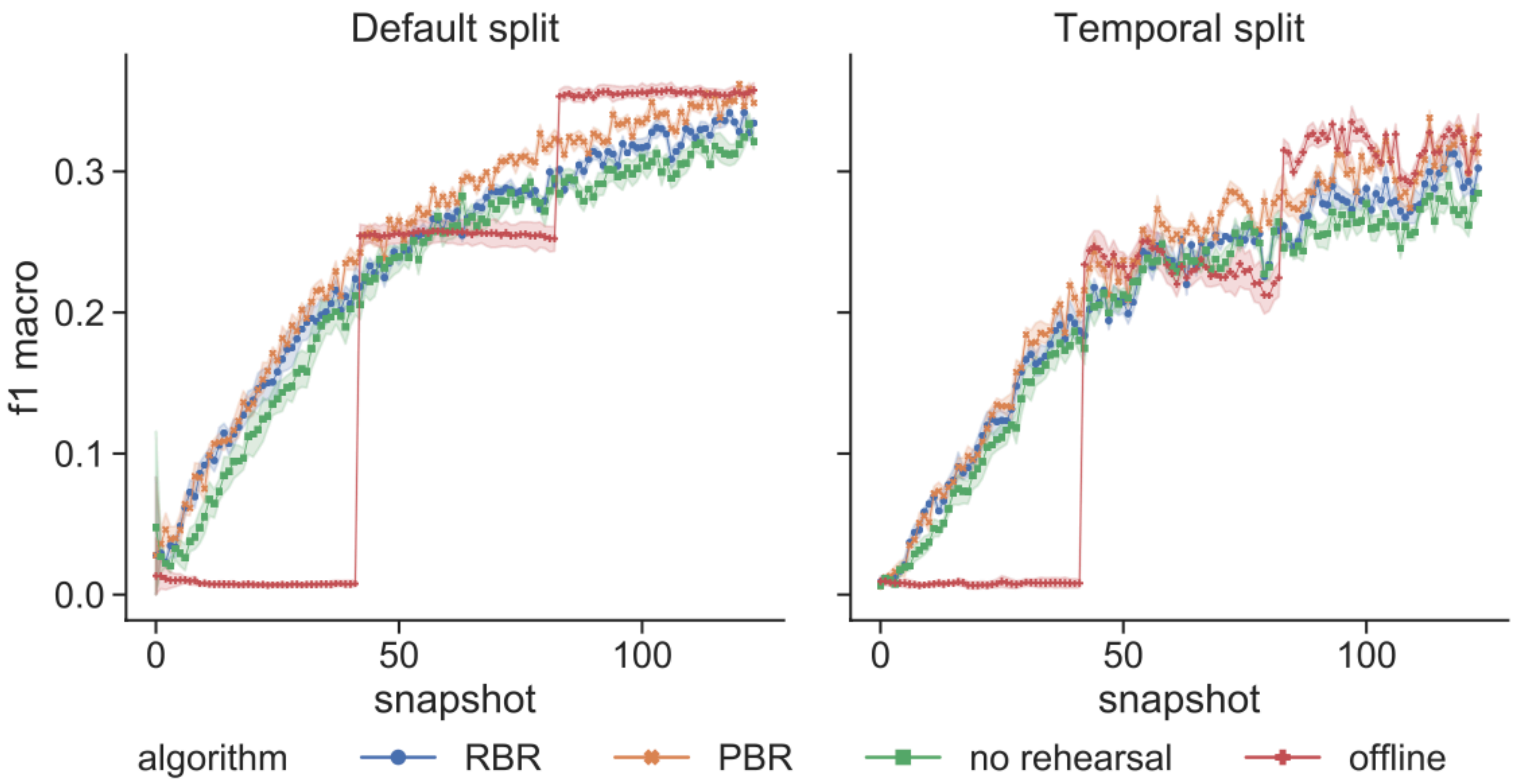
Default split
Temporal split
f1 macro
0.0
0.2
0.4
0.6
0.8
0
25
50
75
100
125
snapshot
snapshot
algorithm
RBR
PBR
no rehearsal
offline

Arxiv

- Dynamic vertex addition, publication network



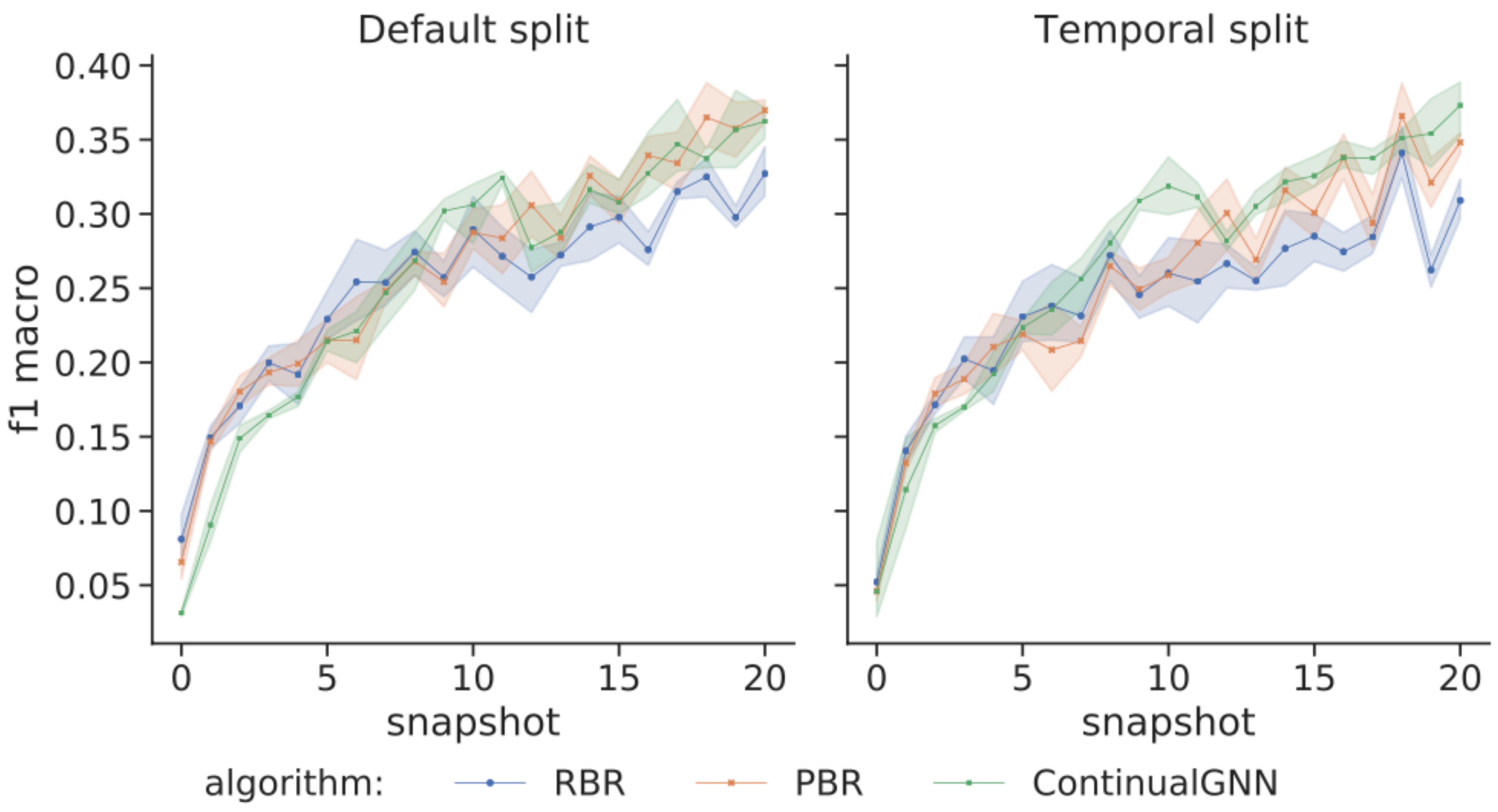
Arxiv



Default split
Temporal split
f1 macro
0.0
0.1
0.2
0.3
0
50
100
snapshot
algorithm
RBR
PBR
no rehearsal
offline

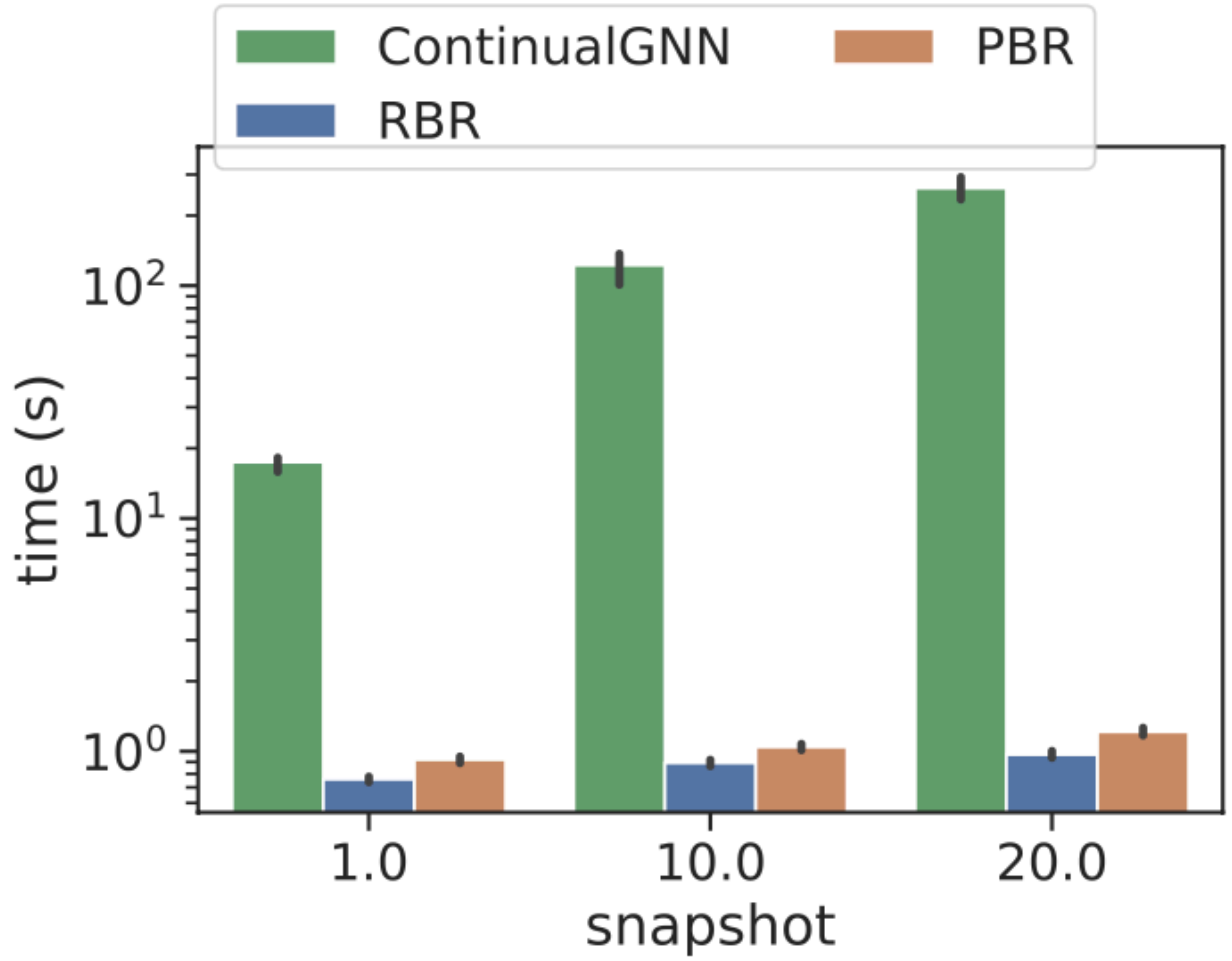
ContinualGNN

Reddit - F1



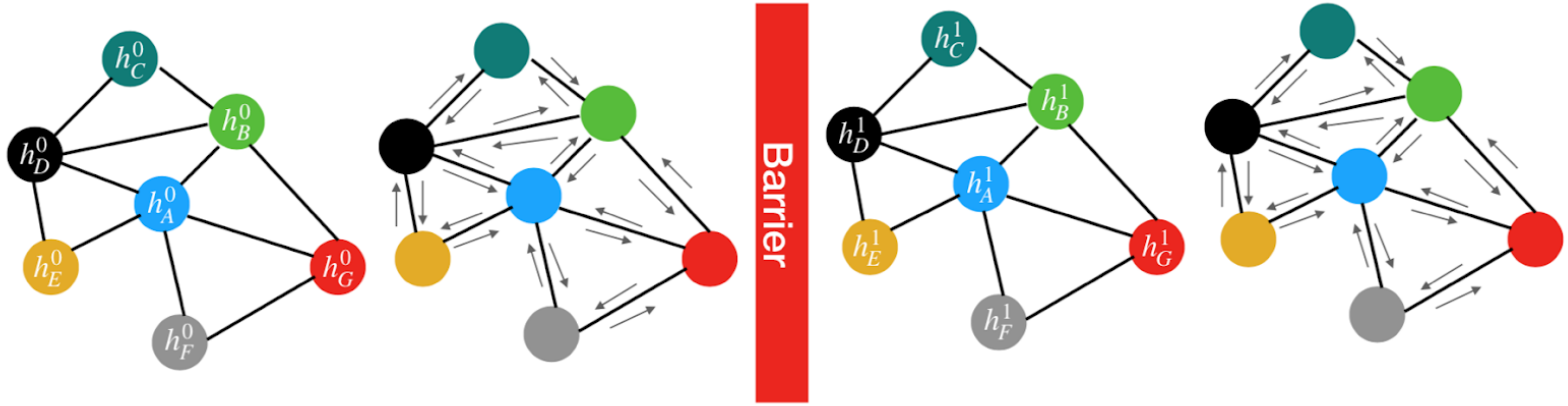
ContinualGNN

Reddit - Training time



Inference

Scalable batch systems



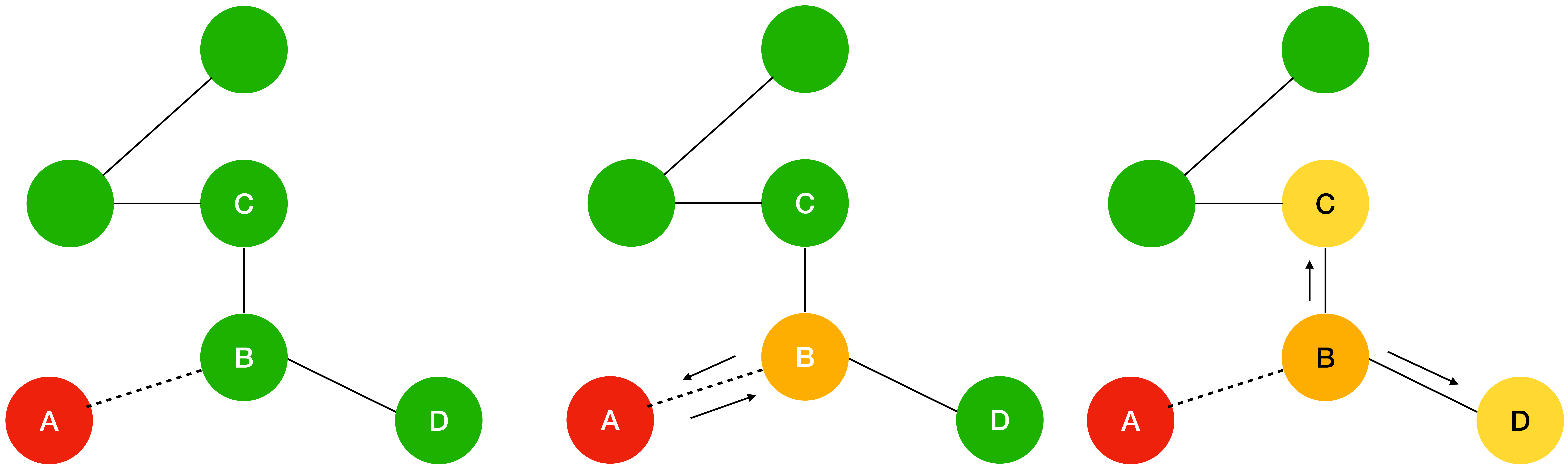
compute $h^0_{*,proj}$ **communicate** $h^0_{*,proj}$ **compute** h^1_*, $h^1_{*,proj}$ **communicate** $h^1_{*,proj}$

Problems

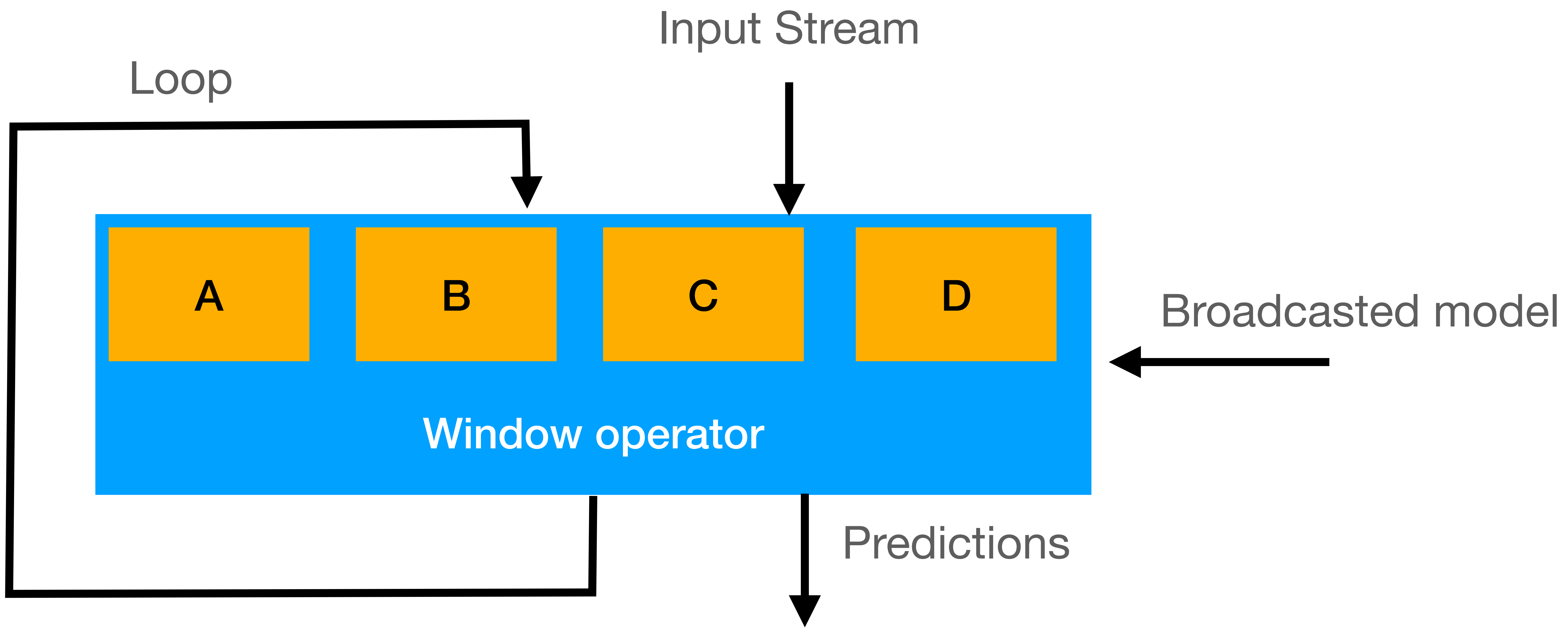
- Examples of Load-compute-store systems: Pregel, Graphx (Spark), Graphlab…
- Same execution strategy, same problems:
  - The straggler task determines the runtime
  - New mutations are not processed while computation is ongoing
  - To much re-computation for nothing

Inference

Incremental algorithm



Asynchronous Inference with Flink



Thanks