Demo of Marius: A System for Large-scale Graph Embeddings

Anze Xie
Department of Computer Sciences,
UW-Madison
axie7@wisc.edu

Anders Carlsson
Department of Computer Sciences,
UW-Madison
awcarlsson@wisc.edu

Jason Mohoney
Department of Computer Sciences,
UW-Madison
mohoney2@wisc.edu

Roger Waleffe
Department of Computer Sciences,
UW-Madison
waleffe@wisc.edu

Shanan Peters
Department of Geoscience,
UW-Madison
peters@geology.wisc.edu

Theodoros Rekatsinas
Department of Computer Sciences,
UW-Madison
rekatsinas@wisc.edu

Shivaram Venkataraman
Department of Computer Sciences,
UW-Madison
shivaram@cs.wisc.edu

ABSTRACT

Graph embeddings have emerged as the de facto representation for modern machine learning over graph data structures. The goal of graph embedding models is to convert high-dimensional sparse graphs into low-dimensional, dense and continuous vector spaces that preserve the graph structure properties. However, learning a graph embedding model is a resource intensive process, and existing solutions rely on expensive distributed computation to scale training to instances that do not fit in GPU memory. This demonstration showcases Marius: a new open-source engine for learning graph embedding models over billion-edge graphs on a single machine. Marius is built around a recently-introduced architecture for machine learning over graphs that utilizes pipelining and a novel data replacement policy to maximize GPU utilization and exploit the entire memory hierarchy (including disk, CPU, and GPU memory) to scale to large instances. The audience will experience how to develop, train, and deploy graph embedding models using Marius' configuration-driven programming model. Moreover, the audience will have the opportunity to explore Marius' deployments on applications including link-prediction on WikiKG90M and reasoning queries on a paleobiology knowledge graph. Marius is available as open source software at https://marius-project.org.

PVLDB Reference Format:
Anze Xie, Anders Carlsson, Jason Mohoney, Roger Waleffe, Shanan Peters, Theodoros Rekatsinas, and Shivaram Venkataraman. Demo of Marius: A System for Large-scale Graph Embeddings. PVLDB, 14(12): 2759-2762, 2021.
doi:10.14778/3476311.3476338

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://marius-project.org/.

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476338

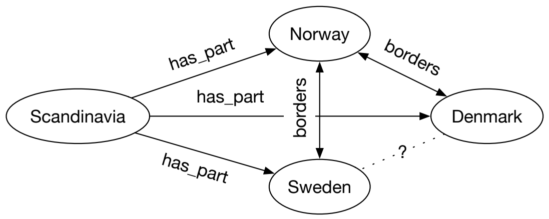


Figure 1: A sample knowledge graph.

1 INTRODUCTION

Generating machine learning models on graph structured data is an important problem that has applications across many domains ranging from social networking [14] to reasoning over knowledge bases [8] to drug discovery [3]. For example, consider a knowledge graph where nodes are entities and the edge types indicate the relationship between them. A data scientist might want to use such a graph to predict links between entities based on the graph structure (see example in Figure 1).

A major step in learning models over graphs is to convert them into continuous vector representations, or *embeddings*, while ensuring that the structural properties of the graph are maintained. Learning such embeddings is a resource-intensive process; not only are graph access patterns are irregular, but the storage required for the embedding vectors can also be high (100s of GB) [9]. Hence, efficient access and storage of these vectors is a unique challenge for scalable graph embedding systems. While existing work in PyTorch BigGraph [9] and DGL-KE [15] have proposed using distributed training, this often leads to wasted resources with low utilization. To address this challenge, we recently introduced Marius [10], a scalable graph embedding learning system. The goal of this demonstration is to showcase the Marius engine.

The Marius Engine We design Marius to make efficient use of system resources to scale graph learning on a single machine. To handle the aforementioned challenges, we design a pipelined training approach in Marius. Graph embeddings are maintained on local disks and we pipeline the steps of reading data from disk to CPU memory, moving data from CPU to GPU memory, and computing updates to the embeddings on the GPU, to improve resource utilization.

Further, Marius minimizes the amount of data read from disk by using an approach inspired by database buffer management. We split the vertex embeddings into partitions and maintain an in-memory partition buffer that is used by the training pipeline. Marius introduces a novel replacement scheme that uses knowledge of the buffer size and the graph traversal order to minimize the number of disk I/Os.

The above two design aspects enable Marius to learn *graph embeddings for billion-edge graphs using just a single GPU on a single machine*.

Marius Demonstration In this paper, we propose a demonstration of Marius to showcase the usability and utility of our system for data scientists. We plan to do this through three scenarios for the conference attendees.

- **Using Marius**: We have developed two ways in which Marius can be deployed in graph learning pipelines. First, Marius introduces a *configuration-driven programming model*, where data scientists can directly train on popular models, loss functions, and datasets without writing any code. In addition, Marius can be used as a Python library that allows more advanced use cases such as developing custom graph embedding models. In the demonstration scenario, the audience will learn how to use Marius via both the aforementioned programming models.
- **Training on real-world graphs**: We plan to showcase how Marius can be used to train embedding models on various real-world graphs including knowledge bases, social graphs, and biological and chemical networks. We will also show how Marius can adapt to CPU or GPU based training based on the hardware available and the dataset in hand.
- **End-to-end Knowledge Retrieval**: Finally, we also plan to combine the above aspects to show how Marius can enable end-to-end knowledge retrieval for real-world scientific tasks. We plan to use a knowledge graph induced from a repository of scientific publications and will also show how the embeddings trained using Marius can be used to power reasoning over a paleobiology knowledge graph.

Audience Takeaways In summary, our demonstration will introduce the audience to scalable graph embedding systems. From a technical point of view, the audience will learn how pipelining and Marius' novel data replacement policy [10] can maximize GPU utilization and thus enable learning graph embeddings for billion-edge graphs with a single machine. From a practical point of view, the audience will learn how to use the open-sourced Marius system and experience how graph embeddings can be applied to a diverse array of applications.

2 MARIUS OVERVIEW

We briefly review graph embeddings and then present an overview of the architecture and programming interface of Marius.

Graph Embeddings A graph *embedding* is a d-dimensional vector representation corresponding to each node (and/or edge-type) in a graph [6]. *Score functions*, i.e., functions that capture the structural properties of the graph are used to form the loss functions used for training [1, 12, 13].

Graph embeddings are commonly used for *link prediction*, where the similarity of two node vector representations is used to infer the existence of a missing edge in a graph. For example, in the knowledge graph in Figure 1 we can use the vector representation of Sweden and the relation embedding for *borders* to predict the existence of the edge Sweden $\xrightarrow{borders}$ Denmark, marked with a question mark in the figure.

The Marius Architecture The training of large scale graphs is limited by data movement costs and CPU/GPU memory sizes [10]. To address the first limitation, Marius uses a pipelined architecture (Figure 2) to minimize wait times for memory operations, and CPU-GPU transfer. The second limitation is addressed by partitioning embedding parameters on disk while buffering partitions in CPU memory, and iterating over the partitions in an ordering which minimizes the number of swaps to disk.

We next explain our design by outlining one iteration of training in Figure 2. The Marius pipeline forms partial batches on the CPU by loading edges and node embeddings from storage (*Load*) and places the partial batch onto a queue, where a worker will transfer it to the GPU (*Transfer*). A full batch is formed by loading the relation embeddings on the GPU, and the forward and backward pass of the model is computed to obtain gradient updates for the embeddings (*Compute*). The relation embeddings are updated on the GPU, and the node embedding parameters are placed on a queue for transfer back to the CPU (*Transfer*). Once transfer is complete, the updates will be applied to the underlying storage (*Update*).

The Marius API We designed Marius' API with two goals in mind: 1) enable out-of-the-box development and deployment of graph embedding pipelines, and 2) seamless integration with Python code that defines analytical pipelines using other popular machine learning libraries such as PyTorch.

The main API objects are shown in Table 1. The `MariusConfig` object is used to set all program parameters: the dataset, storage, execution hardware, model, hyperparameters, pipeline configuration etc. The `MariusConfig` object is used to create a `MariusDataSet` and a `MariusModel`, the former providing a batching interface to the dataset, and the latter describing the model computation. The `MariusModel` is an abstract class that can be extended to implement custom models. The `MariusTrainer` will then train to the specified number of epochs using the pipeline, and the `MariusEvaluator` can be used to run an evaluation task on the trained embeddings.

For out-of-the-box deployment Marius adopts a configuration-based programming paradigm. There are currently over 90 configurable parameters divided into nine main sections, including model, storage, training, pipelining and evaluation (Figure 3).

We have also included detailed documentation for all of these fields, allowing users to exert a high degree of control to tweak their training process as they see fit. Also, to ease manual efforts, all configurable parameters are automatically set to the recommended defaults if not explicitly defined in the configuration file. The use of configuration based programming allows users to run Marius without ever having to write a line of code. This no-code paradigm dramatically expands the potential user base, allowing researchers with little programming experience to effectively use Marius.

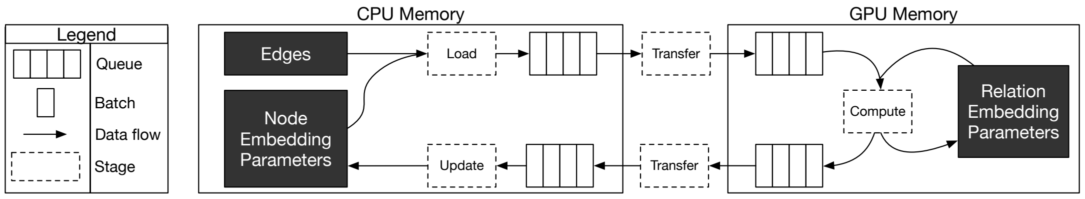


Figure 2: Marius training pipeline: Node embedding parameters are loaded from storage to form a partial batch which is transferred to the GPU. A full batch is formed by loading the parameters residing in GPU memory and the forward and backward pass is computed. Gradient updates are transferred back to CPU memory and applied to storage.

```
[general]
device=GPU

[model]
embedding_size=100
decoder=DistMult

[training]
batch_size=10000
negatives=512
num_epochs=50

[path]
base_directory=training_data/
train_edges=training_set.pt
```

Figure 3: Marius configuration file.

```python
import marius as m
import torch

class trans(m.relation_operator):
    def forward(node_embs, rel_embs):
        return node_embs + rel_embs

class L2(m.comparator):
    def forward(src_embs, dst_embs):
        return torch.cdist(src_embs, dst_embs, p=2)

class TransE(m.model):
    def __init__():
        self.decoder = m.LinkPredictionDecoder()
        self.decoder.relation_operator = trans()
        self.decoder.comparator = L2()
```

Figure 4: Creating a custom embedding model with Marius.

Table 1: Main components of the Marius API.

API Object	Example Usage	Description
MariusConfig	`config = marius.Config(file)`	A dictionary of the Marius configuration options.
MariusDataSet	`train_set = marius.DataSet(config)`	Maintains training state and provides a batching interface to the data set.
MariusModel	`model = marius.Model(config)`	Defines the computation of the model and is extensible to custom models.
MariusTrainer	`trainer = marius.Trainer(train_set, model)`	Manages execution of the training process.

For advanced users, Marius can be imported as a Python library. While Marius is implemented in C++, the bindings we developed can be used seamlessly alongside popular machine learning libraries such as PyTorch. Developers can use this API to write scripts that define the entire embedding training process. Users can also implement their own additional custom models in Python and use them for the training process after setting associated model decoder parameter (Figure 4).

3 DEMONSTRATION SCENARIOS

Our demonstration has into three parts: 1) teach the audience to develop Marius programs, 2) showcase large scale graph learning over Marius, and 3) demonstrate the utility of learned graph embeddings in downstream applications related to knowledge graph reasoning.

3.1 Developing Marius Programs

The first part of our demonstration will expose the audience to the core elements of Marius' API: the config-based and the Python-based programming schemes. Specifically, the audience will learn: 1) how to develop and deploy Marius pipelines using Marius' no-code paradigm, 2) how to use the Python API to define custom models, and 3) about the data converters supported by Marius different formats including TSV, CSV, and Parquet.

In this part, we will focus on the WordNet [5] graph and guide the users around a sample config file. We will discuss the main parts of the file and demonstrate changing a few of the customizable parameters, highlighting frequently used variables such as the device type, the embedding dimension, and the decoder. We will also showcase the ease of using Marius to train graph embeddings, issue nearest neighbor queries and evaluate the training output over the WordNet graph from the command line with the config file just created by users.

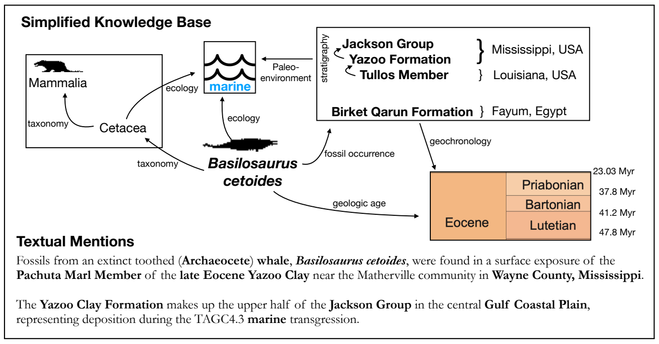
Next, we will demonstrate the extensibility of Marius by diving into more advanced use cases with the Python API. In our example, we will teach the audience how to define their own custom graph embedding model in Python. We will focus on implementing the popular TransE model [1] from scratch, initializing Marius through Python with this custom model and training for the desired number of epochs.

Finally, we will highlight the data converters that Marius is equipped with. We will show an example of downloading and converting one of the 31 popular datasets that the system comes included with out of the box. In our inference example, we will show how the postprocessing methods can transform embeddings to formats such as PyTorch tensors. This step facilitates using these embeddings in downstream PyTorch models. We will showcase the ease of using postprocessed embeddings for link prediction in WordNet graph.

3.2 Large Scale Graph Learning over Marius

The second part of the demonstration will showcase how Marius can be applied over large-scale real-world graphs to support standard graph learning tasks, i.e., *link prediction* and *node classification* over these graphs. For this part we will use two datasets from the Large-Scale Graph ML Competition at the 2021 KDD cup [7]. Specifically, we will use 1) WikiKG90M, a knowledge graph with 87 million nodes and 504 million edges that occupies 139 GB with embedding size of 400; 2) MAG240M, an academic graph for classifying papers in different subject areas with 244 million nodes and 1.7 billion edges that occupies 390 GB with embedding size of 400.

For both graphs, the audience will assume the role of advanced Marius users and will learn to optimize training throughput by configuring Marius training pipeline to use the entire memory hierarchy efficiently.



Fossils from an extinct toothed (**Archaeocete**) **whale**, ***Basilosaurus cetoides***, were found in a surface exposure of the **Pachuta Marl Member** of the **late Eocene Yazoo Clay** near the Matherville community in **Wayne County, Mississippi**.

The **Yazoo Clay Formation** makes up the upper half of the **Jackson Group** in the central **Gulf Coastal Plain**, representing deposition during the TAGC4.3 **marine** transgression.

Figure 5: A sample of the paleobiology text corpus and the corresponding knowledge graph we will use to demonstrate Marius over scientific data applications.

Finally, we will demonstrate how the embeddings trained by Marius can be used to perform the benchmark link prediction and node classification tasks of the KDD 2021 cup for the above two data sets [7]. For WikiKG90M, the task is to perform link prediction (knowledge graph completion), while for MAG240M the task is to predict the primary subject areas of nodes corresponding to arXiv papers (multi-class classification). For this last part, we will demonstrate these tasks on pre-conputed Marius embeddings of the two graphs with standard models such as TransE [1] and Dismult [13].

3.3 Marius Models in Science: Species

The third part of the demonstration will showcase how Marius' ability scale to large-scale inputs can enable unique types of analysis. We will demonstrate how Marius can be used to jointly embed a text corpus and a knowledge graph (treating the co-occurrence of important terms in the text as a mode of the input graph) to enable discovery of and reasoning about new relational facts.

We will showcase the application of Marius in paleobiology—a joint project in collaboration with colleagues in the Department of Geoscience. The goal of this project is to *jointly* analyze a text corpus of scientific publications and a corresponding knowledge graph[2, 4, 11]. An illustrative example of this corpus and knowledge graph is shown in Figure 5. The corpus we consider is constructed by combining scientific publications collected by the xDD [1] and COSMOS [2] platforms at the University of Wisconsin-Madison and the Paleobiology Database [3].

The audience will experience how Marius can help paleobiologists answer two types of reasoning queries: 1) *Analogical reasoning* queries that seek to find a common relational system between two entities; for example the query "*Basilosaurus* $\xrightarrow{\text{geochronology}}$ Eocene; *Pakicetus* $\xrightarrow{\text{geochronology}}$?" seeks to find the **geochronology** of the genus *Pakicetus* by using the known facts on the geochronology of *Basilosaurus* as an example. 2) *Relation extraction* queries, where new facts that are mentioned in the text, but are not explicit in the knowledge graph, need to be answered. For instance, in the example of Figure 5 we use the Marius embeddings to extract the relation "*Basilosaurus cetoides* $\xrightarrow{\text{fossil occurrence}}$ Pachuta Marl Member $\xrightarrow{\text{stratigraphy}}$ Yazoo Formation" which is not explicit in the knowledge graph. We will demonstrate a set of such queries to the audience over the paleobiology corpus corresponding to a part of the Paleobiology Database.

[1]https://xdd.wisc.edu
[2]https://cosmos.wisc.edu
[3]https://paleobiodb.org

ACKNOWLEDGMENTS

This work was supported by NSF under grant 1815538 and DARPA under grant ASKE HR00111990013. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of DARPA or the U.S. Government. This work is also supported by the National Science Foundation grant CNS-1838733, a Facebook faculty research award and by the Office of the Vice Chancellor for Research and Graduate Education at UW-Madison with funding from the Wisconsin Alumni Research Foundation.

REFERENCES

[1] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems* 26 (2013), 2787–2795.
[2] KD Burke, JW Williams, MA Chandler, AM Haywood, DJ Lunt, and BL Otto-Bliesner. 2018. Pliocene and Eocene provide best analogs for near-future climates. *Proceedings of the National Academy of Sciences* 115, 52 (2018), 13288–13293.
[3] Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. 2018. The rise of deep learning in drug discovery. *Drug Discovery Today* 23, 6 (2018), 1241–1250. https://doi.org/10.1016/j.drudis.2018.01.039
[4] RA Close, Roger BJ Benson, EE Saupe, ME Clapham, and RJ Butler. 2020. The spatial structure of Phanerozoic marine animal diversity. *Science* 368, 6489 (2020).
[5] Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database.* Bradford Books.
[6] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. *CoRR* abs/1607.00653 (2016). arXiv:1607.00653
[7] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs. *arXiv preprint arXiv:2103.09430* (2021).
[8] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2021. A Survey on Knowledge Graphs: Representation, Acquisition and Applications. arXiv:2002.00388
[9] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-biggraph: A large-scale graph embedding system. *arXiv preprint arXiv:1903.12287* (2019).
[10] Jason Mohoney, Roger Waleffe, Yiheng Xu, Theodoros Rekatsinas, and Shivaram Venkataraman. 2021. Marius: Learning Massive Graph Embeddings on a Single Machine. In *OSDI 2021.*
[11] Carl J Reddin, Paulina S Nätscher, Ádám T Kocsis, Hans-Otto Pörtner, and Wolfgang Kiessling. 2020. Marine clade sensitivities to climate change conform across timescales. *Nature Climate Change* 10, 3 (2020), 249–253.
[12] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *Proceedings of The 33rd International Conference on Machine Learning*, Vol. 48. 2071–2080.
[13] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575* (2014).
[14] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD 2018.* 974–983.
[15] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. 2020. DGL-KE: Training knowledge graph embeddings at scale. *arXiv preprint arXiv:2004.08532* (2020).