


To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy. ×

Introducing PyTorch BigGraph

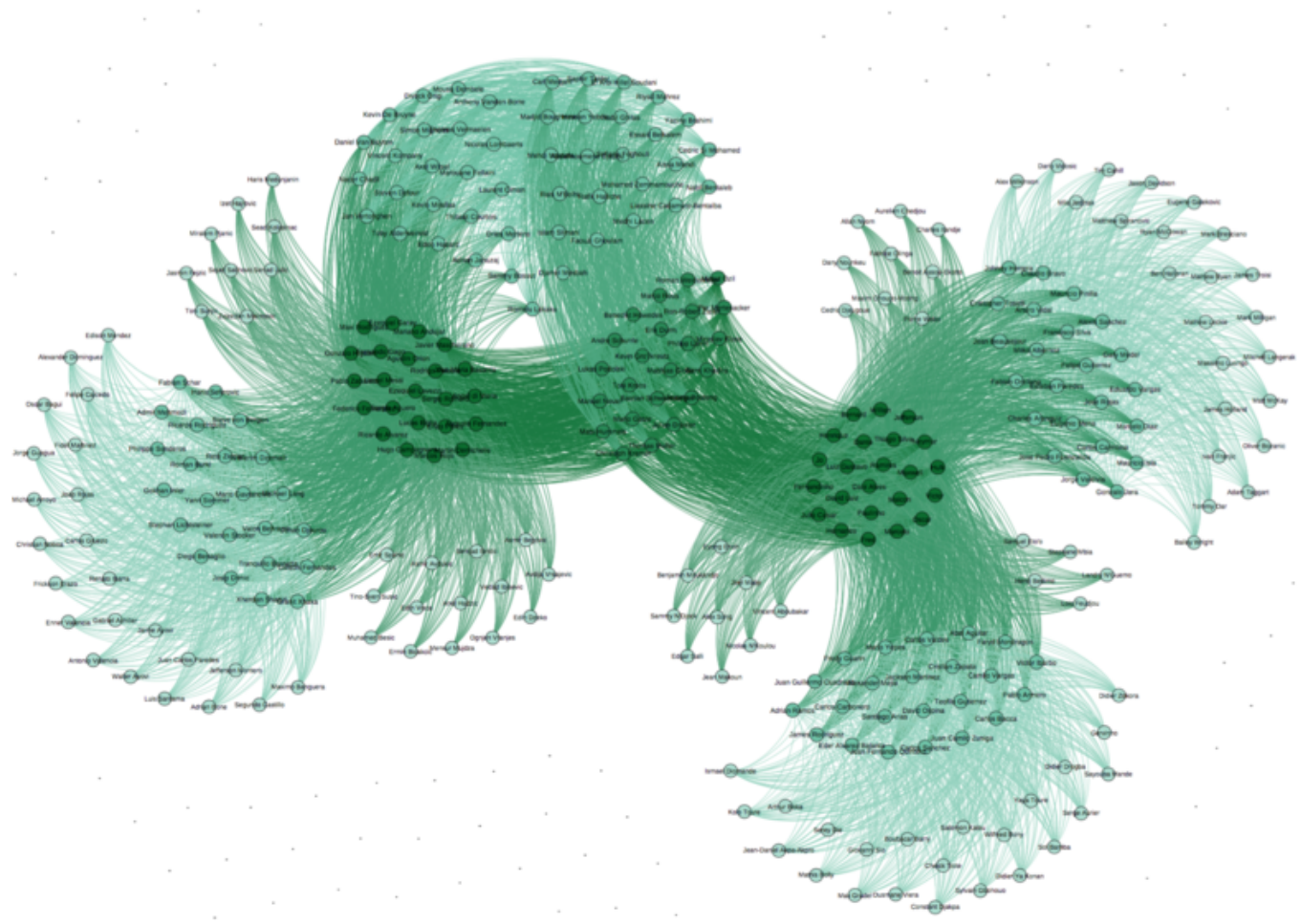
Facebook’s New Framework for Processing Large Graphs



Jesus Rodriguez

Follow

Apr 3 · 6 min read ★



Graphs are one of the fundamental data structures in machine learning applications. Specifically, graph-embedding methods are a form of unsupervised learning, in that they learn representations of nodes using the native graph structure. Training data in mainstream scenarios such as social media predictions, internet of things(IOT) pattern detection or drug-sequence modeling are naturally represented using graph structures. Any one of those scenarios can easily produce graphs with billions of interconnected nodes. While the richness and intrinsic navigation capabilities of graph structures is a great playground for machine learning models, their complexity poses massive scalability challenges. Not surprisingly, the support for large-scale graph data structures in modern deep learning frameworks is still quite limited. Recently, Facebook unveiled PyTorch BigGraph, a new framework that makes it much faster and easier to produce graph embeddings for extremely large graphs in PyTorch models.

To some extent, graph structures can be seen as an alternative to labeled training dataset as the connections between the nodes can be used to infer specific relationships. This is the approach followed by unsupervised graph embedding methods which learn a vector representation of each node in a graph by optimizing the objective that the embeddings for pairs of nodes

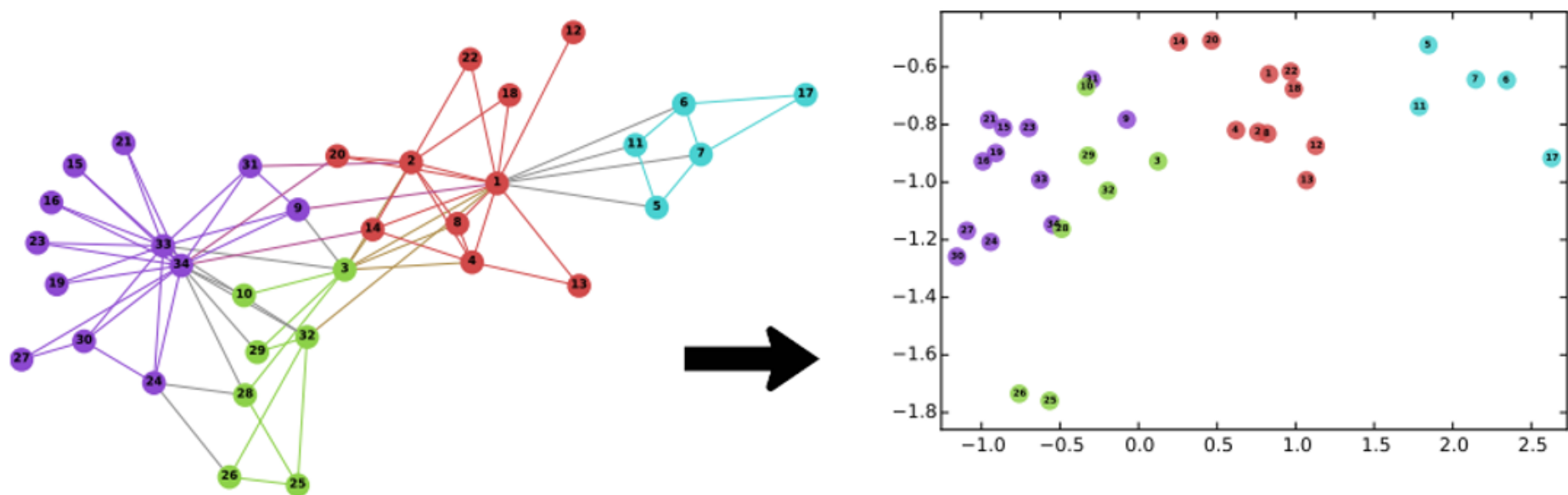
Wi
sh

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.

×

1

trained on text.



Most graph embedding methods result quite constrained when applied to large graph structures. To give a example, a model with two billion nodes and 100 embedding parameters per node (expressed as floats) would require 800GB of memory just to store its parameters, thus many standard methods exceed the memory capacity of typical commodity servers. To represents a major challenge for deep learning models and is the genesis of Facebook’s BigGraph framework.

PyTorch BigGraph

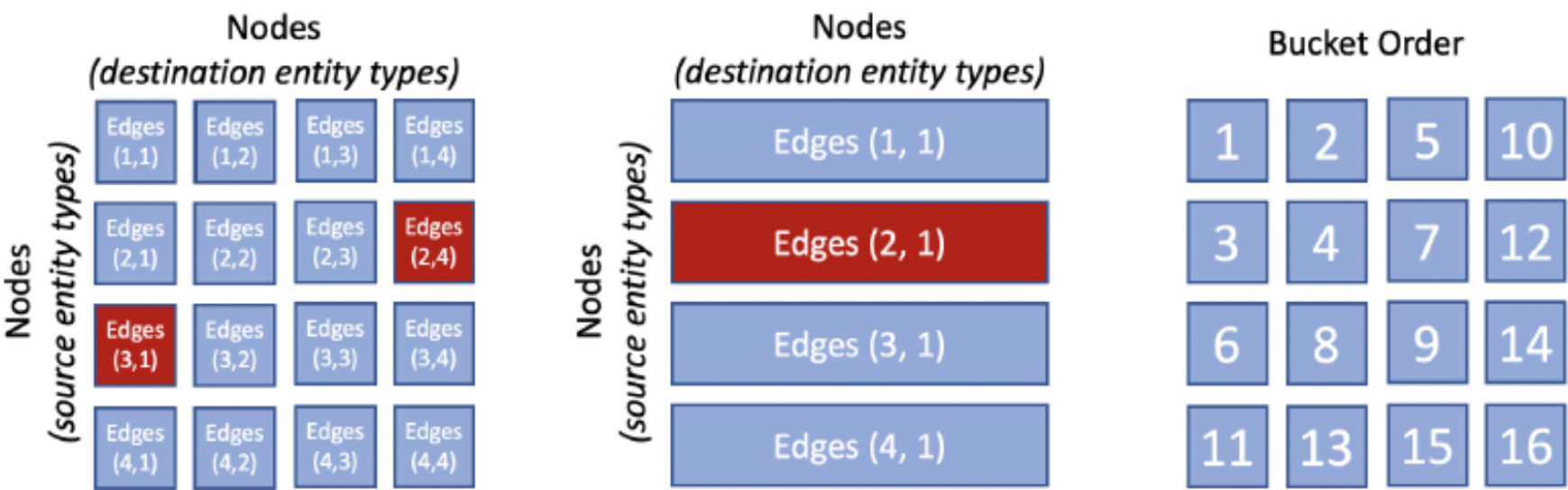
The goal of PyTorch BigGraph(PBG) is to enable graph embedding models to scale to graphs with billions of nodes and trillions of edges. PBG achieves that by enabling four fundamental building blocks:

- *graph partitioning*, so that the model does not have to be fully loaded into memory
- *multi-threaded computation* on each machine
- *distributed execution* across multiple machines (optional), all simultaneously operating on disjoint parts of the graph
- *batched negative sampling*, allowing for processing > 1 million edges/sec/machine with 100 negatives per edge

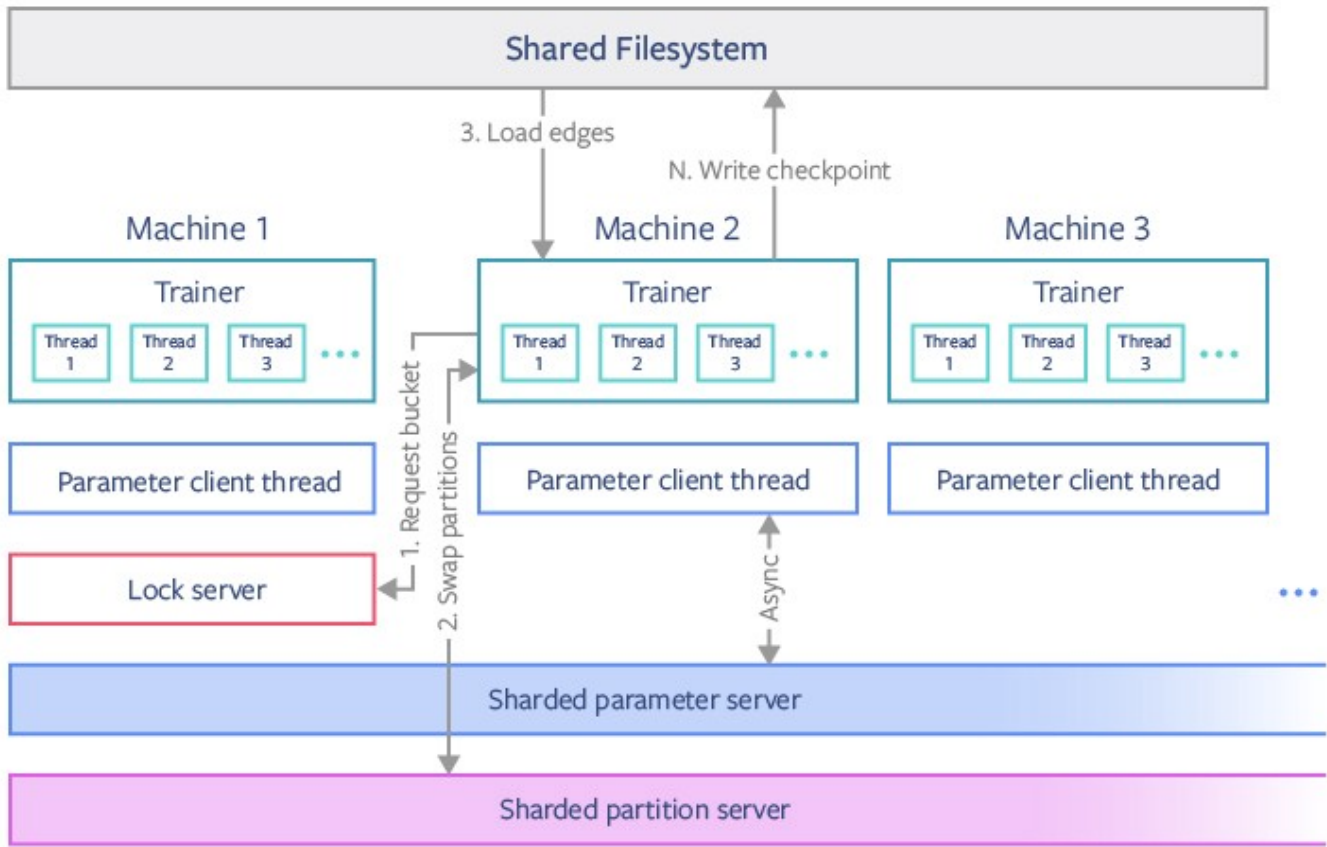
PBG addresses some of the shortcomings of traditional graph embedding methods by partitioning the graph structure into randomly divided into P partitions that are sized so that two partitions can fit in memory. For example, if an edge has a source in partition p1 and destination in partition p2 then it is placed into bucket (p1, p2). In the same model, the graph edges are then divided into P2 buckets based on their source and destination node. Once the nodes and edges are partitioned, training can be performed on one bucket at a time. The training of bucket (p1, p2) only requires the

en To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#),
sti including cookie policy. ×

embedding partition.



Another area in which PBG really innovates is the parallelization and distribution of the training mechanics. PBG uses PyTorch parallelization primitives to implement a distributed training model that leverages the block partition structure illustrated previously. In this model, individual machines coordinate to train on disjoint buckets using a lock server which parcels out buckets to the workers in order to minimize communication between the different machines. Each machine can train the model in parallel using different buckets.



In the previous figure, the Trainer module in machine 2 requests a bucket from the lock server on machine 1, which locks that bucket's partitions. The trainer then saves any partitions that it is no longer using and loads new partitions that it needs to and from the sharded partition servers, at which point it can release its old partitions on the lock server. Edges are then loaded from a shared filesystem, and training occurs on multiple threads without inter-thread synchronization. In a separate thread, a small number

of

pa

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.

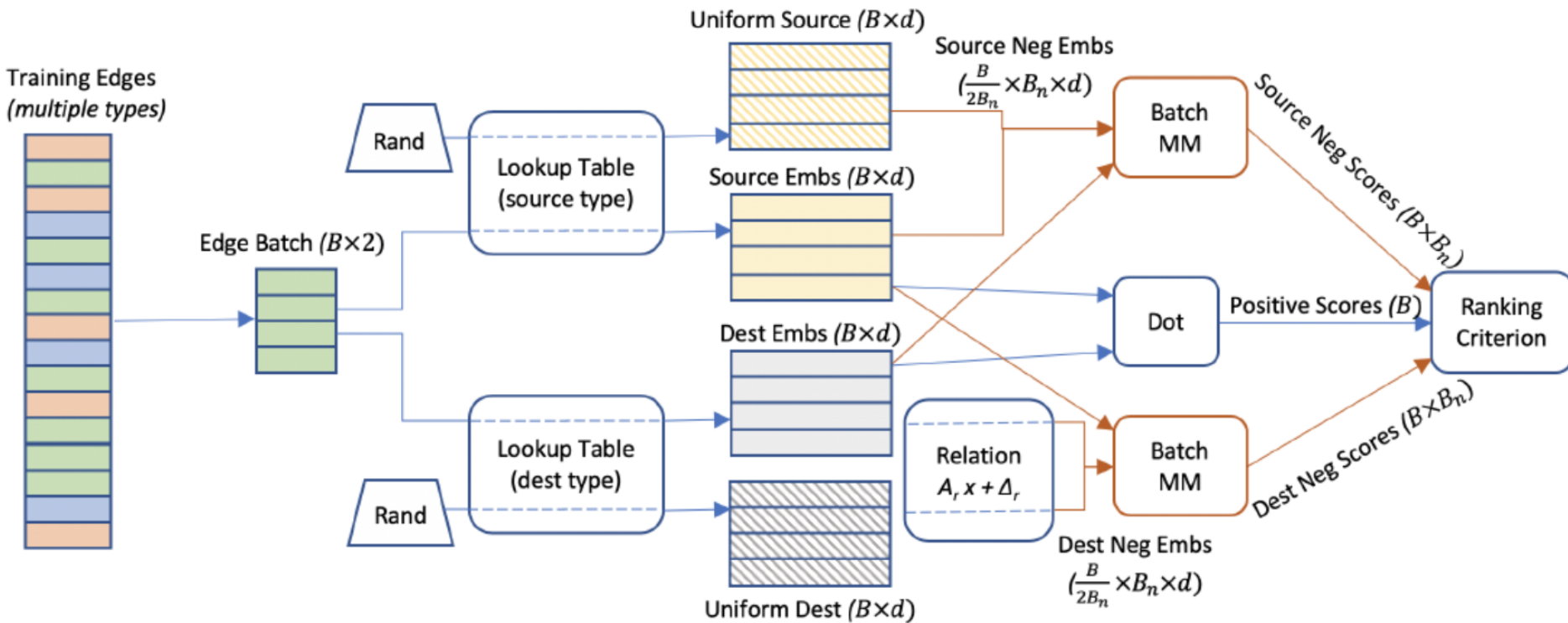
×

d

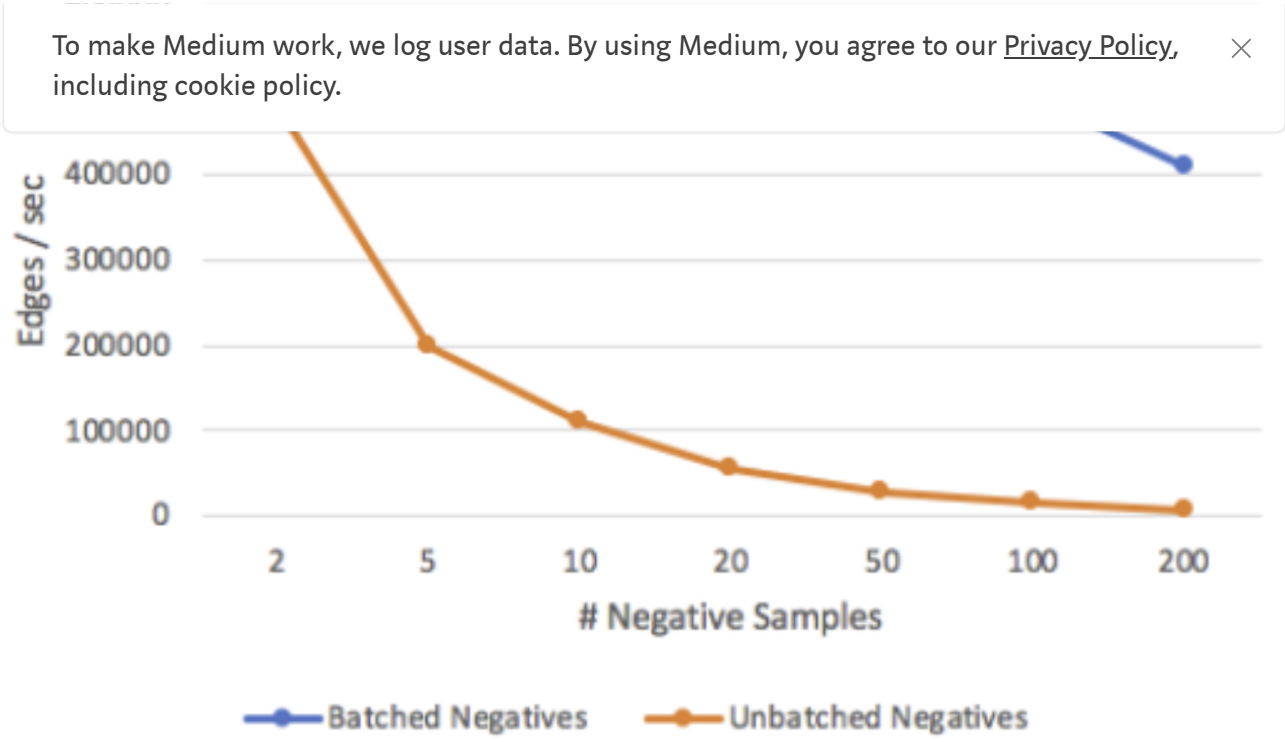
filesystem from the trainers. This model allows a set of P buckets to be parallelized using up to P/2 machines.

One of the indirect innovations of PBG is the use of batched negative sampling techniques. Traditional graph embedding models, construct random “false” edges as negative training examples along with the true positive edges. This significantly speeds training because only a small percentage of weights must be updated with each new sample. However, the negative samples end up introducing a performance overhead in the processing of the graph and end up “corrupting” true edges with random source or destination nodes. PBG introduces a method that reuses a single batch of N random nodes to produce corrupted negative samples for N training edges. In comparison to other embedding methods, this technique allows us to train on many negative examples per true edge at little computational cost.

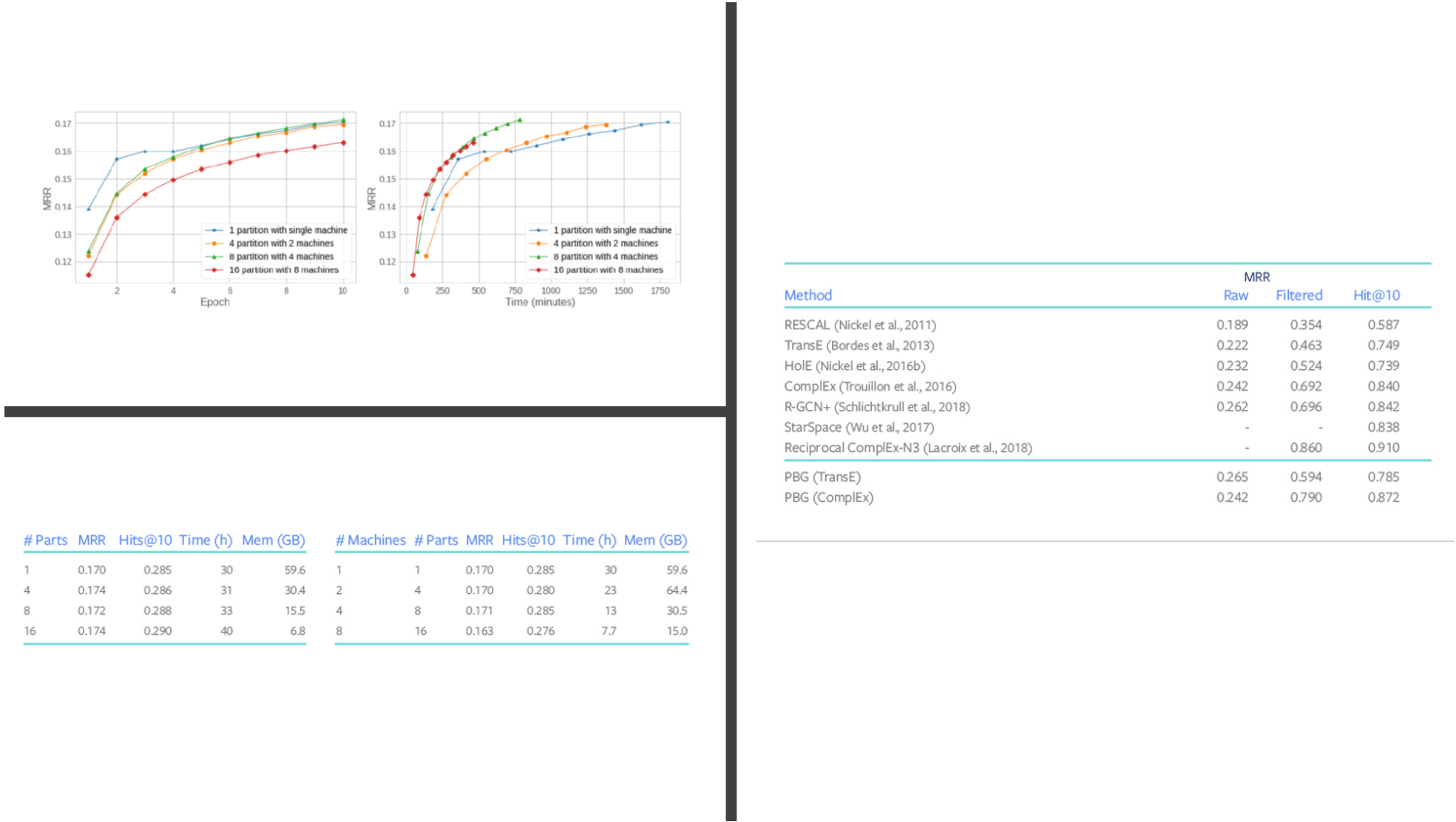
To increase memory efficiency and computational resources on large graphs, PBG leverages a single batch of Bn sampled source or destination nodes to construct multiple negative examples. In a typical setup, PBG takes a batch of B = 1000 positive edges from the training set, and breaks it into chunks of 50 edges. The destination (equivalently, source) embeddings from each chunk is concatenated with 50 embeddings sampled uniformly from the tail entity type. The outer product of the 50 positives with the 200 sampled nodes equates to 9900 negative examples.



The batched negative sampling approach has a direct impact in the speed of the training of the models. Without batching, the speed of training is inversely proportional to the number of negative samples. Batched training improves that equation achieving constant training speed.



Facebook evaluated PGB using different graph datasets such as LiveJournal, Twitter data and YouTube user interaction data. Additionally, PGB was benchmarked using the Freebase knowledge graph, which contains more than 120 million nodes and 2.7 billion edges as well as a smaller subset of the Freebase graph, known as FB15k, which contains 15,000 nodes and 600,000 edges and is commonly used as a benchmark for multi-relation embedding methods. The FB15k experiments showed PGB performing similarly to state of the art graph embedding models. However, when evaluated against the full Freebase dataset, PGB show memory consumptions improves by over 88%.



PBG is one of the first methods that can scale and the training and processing of graph data to structures with billions of nodes and trillions of

ed
an

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.

×

- Machine Learning
- Deep Learning
- Data Science
- Artificial Intelligence
- Investor Labs

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. Upgrade

- About
- Help
- Legal